

---

# **hyperbox-brain**

***Release 0.1.1***

**Thanh Tung Khuat and Bogdan Gabrys**

**Jun 12, 2023**



## USER GUIDES

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Installation . . . . .	3
1.2	Features . . . . .	5
1.3	Available models . . . . .	9
1.4	Quickstart . . . . .	11
1.5	Contributing . . . . .	13
1.6	About hyperbox-brain . . . . .	23
<b>2</b>	<b>API Reference</b>	<b>25</b>
2.1	utility functions . . . . .	25
2.2	base . . . . .	55
2.3	mixed-data learners . . . . .	68
2.4	batch learners . . . . .	91
2.5	ensemble learners . . . . .	99
2.6	incremental learners . . . . .	131
2.7	multigranular learners . . . . .	151
2.8	Tutorials . . . . .	163
<b>3</b>	<b>Indices and tables</b>	<b>373</b>
	<b>Bibliography</b>	<b>375</b>
	<b>Python Module Index</b>	<b>379</b>
	<b>Index</b>	<b>381</b>





A scikit-learn compatible hyperbox-based machine learning library in Python.



## INTRODUCTION

**hyperbox-brain** is a Python open source toolbox implementing hyperbox-based machine learning algorithms built on top of scikit-learn and is distributed under the GPL-3.0 license.

The project was started in 2018 by Prof. [Bogdan Gabrys](#) and Dr. [Thanh Tung Khuat](#) at the Complex Adaptive Systems Lab - The University of Technology Sydney. This project is a core module aiming to the formulation of explainable life-long learning systems in near future.

If you use hyperbox-brain, please use this BibTeX entry:

```
@article{khga23,  
  title={hyperbox-brain: A Python toolbox for hyperbox-based machine learning},  
  ↪algorithms},  
  author={Khuat, Thanh Tung and Gabrys, Bogdan},  
  journal={SoftwareX},  
  volume={23},  
  pages={101425},  
  year={2023},  
  url={https://doi.org/10.1016/j.softx.2023.101425},  
  publisher={Elsevier}  
}
```

### 1.1 Installation

- *Dependencies*
- *conda installation*
- *pip installation*
- *From source*
  - *Using conda*
  - *Using pip*
- *Testing*

### 1.1.1 Dependencies

Hyperbox-brain requires:

- Python ( $\geq 3.6$ )
- Scikit-learn ( $\geq 0.24.0$ )
- NumPy ( $\geq 1.14.6$ )
- SciPy ( $\geq 1.1.0$ )
- joblib ( $\geq 0.11$ )
- threadpoolctl ( $\geq 2.0.0$ )
- Pandas ( $\geq 0.25.0$ )

---

Hyperbox-brain plotting capabilities (i.e., functions start with `show_` or `draw_`) require Matplotlib ( $\geq 2.2.3$ ) and Plotly ( $\geq 4.10.0$ ). For running the examples Matplotlib  $\geq 2.2.3$  and Plotly  $\geq 4.10.0$  are required. A few examples require pandas  $\geq 0.25.0$ .

### 1.1.2 conda installation

You need a working conda installation. Get the correct miniconda for your system from [here](#).

To install hyperbox-brain, you need to use the conda-forge channel:

```
conda install -c conda-forge hyperbox-brain
```

We recommend to use a [conda virtual environment](#).

### 1.1.3 pip installation

If you already have a working installation of numpy, scipy, pandas, matplotlib, and scikit-learn, the easiest way to install hyperbox-brain is using pip:

```
pip install -U hyperbox-brain
```

Again, we recommend to use a [virtual environment](#) for this.

### 1.1.4 From source

If you would like to use the most recent additions to hyperbox-brain or help development, you should install hyperbox-brain from source.



## Using conda

To install hyperbox-brain from source using conda, proceed as follows:

```
git clone https://github.com/UTS-CASLab/hyperbox-brain.git
cd hyperbox-brain
conda env create
source activate hyperbox-brain
pip install .
```

## Using pip

For pip, follow these instructions instead:

```
git clone https://github.com/UTS-CASLab/hyperbox-brain.git
cd hyperbox-brain
# create and activate a virtual environment
pip install -r requirements.txt
# install hyperbox-brain version for your system (see below)
pip install .
```

### 1.1.5 Testing

After installation, you can launch the test suite from outside the source directory (you will need to have `pytest >= 5.0.1` installed):

```
pytest hbbrain
```

## 1.2 Features

The **hyper-box brain** toolbox has the following main characteristics:

### 1.2.1 Types of input variables

The hyperbox-brain library separates learning models for continuous variables only and mixed-attribute data.

### 1.2.2 Incremental learning

Incremental (online) learning models are created incrementally and are updated continuously. They are appropriate for big data applications where real-time response is an important requirement. These learning models generate a new hyperbox or expand an existing hyperbox to cover each incoming input pattern.

### 1.2.3 Agglomerative learning

Agglomerative (batch) learning models are trained using all training data available at the training time. They use the aggregation of existing hyperboxes to form new larger sized hyperboxes based on the similarity measures among hyperboxes.

### 1.2.4 Ensemble learning

Ensemble models in the hyperbox-brain toolbox build a set of hyperbox-based learners from a subset of training samples or a subset of both training samples and features. Training subsets of base learners can be formed by stratified random subsampling, resampling, or class-balanced random subsampling. The final predicted results of an ensemble model are an aggregation of predictions from all base learners based on a majority voting mechanism. An interesting characteristic of hyperbox-based models is resulting hyperboxes from all base learners or decision trees can be merged to formulate a single model. This contributes to increasing the explainability of the estimator while still taking advantage of strong points of ensemble models.

### 1.2.5 Multigranularity learning

Multi-granularity learning algorithms can construct classifiers from multiresolution hierarchical granular representations using hyperbox fuzzy sets. This algorithm forms a series of granular inferences hierarchically through many levels of abstraction. An attractive characteristic of these classifiers is that they can maintain a high accuracy in comparison to other fuzzy min-max models at a low degree of granularity based on reusing the knowledge learned from lower levels of abstraction.

### 1.2.6 Learning from both labelled and unlabelled data

One of the exciting features of learning algorithms for the general fuzzy min-max neural network is the capability of creating classification boundaries among known classes and clustering data and representing them as hyperboxes in the case that labels are not available. Unlabelled hyperboxes is then possible to be labelled on the basis of the evidence of next incoming input samples. As a result, the GFMMNN models have the ability to learn from the mixed labelled and unlabelled datasets in a native way.

### 1.2.7 Ability to directly process missing data

Learning algorithms for the general fuzzy min-max neural network supported by the library may classify inputs with missing data directly without the need for replacing or imputing missing values as in other classifiers.

### 1.2.8 Continual learning ability of new classes

Incremental learning algorithms of hyperbox-based models in the **hyperbox-brain** library can grow and accommodate new classes of data without retraining the whole classifier. Incremental learning algorithms themselves can generate new hyperboxes to represent clusters of new data with potentially new class labels both in the middle of normal training procedure and in the operating time where training has been finished. This property is a key feature for smart life-long learning systems.

### 1.2.9 Data editing and pruning approaches

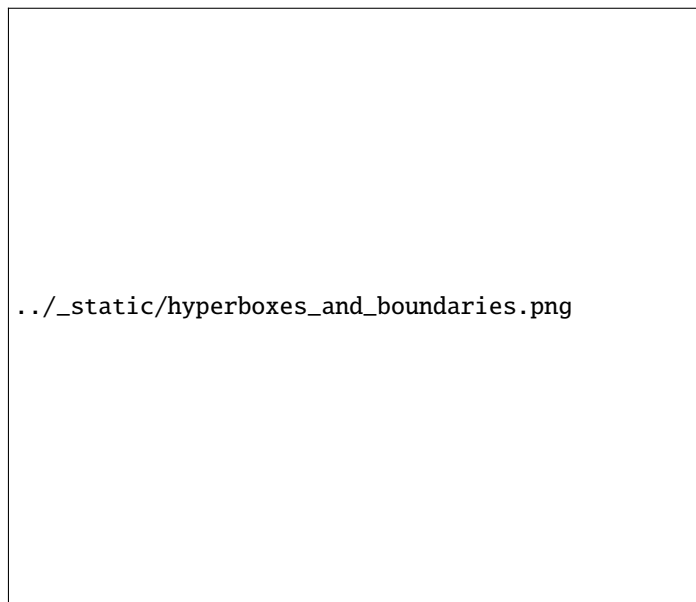
By combining the repeated cross-validation methods provided by scikit-learn and hyperbox-based learning algorithms, evidence from training multiple models can be deployed for identifying which data points from the original training set or the hyperboxes from the generated multiple models should be retained and those that should be edited out or pruned before further processing.

### 1.2.10 Scikit-learn compatible estimators

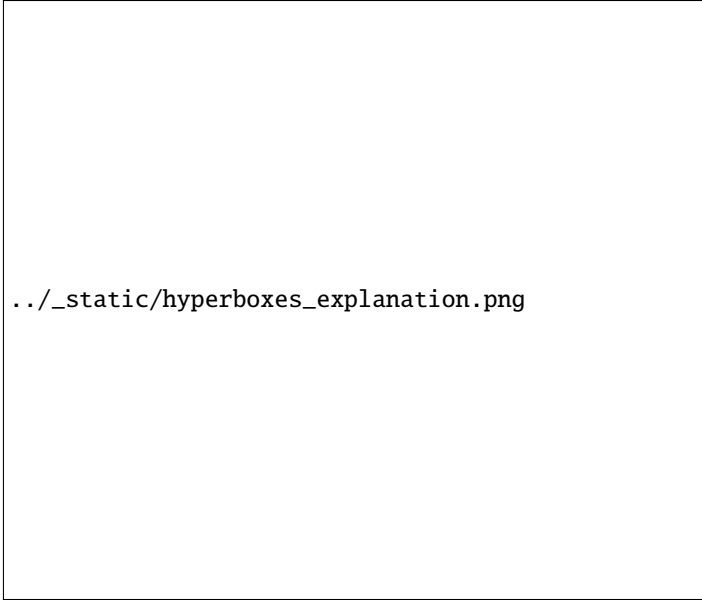
The estimators in hyperbox-brain is compatible with the well-known scikit-learn toolbox. Therefore, it is possible to use hyperbox-based estimators in scikit-learn [pipelines](#), scikit-learn hyperparameter optimizers (e.g., [grid search](#) and [random search](#)), and scikit-learn model validation (e.g., [cross-validation scores](#)). In addition, the hyperbox-brain toolbox can be used within hyperparameter optimisation libraries built on top of scikit-learn such as [hyperopt](#).

### 1.2.11 Explainability of predicted results

The hyperbox-brain library can provide the explanation of predicted results via visualisation. This toolbox provides the visualisation of existing hyperboxes and the decision boundaries of a trained hyperbox-based model if input features are two-dimensional features:




For two-dimensional data, the toolbox also provides the reason behind the class prediction for each input sample by showing representative hyperboxes for each class which join the prediction process of the trained model for an given input pattern:



`../_static/hyperboxes_explanation.png`

For input patterns with two or more dimensions, the hyperbox-brain toolbox uses a parallel coordinates graph to display representative hyperboxes for each class which join the prediction process of the trained model for an given input pattern:



`../_static/parallel_coord_explanation.PNG`

### 1.2.12 Easy to use

Hyperbox-brain is designed for users with any experience level. Learning models are easy to create, setup, and run. Existing methods are easy to modify and extend.

### 1.2.13 Jupyter notebooks

The learning models in the hyperbox-brain toolbox can be easily retrieved in notebooks in the Jupyter or JupyterLab environments.

In order to display plots from hyperbox-brain within a [Jupyter Notebook](#) we need to define the proper matplotlib backend to use. This can be performed by including the following magic command at the beginning of the Notebook:

```
%matplotlib notebook
```

[JupyterLab](#) is the next-generation user interface for Jupyter, and it may display interactive plots with some caveats. If you use JupyterLab then the current solution is to use the [jupyter-matplotlib](#) extension:

```
%matplotlib widget
```

[Examples](#) regarding how to use the classes and functions in the hyperbox-brain toolbox have been written under the form of Jupyter notebooks.

## 1.3 Available models

The following table summarises the supported hyperbox-based learning algorithms in this toolbox.

Model	Feature type	Model type	Learning type	Implementation	Example	References
EIOL-GFMM	Mixed	Single	Instance-incremental	ExtendedImprovedOnlineGFMM	Note-book	<sup>1</sup>
Freq-Cat-Onln-GFMM	Mixed	Single	Batch-incremental	FreqCatOnlineGFMM	Note-book	<sup>2</sup>
OneHot-Onln-GFMM	Mixed	Single	Batch-incremental	OneHotOnlineGFMM	Note-book	Page 11, 2
Onln-GFMM	Continuous	Single	Instance-incremental	OnlineGFMM	Note-book	<sup>3, 4</sup>
IOL-GFMM	Continuous	Single	Instance-incremental	ImprovedOnlineGFMM	Note-book	<sup>5</sup> , Page 11, 4
FMNN	Continuous	Single	Instance-incremental	FMNNClassifier	Note-book	<sup>6</sup>
EFMNN	Continuous	Single	Instance-incremental	EFMNNClassifier	Note-book	<sup>7</sup>
KNEFMNN	Continuous	Single	Instance-incremental	KNEFMNNClassifier	Note-book	<sup>8</sup>
RFMNN	Continuous	Single	Instance-incremental	RFMNNClassifier	Note-book	<sup>9</sup>
AGGLO-SM	Continuous	Single	Batch	AgglomerativeLearningGFMM	Note-book	<sup>10</sup> , Page 11, 4
AGGLO-2	Continuous	Single	Batch	AccelAgglomerativeLearning-GFMM	Note-book	Page 11, 10, Page 11, 4
MRHGRC	Continuous	Granularity	Multi-Granular learning	MultiGranularGFMM	Note-book	<sup>11</sup>
Decision-level Bagging of hyperbox-based learners	Continuous	Combination	Ensemble	DecisionCombinationBagging	Note-book	<sup>12</sup>
Decision-level Bagging of hyperbox-based learners with hyper-parameter optimisation	Continuous	Combination	Ensemble	DecisionCombinationCrossVal-Bagging	Note-book	
Model-level Bagging of hyperbox-based learners	Continuous	Combination	Ensemble	ModelCombinationBagging	Note-book	Page 11, 12
Model-level Bagging of hyperbox-based learners with hyper-parameter optimisation	Continuous	Combination	Ensemble	ModelCombinationCrossVal-Bagging	Note-book	
Random hyperboxes	Continuous	Combination	Ensemble	RandomHyperboxesClassifier	Note-book	<sup>13</sup>
Random hyperboxes with hyper-parameter optimisation for base learners	Continuous	Combination	Ensemble	CrossValRandomHyperboxesClassifier	Note-book	

### 1.3.1 References

## 1.4 Quickstart

### 1.4.1 Training a model

Simply use an estimator by initialising, fitting and predicting:

```
from sklearn.datasets import load_iris
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from hbbrain.numerical_data.incremental_learner.onln_gfmm import OnlineGFMM

# Load dataset
X, y = load_iris(return_X_y=True)
# Normalise features into the range of [0, 1] because hyperbox-based models only work in
↳ a unit range
scaler = MinMaxScaler()
scaler.fit(X)
X = scaler.transform(X)
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Training a model
clf = OnlineGFMM(theta=0.1).fit(X_train, y_train)
# Make prediction
y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print(f'Accuracy = {acc * 100: .2f}%')
```

### 1.4.2 In an sklearn Pipeline

Using hyperbox-based estimators in a `sklearn` Pipeline:

```
from sklearn.datasets import load_iris
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from hbbrain.numerical_data.incremental_learner.onln_gfmm import OnlineGFMM

# Load dataset
X, y = load_iris(return_X_y=True)
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Create a GFMM model
onln_gfmm_clf = OnlineGFMM(theta=0.1)
# Create a pipeline
pipe = Pipeline([
    ('scaler', MinMaxScaler()),
    ('onln_gfmm', onln_gfmm_clf)
])
```

(continues on next page)

- T. T. Khuat and B. Gabrys “An Online Learning Algorithm for a Neuro-Fuzzy Classifier with Mixed-Attribute Data”, ArXiv preprint, arXiv:2009.14670, 2020.

## 1.4. Quickstart

- T. T. Khuat and B. Gabrys “An in-depth comparison of methods handling mixed-attribute data for general fuzzy min-max neural network”, Neurocomputing, vol 464, pp. 175-202, 2021.

(continued from previous page)

```
# Training
pipe.fit(X_train, y_train)
# Make prediction
acc = pipe.score(X_test, y_test)
print(f'Testing accuracy = {acc * 100: .2f}%')
```

### 1.4.3 Hyper-parameter search

This example shows how to use hyperbox-based models with `sklearn` random search:

```
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split
from hbbrain.numerical_data.ensemble_learner.random_hyperboxes import_
↳ RandomHyperboxesClassifier
from hbbrain.numerical_data.incremental_learner.onln_gfmm import OnlineGFMM

# Load dataset
X, y = load_breast_cancer(return_X_y=True)
# Normalise features into the range of [0, 1] because hyperbox-based models only work in_
↳ a unit range
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Initialise search ranges for hyper-parameters
parameters = {'n_estimators': [20, 30, 50, 100, 200, 500],
              'max_samples': [0.2, 0.3, 0.4, 0.5, 0.6],
              'max_features' : [0.2, 0.3, 0.4, 0.5, 0.6],
              'class_balanced' : [True, False],
              'feature_balanced' : [True, False],
              'n_jobs' : [4],
              'random_state' : [0],
              'base_estimator__theta' : np.arange(0.05, 0.61, 0.05),
              'base_estimator__gamma' : [0.5, 1, 2, 4, 8, 16]}
# Init base learner. This example uses the original online learning algorithm to train a_
↳ GFMM classifier
base_estimator = OnlineGFMM()
# Using random search with only 40 random combinations of parameters
random_hyperboxes_clf = RandomHyperboxesClassifier(base_estimator=base_estimator)
clf_rd_search = RandomizedSearchCV(random_hyperboxes_clf, parameters, n_iter=40, cv=5,
↳ random_state=0)
# Fit model
clf_rd_search.fit(X_train, y_train)
# Print out best scores and hyper-parameters
print("Best average score = ", clf_rd_search.best_score_)
print("Best params: ", clf_rd_search.best_params_)
# Using the best model to make prediction
best_gfmm_rd_search = clf_rd_search.best_estimator_
```

(continues on next page)



(continued from previous page)

```
y_pred_rd_search = best_gfmm_rd_search.predict(X_test)
acc_rd_search = accuracy_score(y_test, y_pred_rd_search)
print(f'Accuracy (random-search) = {acc_rd_search * 100: .2f}%')
```

## 1.5 Contributing

We welcome contributions from the community. Here you will find information to start contributing to **hyperbox-brain**.

The project is hosted on <https://github.com/UTS-CASLab/hyperbox-brain>

### Our community, our values

We are a community based on openness and friendly, politeness, constructive discussions.

We aspire to treat everybody equally, and value their contributions. We are particularly seeking people from underrepresented backgrounds in Open Source Software and Hyperbox-based Machine Learning in particular to join and contribute their expertise and experience.

Decisions are made based on technical merit and consensus.

Code is not the only way to help the project. Reviewing pull requests, answering questions to help others on mailing lists or issues, organising and running tutorials, working on the website, enhancing the quality of documentation, are all priceless contributions.

We abide by the principles of openness, respect, and consideration of others of the Python Software Foundation: <https://www.python.org/psf/codeofconduct/>

In case you experience issues using this package, do not hesitate to submit a ticket to the [GitHub issue tracker](#). You are also welcome to post feature requests or pull requests.

### 1.5.1 Ways to contribute

There are various methods to contribute to hyperbox-brain, with the most common ones being contribution of code or documentation to the project. Enhancing the documentation is no less important than enhancing the library itself. If you find a typo in the documentation, or have made improvements, do not hesitate to send an email to the mailing list or preferably submit a GitHub pull request. Full documentation can be found under the doc/ directory.

But there are many other ways to help. In particular helping to *improve, triage, and investigate issues* and *reviewing other developers' pull requests* are very valuable contributions that decrease the burden on the project maintainers.

Another way to contribute is to report issues you're facing, and give a "thumbs up" on issues that others reported and that are relevant to you. It also helps us if you spread the word: reference the project from your blog and articles, link to it from your website, or simply star to say "I use it":

In case a contribution/issue involves changes to the API principles or changes to dependencies or supported versions, it must be essential to submit as a pull-request and send an email to inform the project owner.

## 1.5.2 Submitting a bug report or a feature request

We use GitHub issues to track all bugs and feature requests; feel free to open an issue if you have found a bug or wish to see a feature implemented.

In case you experience issues using this package, do not hesitate to submit a ticket to the [Bug Tracker](#). You are also welcome to post feature requests or pull requests.

It is recommended to check that your issue complies with the following rules before submitting:

- Verify that your issue is not being currently addressed by other [issues](#) or [pull requests](#).
- If you are submitting an algorithm or feature request, please verify the algorithm carefully and discuss it with the governance board.
- If you are submitting a bug report, we strongly encourage you to follow the guidelines in [How to make a good bug report](#).

### How to make a good bug report

When you submit an issue to [Github](#), please do your best to follow these guidelines! This will make it a lot easier to provide you with good feedback:

- The ideal bug report contains a description of how to reproduce this bug via code snippet. By doing this way, anyone can try to reproduce the bug easily (see [this](#) for more details). If your snippet is longer than around 50 lines, please link to a [gist](#) or a github repo.
- If it is not feasible to include a reproducible snippet, please be specific about what **estimators and/or functions are involved and the shape of the data**.
- If an exception is raised, please **provide the full traceback**.
- Please include your **operating system type and version number**, as well as your **Python, hyperbox-brain, hyperbox-brain, joblib, numpy, matplotlib, plotly, and pandas versions**. This information can be found by running the following code snippet:

```
>>> import hbbrain
>>> hbbrain.show_versions()
```

- Please ensure all **code snippets and error messages are formatted in appropriate code blocks**. See [Creating and highlighting code blocks](#) for more details.

## 1.5.3 Contributing code

---

**Note:** To avoid duplicating work, it is highly recommended that you search through the [issue tracker](#) and the [PR list](#). If in doubt about duplicated work, or if you want to work on a non-trivial feature, it's recommended to first open an issue in the [issue tracker](#) to get some feedbacks from core developers.

One easy way to find an issue to work on is by applying the “help wanted” label in your search. This lists all the issues that have been unclaimed so far. In order to claim an issue for yourself, please comment exactly `/take` on it to assign the issue to you.

---

## How to contribute

The best method to contribute to hyperbox-brain is to fork the [main repository](#) on GitHub, then submit a “pull request” (PR).

In the first few steps, we explain how to locally install hyperbox-brain, and how to set up your git repository:

1. [Create an account](#) on GitHub if you do not already have one.
2. Fork the [project repository](#): click on the ‘Fork’ button near the top of the page. This creates a copy of the code under your account on the GitHub user account. For more details on how to fork a repository see [this guide](#).
3. Clone your fork of the hyperbox-brain repo from your GitHub account to your local disk:

```
git clone git@github.com:YourLogin/hyperbox-brain.git # add --depth 1 if your
↪connection is slow
cd hyperbox-brain
```

4. Follow the steps in the [installation from source](#) to build hyperbox-brain in development mode and return to this document.
5. Install the development dependencies:

```
pip install pytest pytest-cov flake8 mypy numpydoc black==22.3.0
```

6. Add the upstream remote. This saves a reference to the main hyperbox-brain repository, which you can use to keep your repository synchronized with the latest changes:

```
git remote add upstream git@github.com:UTS-CASLab/hyperbox-brain.git
```

7. Check that the *upstream* and *origin* remote aliases are configured correctly by running `git remote -v` which should display:

```
origin  git@github.com:YourLogin/hyperbox-brain.git (fetch)
origin  git@github.com:YourLogin/hyperbox-brain.git (push)
upstream  git@github.com:UTS-CASLab/hyperbox-brain.git (fetch)
upstream  git@github.com:UTS-CASLab/hyperbox-brain.git (push)
```

You should now have a working installation of hyperbox-brain, and your git repository properly configured. The next steps now describe the process of modifying code and submitting a PR.

8. Synchronize your main branch with the upstream/main branch, more details on [GitHub Docs](#):

```
git checkout main
git fetch upstream
git merge upstream/main
```

9. Create a feature branch to hold your development changes:

```
git checkout -b my_feature
```

and start making changes. Always use a feature branch. It’s good practice to never work on the main branch!

10. **(Optional)** Install [pre-commit](#) to run code style checks before each commit:

```
pip install pre-commit
pre-commit install
```

pre-commit checks can be disabled for a particular commit with `git commit -n`.

11. Develop the feature on your feature branch on your computer, using Git to do the version control. When you're done editing, add changed files using `git add` and then `git commit`:

```
git add modified_files
git commit
```

to record your changes in Git, then push the changes to your GitHub account with:

```
git push -u origin my_feature
```

12. Follow [these instructions](#) to create a pull request from your fork. This will send an email to the committers. You may want to consider sending an email to the mailing list for more visibility.

It is often helpful to keep your local feature branch synchronized with the latest changes of the main hyperbox-brain repository:

```
git fetch upstream
git merge upstream/main
```

Subsequently, you might need to solve the conflicts. You can refer to the [Git documentation related to resolving merge conflict using the command line](#).

### Learning git:

The [Git documentation](#) and <http://try.github.io> are excellent resources to get started with git, and understanding all of the commands shown here.

## Pull request checklist

Before a PR can be merged, it needs to be approved by two core developers. Please prefix the title of your pull request with [MRG] if the contribution is complete and should be subjected to a detailed review. An incomplete contribution – where you expect to do more work before receiving a full review – should be prefixed [WIP] (to indicate a work in progress) and changed to [MRG] when it matures. WIPs may be useful to: indicate you are working on something to avoid duplicated work, request broad review of functionality or API, or seek collaborators. WIPs often benefit from the inclusion of a [task list](#) in the PR description.

In order to ease the reviewing process, we recommend that your contribution complies with the following rules before marking a PR as [MRG]. The **bolded** ones are especially important:

1. **Give your pull request a helpful title** that summarizes what your contribution does. This title will often become the commit message once merged so it should summarize your contribution for posterity. In some cases “Fix <ISSUE TITLE>” is enough. “Fix #<ISSUE NUMBER>” is never a good title.
2. **Make sure your code passes the tests**. The whole test suite can be run with *pytest*, but it is usually not recommended since it takes a long time. It is often enough to only run the test related to your changes: for example, if you changed something in *hbbrain/mixed\_data/eiol\_gfmm.py*, running the following commands will usually be enough:
  - `pytest hbbrain/mixed_data/eiol_gfmm.py` to make sure the doctest examples are correct.
  - `pytest hbbrain/mixed_data/tests/test_eiol_gfmm.py` to run the tests specific to the file.
  - `pytest hbbrain/mixed_data` to test the whole *mixed\_data* module
  - `pytest docs/api/mixed_data.rst` and `pytest docs/tutorials/mixed_data_learner.rst` to make sure the user guide examples are correct.

For guidelines on how to use `pytest` efficiently, see the [document](#).

3. **Make sure your code is properly commented and documented**, and **make sure the documentation renders properly**. To build the documentation, please refer to our [Documentation](#) guidelines.
4. **Tests are necessary for enhancements to be accepted**. Bug-fixes or new features should be provided with [non-regression tests](#). These tests verify the correct behavior of the fix or feature. In this manner, further modifications on the code base are granted to be consistent with the desired behavior. In the case of bug fixes, at the time of the PR, the non-regression tests should fail for the code base in the main branch and pass for the PR code.
5. Run *black* to auto-format your code.

```
black .
```

See black's [editor integration documentation](#) to configure your editor to run *black*.

6. **Make sure that your PR does not add PEP8 violations**. To check the code that you changed, you can run the following command:

```
git diff upstream/main -u -- "*.py" | flake8 --diff
```

or *make flake8-diff* which should work on unix-like system.

7. Follow the [Coding guidelines](#).
8. When applicable, use the validation tools and scripts in the `hbbrain.utils` submodule. You can add any functions to this submodule if necessary for your implementation.
9. Often pull requests resolve one or more other issues (or pull requests). If merging your pull request means that some other issues/PRs should be closed, you should [use keywords to create link to them](#) (e.g., `Fixes #1234`; multiple issues/PRs are allowed as long as each one is preceded by a keyword). Upon merging, those issues/PRs will automatically be closed by GitHub. If your pull request is simply related to some other issues/PRs, create a link to them without using the keywords (e.g., `See also #1234`).
10. PRs should often substantiate the change, through benchmarks of performance and efficiency or through examples of usage. Examples also illustrate the features and intricacies of the library to users. Have a look at other examples in the [examples](#) directory for reference. Examples should demonstrate why the new functionality is useful in practice and, if possible, compare it to other methods available in hyperbox-brain.
11. New features have some maintenance overhead. We expect PR authors to take part in the maintenance for the code they submit, at least initially. New features need to be illustrated with narrative documentation in the user guide, with small code snippets. If relevant, please also add references in the literature, with PDF links when possible.
12. The user guide should also include expected time and space complexity of the algorithm and scalability, e.g. “this algorithm can scale to a large number of samples > 1000000, but does not scale in dimensionality: `n_features` is expected to be lower than 100”.

You can check for common programming errors with the following tools:

1. Code with a good unittest coverage (at least 80%, better 100%), check with:

```
pip install pytest pytest-cov
pytest --cov hbbrain path/to/tests_for_package
```

2. Run static analysis with *mypy*:

```
mypy hbbrain
```

must not produce new errors in your pull request. Using `# type: ignore` annotation can be a workaround for a few cases that are not supported by mypy, in particular, when importing C or Cython modules on properties with decorators.

### Coding guidelines

The following are some guidelines on how new code should be written for inclusion in hyperbox-brain, and which may be appropriate to adopt in external projects. Certainly, there are special cases and there will be exceptions to these rules. However, following these rules when submitting new code makes the review easier so new code can be integrated in less time.

Uniformly formatted code makes it easier to share code ownership. The hyperbox-brain project tries to closely follow the official Python guidelines detailed in PEP8 that detail how code should be formatted and indented. Please read it and follow it.

In addition, we add the following guidelines:

- Use underscores to separate words in non class names: `n_samples` rather than `nsamples`.
- Avoid multiple statements on one line. Prefer a line return after a control flow statement (if/for).
- Unit tests should use absolute imports, exactly as client code would.
- Please don't use `import *` in any case. It is considered harmful by the official Python recommendations. It makes the code harder to read as the origin of symbols is no longer explicitly referenced, but most important, it prevents using a static analysis tool like `pyflakes` to automatically find bugs in hyperbox-brain.
- Use the `numpy docstring` standard in all your docstrings.

A good example of code that we like can be found [here](#).

### 1.5.4 Documentation

We are happy to accept any sort of documentation: function docstrings, reStructuredText documents (like this one), tutorials, etc. reStructuredText documents live in the source code repository under the `docs/` directory.

You can edit the documentation using any text editor, and then generate the HTML output by typing `make` from the `docs/` directory. Alternatively, `make html` may be used to generate the documentation **with** the example gallery (which takes quite some time). The resulting HTML files will be placed in `_build/html` and are viewable in a web browser.

#### Building the documentation

First, make sure you have [properly installed](#) the development version.

Building the documentation requires installing some additional packages:

```
pip install sphinx sphinx-rtd-theme readthedocs-sphinx-search numpydoc \
            sphinx-gallery hyperbox-brain nbsphinx sphinx-autodocgen \
            pandas IPython
```

To build the documentation, you need to be in the `docs` folder:

```
cd docs
```

In the vast majority of cases, you only need to generate the full web site, without the example gallery:

```
make
```

The documentation will be generated in the `_build/html` directory. To also generate the example gallery you can use:

```
make html
```

This will run all the examples, which takes a while. If you only want to generate a few examples, you can use:

```
EXAMPLES_PATTERN=your_regex_goes_here make html
```

This is particularly useful if you are modifying a few examples.

Set the environment variable `NO_MATHJAX=1` if you intend to view the documentation in an offline setting.

To build the PDF manual, run:

```
make latexpdf
```

### Warning: Sphinx version

While we do our best to have the documentation build under as many versions of Sphinx as possible, the different versions tend to behave slightly differently.

## Guidelines for writing documentation

It is essential to keep a good compromise between mathematical and algorithmic details, and give intuition to the reader on what the algorithm does.

Basically, to elaborate on the above, it is best to always start with a small paragraph with a hand-waving explanation of what the method does to the data. Then, it is very helpful to point out why the feature is useful and when it should be used - the latter also including “big O” ( $O(g(n))$ ) complexities of the algorithm, as opposed to just *rules of thumb*, as the latter can be very machine-dependent. If those complexities are not available, then rules of thumb may be provided instead.

Secondly, a generated figure from an example should then be included to further provide some intuition.

Next, one or two small code examples to show its use can be added.

Next, any math and equations, followed by references, can be added to further the documentation. Not starting the documentation with the maths makes it more friendly towards users that are just interested in what the feature will do, as opposed to how it works “under the hood”.

Finally, follow the formatting rules below to make it consistently good:

- Add “See Also” in docstrings for related classes/functions.
- “See Also” in docstrings should be one line per reference, with a colon and an explanation, for example:

```
See Also
```

```
-----
```

```
SelectKBest : Select features based on the k highest scores.
```

```
SelectNSamples : Select samples based on a false negative rate test.
```

- When documenting the parameters and attributes, here is a list of some well-formatted examples:

```
n_hyperboxes : int, default=10
    The number of hyperboxes generated by the algorithm.

some_param : {'hello', 'goodbye'}, bool or int, default=True
    The parameter description goes here, which can be either a string
    literal (either `hello` or `goodbye`), a bool, or an int. The default
    value is True.

array_parameter : {array-like, sparse matrix} of shape (n_samples, n_features) or ↵
↵(n_samples,)
    This parameter accepts data in either of the mentioned forms, with one
    of the mentioned shapes. The default value is
    `np.ones(shape=(n_samples,))`.

list_param : list of int

typed_ndarray : ndarray of shape (n_samples,), dtype=np.int32

sample_weight : array-like of shape (n_samples,), default=None

multioutput_array : ndarray of shape (n_samples, n_classes) or list of such arrays
```

In general have the following in mind:

1. Use Python basic types.
  2. Use parenthesis for defining shapes: array-like of shape (n\_samples,) or array-like of shape (n\_samples, n\_features)
  3. For strings with multiple options, use brackets: input: {'log', 'squared', 'multinomial'}
  4. 1D or 2D data can be a subset of {array-like, ndarray, sparse matrix, dataframe}. Note that array-like can also be a list, while ndarray is explicitly only a `numpy.ndarray`.
  5. Specify dataframe when “frame-like” features are being used, such as the column names.
  6. When specifying the data type of a list, use `of` as a delimiter: `list of int`. When the parameter supports arrays giving details about the shape and/or data type and a list of such arrays, you can use one of array-like of shape (n\_samples,) or list of such arrays.
  7. When specifying the dtype of an ndarray, use e.g. `dtype=np.int32` after defining the shape: `ndarray of shape (n_samples,), dtype=np.int32`. You can specify multiple dtype as a set: `array-like of shape (n_samples,), dtype={np.float64, np.float32}`. If one wants to mention arbitrary precision, use *integral* and *floating* rather than the Python dtype *int* and *float*. When both *int* and *floating* are supported, there is no need to specify the dtype.
  8. When the default is `None`, `None` only needs to be specified at the end with `default=None`. Be sure to include in the docstring, what it means for the parameter or attribute to be `None`.
- For unwritten formatting rules, try to follow existing good works:
    - When bibliographic references are available with [arxiv](#) or [Digital Object Identifier](#) identification numbers, use the sphinx directives `:arxiv:` or `:doi:`.
    - For “References” in docstrings, see [this document](#).
  - When editing reStructuredText (`.rst`) files, try to keep line length under 80 characters when possible (exceptions include links and tables).



- Do not modify sphinx labels as this would break existing cross references and external links pointing to specific sections in the hyperbox-brain documentation.
- Before submitting your pull request check if your modifications have introduced new sphinx warnings and try to fix them.

### 1.5.5 Issue Tracker Tags

All issues and pull requests on the [GitHub issue tracker](#) should have (at least) one of the following tags:

**Bug / Crash**

Something is happening that clearly shouldn't happen. Wrong results as well as unexpected errors from estimators go here.

**Cleanup / Enhancement**

Improving performance, usability, consistency.

**Documentation**

Missing, incorrect or sub-standard documentations and examples.

**New Feature**

Feature requests and pull requests implementing a new feature.

There are four other tags to help new contributors:

**good first issue**

This issue is ideal for a first contribution to hyperbox-brain. Ask for help if the formulation is unclear. If you have already contributed to hyperbox-brain, look at Easy issues instead.

**Easy**

This issue can be tackled without much prior experience.

**Moderate**

Might need some knowledge of machine learning or the package, but is still approachable for someone new to the project.

**help wanted**

This tag marks an issue which currently lacks a contributor or a PR that needs another contributor to take over the work. These issues can range in difficulty, and may not be approachable for new contributors. Note that not all issues which need contributors will have this tag.

### 1.5.6 Code Review Guidelines

Reviewing code contributed to the project as PRs is a crucial component of hyperbox-brain development. We encourage anyone to start reviewing code of other developers. The code review process is often highly educational for everybody involved. This is particularly appropriate if it is a feature you would like to use, and so can respond critically about whether the PR meets your needs. While each pull request needs to be signed off by two core developers, you can speed up this process by providing your feedback.

---

**Note:** The difference between an objective improvement and a subjective one isn't always clear. Reviewers should recall that code review is primarily about reducing risk in the project. When reviewing code, one should aim at preventing situations which may require a bug fix, a deprecation, or a retraction. Regarding docs: typos, grammar issues and disambiguations are better addressed immediately.

---

Here are a few important aspects that need to be covered in any code review, from high-level questions to a more detailed check-list.

- Do we want this in the library? Is it likely to be used? Do you, as a hyperbox-brain user, like the change and intend to use it? Is it in the scope of hyperbox-brain? Will the cost of maintaining a new feature be worth its benefits?
- Is the code consistent with the API of hyperbox-brain? Are public functions/classes/parameters well named and intuitively designed?
- Are all public functions/classes and their parameters, return types, and stored attributes named according to hyperbox-brain conventions and documented clearly?
- Is any new functionality described in the user-guide and illustrated with examples?
- Is every public function/class tested? Are a reasonable set of parameters, their values, value types, and combinations tested? Do the tests validate that the code is correct, i.e. doing what the documentation says it does? If the change is a bug-fix, is a non-regression test included? Look at [this document](#) to get started with testing in Python.
- Do the tests pass in the continuous integration build? If appropriate, help the contributor understand why tests failed.
- Do the tests cover every line of code (see the coverage report in the build log)? If not, are the lines missing coverage good exceptions?
- Is the code easy to read and low on redundancy? Should variable names be improved for clarity or consistency? Should comments be added? Should comments be removed as unhelpful or extraneous?
- Could the code easily be rewritten to run much more efficiently for relevant settings?
- Is the code backwards compatible with previous versions? (or is a deprecation cycle necessary?)
- Will the new code add any dependencies on other libraries? (this is unlikely to be accepted)
- Does the documentation render properly (see the [Documentation](#) section for more details), and are the plots instructive?

## Communication Guidelines

Reviewing open pull requests (PRs) helps move the project forward. It is a great way to get familiar with the codebase and should motivate the contributor to keep involved in the project.<sup>1</sup>

- Every PR, good or bad, is an act of generosity. Opening with a positive comment will help the author feel rewarded, and your subsequent remarks may be heard more clearly. You may feel good also.
- Begin if possible with the large issues, so the author knows they've been understood. Resist the temptation to immediately go line by line, or to open with small pervasive issues.
- Do not let perfect be the enemy of the good. If you find yourself making many small suggestions that don't fall into the [Code Review Guidelines](#), consider the following approaches:
  - refrain from submitting these;
  - prefix them as “Nit” so that the contributor knows it's OK not to address;
  - follow up in a subsequent PR, out of courtesy, you may want to let the original contributor know.
- Do not rush, take the time to make your comments clear and justify your suggestions.
- You are the face of the project. Bad days occur to everyone, in that occasion you deserve a break: try to take your time and stay offline.

---

<sup>1</sup> Adapted from the numpy [communication guidelines](#).

---

**Important:** This guide line is adapted from [scikit-learn guidelines](#) under the MIT licence.

---

## 1.6 About hyperbox-brain

hyperbox-brain is an open-source machine learning package in Python for hyperbox-based machine learning algorithms. Learning algorithms using hyperboxes as fundamental representational and building blocks are a branch of machine learning methods. These algorithms have enormous potential for high scalability and online adaptation of predictors built using hyperbox data representations to the dynamically changing environments. This library focuses on developing and extending the learning algorithms for a specific type of universal hyperbox-based classifiers, i.e., fuzzy min-max neural networks and general fuzzy min-max neural network.

Hyperboxes can be used to deal with the pattern classification and clustering problems effectively by partitioning the pattern space and assigning a class label or cluster associated with a degree of certainty for each region. Each fuzzy min-max hyperbox is represented by minimum and maximum points together with a fuzzy membership function. The membership function is employed to compute the degree-of-fit of each input sample to a given hyperbox. Meanwhile, the hyperboxes are continuously adjusted during the training process to cover the input patterns. The use of hyperboxes for learning systems can form a core module aiming to build smart adaptive systems and life-long learning systems in the near future.

### 1.6.1 Ecosystem

hyperbox-brain is part of the hyperbox-based machine learning ecosystem. In Python, this library can be used together with pipeline and hyper-parameter optimisers in the [scikit-learn](#) library. This library can be also compatible with other optimisers in Python such as [hyperopt](#) and [Optuna](#).

### 1.6.2 Development team

This library is the result of hyperbox-based machine learning project conducted by the Complex Adaptive Systems in the [University of Technology Sydney](#). Current members of the development team (in alphabetical order):

- Prof. Bogdan Gabrys
- Dr. Thanh Tung Khuat

We also acknowledge the [individual members](#) of the open-source community who have contributed to this project.

### 1.6.3 Citing

If hyperbox-brain has been useful for your research and you would like to cite it in an academic publication, please use the following paper:

```
@article{khga23,
  title={hyperbox-brain: A Python toolbox for hyperbox-based machine learning_
↪ algorithms},
  author={Khuat, Thanh Tung and Gabrys, Bogdan},
  journal={SoftwareX},
  volume={23},
  pages={101425},
  year={2023},
```

(continues on next page)

(continued from previous page)

```
url={https://doi.org/10.1016/j.softx.2023.101425},  
publisher={Elsevier}  
}
```

### 1.6.4 Logo

The hyperbox-brain logo is designed by Thanh Tung Khuat.

## API REFERENCE

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

### 2.1 utility functions

#### 2.1.1 `utils.membership_calc`

Contain all functions supporting for computing fuzzy membership values in various ways.

`hbbrain.utils.membership_calc.asym_similarity_val_one_many_hyperboxes` (*xl*, *xu*, *V*, *W*, *g=1*,  
*asimil\_type='max'*)

Calculate the asymetrical similarity value of a specific hyperbox (lower bound - *xl*, upper bound - *xu*) and hyperboxes having lower and upper bounds stored in two matrix *V* and *W* respectively

##### Parameters

- xl**  
[array-like of shape (n\_features,)] Lower bound of an input hyperbox.
- xu**  
[array-like of shape (n\_features,)] Upper bound of an input hyperbox.
- V**  
[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all minimal points of all existing hyperboxes, in which each row is a minimal point of a hyperbox.
- W**  
[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all maximal points of all existing hyperboxes, in which each row is a maximal point of a hyperbox.
- g**  
[float or ndarray of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each dimension.
- asimil\_type**  
[{'max', 'min'}, optional, default='max'] Type of handling asymmetric similarity matrix.

##### Returns

- b**  
[array-like of shape (n\_hyperboxes,)] Similarity values of the specific hyperbox with all hyperboxes having lower and upper bounds in *V* and *W*.

`hbbrain.utils.membership_calc.bitwise_membership(x_cat, D)`

Compute membership values between categorical features in the input pattern  $X_{cat}$  and all categorical features of existing hyperboxes stored in  $D$ .

**Parameters**

**x\_cat**

[array-like of shape (n\_cat\_features, )] Categorical features of an input pattern. Each feature is represented by an array of one-hot encoded values.

**D**

[array-like of shape (n\_hyperboxes, n\_cat\_features)] A matrix stores all bounds of categorical features for all existing hyperboxes, in which each row stores categorical features of a hyperbox.

**Returns**

**mem\_val**

[array-like of shape (n\_hyperboxes, )] An array stores the degrees of membership from the input pattern to all existing hyperboxes which are computed based on categorical features.

`hbbrain.utils.membership_calc.f_sim_freq_cat_features(x_cat, E, similarity_of_cat_vals)`

Compute similarity values in each categorical dimension between a input categorical features  $x_{cat}$  and each element in the current list of categorical bounds of existing hyperboxes.

**Parameters**

**x\_cat**

[array-like of shape (n\_cat\_features, )] Categorical features of an input pattern.

**E**

[array-like of shape (n\_hyperboxes, n\_cat\_features)] A matrix stores all bounds of categorical features for all existing hyperboxes, in which each row stores categorical features of a hyperbox.

**similarity\_of\_cat\_vals**

[array-like of shape (n\_cat\_features,)] An array stores all similarity values among all pairs of categorical values for each categorical feature index. Each element in this array is a dictionary with keys being a hashed value of two categorical values and values of this dictionary being a similarity value.

**Returns**

**sim\_vals**

[array-like of shape (n\_hyperboxes, n\_cat\_features)] A matrix stores similarity values in each categorical dimension between the input categorical features and all bounds of categorical features of existing hyperboxes.

`hbbrain.utils.membership_calc.get_membership_extended_iol_gfmm_all_classes(Xl, Xu, X_cat, V, W, D, C, g=1, alpha=0.5)`

Return membership values (according to the membership function of the GFMM classifiers) with respect to all class labels between the continuous input patterns stored in two lower and upper bound input matrices  $Xl$  and  $Xu$  while categorical input patterns stored in the matrix  $Xd$  and existing hyperboxes represented by two matrices of minimum and maximum points  $V$  and  $W$  for continuous features and the matrix of categorical features  $D$  together with corresponding class labels in vector  $C$ .

**Parameters**

**Xl**  
[array-like of shape (n\_samples, n\_continuous\_features) or (n\_continuous\_features, )] Lower bounds of input samples.

**Xu**  
[array-like of shape (n\_samples, n\_continuous\_features) or (n\_continuous\_features, )] Upper bounds of input samples.

**X\_cat**  
[array-like of shape (n\_samples, n\_cat\_features) or (n\_cat\_features, )] Categorical bounds of input samples.

**V**  
[array-like of shape (n\_hyperboxes, n\_continuous\_features)] Minimum points of the existing hyperboxes in the trained model.

**W**  
[array-like of shape (n\_hyperboxes, n\_continuous\_features)] Maximum points of the existing hyperboxes in the trained model.

**D**  
[array-like of shape (n\_hyperboxes, n\_cat\_features)] Categorical bound of the existing hyperboxes in the trained model.

**C**  
[array-like of shape (n\_hyperboxes,)] Class labels of all existing hyperboxes corresponding to the values stored in *V*, *W*, and *D*.

**g**  
[float or ndarray of shape (n\_continuous\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each continuous dimension.

**alpha**  
[float, optional, default=0.5] The trade-off weighting factor between the impacts of categorical features and numerical features on the outputs of membership values.

## Returns

**mem\_vals\_matrix**  
[array-like of shape (n\_samples, n\_classes)] Membership values with regard to all class labels for each input sample. Each row is a vector of membership values. Each column represents an index of a class label sorted in an ascending order of class labels.

**hyperbox\_ids\_matrix**  
[array-like of shape (n\_samples, n\_classes)] Storing the indices of hyperboxes corresponding to membership values for classes.

`hbbrain.utils.membership_calc.get_membership_fmnn_all_classes(X, V, W, C, g=1)`

Return membership values (according to the membership function of the FMNN classifiers) with respect to all class labels between the input patterns stored in the matrix *X* and existing hyperboxes represented by two matrices of minimum and maximum points *V* and *W* together with the corresponding class labels in the vector *C*.

## Parameters

**X**  
[array-like of shape (n\_samples, n\_features) or (n\_features, )] Input samples.

**V**  
[array-like of shape (n\_hyperboxes, n\_features)] Minimum points of the existing hyperboxes in the trained model.

**W**  
[array-like of shape (n\_hyperboxes, n\_features)] Maximum points of the existing hyperboxes in the trained model.

**C**  
[array-like of shape (n\_hyperboxes,)] Class labels of all existing hyperboxes corresponding to the values stored in *V* and *W*.

**g**  
[float or ndarray of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each dimension.

#### Returns

**mem\_vals\_matrix**  
[array-like of shape (n\_samples, n\_classes)] Membership values with respect to all class labels for each input sample. Each row is a vector of membership values. Each column represents an index of a class label sorted in an ascending order of class labels.

**hyperbox\_ids\_matrix**  
[array-like of shape (n\_samples, n\_classes)] Storing the indices of hyperboxes corresponding to membership values for classes.

hbbrain.utils.membership\_calc.**get\_membership\_free\_range\_gfmm\_all\_classes**(*Xl, Xu, V, W, C, g=1*)

Return membership values (according to the membership function of the GFMM classifiers with unbounded range) with respect to all class labels between the input patterns stored in two lower and upper bound input matrices *Xl* and *Xu* and existing hyperboxes represented by two matrices of minimum and maximum points *V* and *W* together with corresponding class labels in vector *C*.

#### Parameters

**Xl**  
[array-like of shape (n\_samples, n\_features) or (n\_features, )] Lower bounds of input samples.

**Xu**  
[array-like of shape (n\_samples, n\_features) or (n\_features, )] Upper bounds of input samples.

**V**  
[array-like of shape (n\_hyperboxes, n\_features)] Minimum points of the existing hyperboxes in the trained model.

**W**  
[array-like of shape (n\_hyperboxes, n\_features)] Maximum points of the existing hyperboxes in the trained model.

**C**  
[array-like of shape (n\_hyperboxes,)] Class labels of all existing hyperboxes corresponding to the values stored in *V* and *W*.

**g**  
[float or ndarray of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each dimension.

#### Returns

**mem\_vals\_matrix**  
[array-like of shape (n\_samples, n\_classes)] Membership values with regard to all class labels



for each input sample. Each row is a vector of membership values. Each column represents an index of a class label sorted in an ascending order of class labels.

#### **hyperbox\_ids\_matrix**

[array-like of shape (n\_samples, n\_classes)] Storing the indices of hyperboxes corresponding to membership values for classes.

`hbbrain.utils.membership_calc.get_membership_freq_cat_gfmm_all_classes(Xl, Xu, X_cat, V, W, E, F, C, similarity_of_cat_vals, g=1)`

Return membership values (according to the membership function of the GFMM classifiers) with respect to all class labels between the input patterns stored in two lower and upper bound matrices for input continuous features *Xl* and *Xu* and two lower and upper bound matrices for input categorical features and existing hyperboxes represented by four matrices of minimum and maximum points for continuous features *V* and *W* and lower and upper bounds for categorical features *E* and *F* together with corresponding class labels in vector *C*.

#### **Parameters**

##### ***Xl***

[array-like of shape (n\_samples, n\_continuous\_features) or (n\_continuous\_features, )] Lower bounds of continuous features of all input samples. If None, there are no continuous features.

##### ***Xu***

[array-like of shape (n\_samples, n\_continuous\_features) or (n\_continuous\_features, )] Lower bounds of continuous features of all input samples. If None, there are no continuous features.

##### ***X\_cat***

[array-like of shape (n\_samples, n\_cat\_features) or (n\_cat\_features, )] Categorical features of all input patterns. If None, there are no categorical features.

##### ***V***

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] Minimum points of all continuous features of the existing hyperboxes in the trained model. If None, there are no continuous features.

##### ***W***

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] Maximum points of all continuous features of the existing hyperboxes in the trained model. If None, there are no continuous features.

##### ***E***

[array-like of shape (n\_hyperboxes, n\_cat\_features)] Lower bounds of all categorical features of the existing hyperboxes in the trained model. If None, there are no categorical features.

##### ***F***

[array-like of shape (n\_hyperboxes, n\_cat\_features)] Upper bounds of all categorical features of the existing hyperboxes in the trained model. If None, there are no categorical features.

##### ***C***

[array-like of shape (n\_hyperboxes,)] Class labels of all existing hyperboxes corresponding to the values stored in *V*, *W*, and *E*, *F*.

#### **similarity\_of\_cat\_vals**

[array-like of shape (n\_cat\_features,)] An array stores all similarity values among all pairs of categorical values for each categorical feature index. Each element in this array is an dictionary with keys being a hashed value of two categorical values and values of this dictionary being a similarity value.

**g**  
[float or ndarray of shape (n\_continuous\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each continuous dimension.

#### Returns

##### **mem\_vals\_matrix**

[array-like of shape (n\_samples, n\_classes)] Membership values with regard to all class labels for each input sample. Each row is a vector of membership values. Each column represents an index of a class label sorted in an ascending order of class labels.

##### **hyperbox\_ids\_matrix**

[array-like of shape (n\_samples, n\_classes)] Storing the indices of hyperboxes corresponding to membership values for classes.

hbbrain.utils.membership\_calc.**get\_membership\_gfmm\_all\_classes**(*Xl, Xu, V, W, C, g=1*)

Return membership values (according to the membership function of the GFMM classifiers) with respect to all class labels between the input patterns stored in two lower and upper bound input matrices *Xl* and *Xu* and existing hyperboxes represented by two matrices of minimum and maximum points *V* and *W* together with corresponding class labels in vector *C*.

#### Parameters

##### **Xl**

[array-like of shape (n\_samples, n\_features) or (n\_features, )] Lower bounds of input samples.

##### **Xu**

[array-like of shape (n\_samples, n\_features) or (n\_features, )] Upper bounds of input samples.

##### **V**

[array-like of shape (n\_hyperboxes, n\_features)] Minimum points of the existing hyperboxes in the trained model.

##### **W**

[array-like of shape (n\_hyperboxes, n\_features)] Maximum points of the existing hyperboxes in the trained model.

##### **C**

[array-like of shape (n\_hyperboxes,)] Class labels of all existing hyperboxes corresponding to the values stored in *V* and *W*.

##### **g**

[float or ndarray of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each dimension.

#### Returns

##### **mem\_vals\_matrix**

[array-like of shape (n\_samples, n\_classes)] Membership values with regard to all class labels for each input sample. Each row is a vector of membership values. Each column represents an index of a class label sorted in an ascending order of class labels.

##### **hyperbox\_ids\_matrix**

[array-like of shape (n\_samples, n\_classes)] Storing the indices of hyperboxes corresponding to membership values for classes.

hbbrain.utils.membership\_calc.**get\_membership\_onehot\_gfmm\_all\_classes**(*Xl, Xu, Xd, V, W, D, C, g=1*)

Return membership values (according to the membership function of the GFMM classifiers) with respect to all class labels between the input patterns stored in two lower and upper bound input matrices  $Xl$  and  $Xu$  and existing hyperboxes represented by two matrices of minimum and maximum points  $V$  and  $W$  together with corresponding class labels in vector  $C$ .

#### Parameters

##### $Xl$

[array-like of shape (n\_samples, n\_continuous\_features) or (n\_continuous\_features, )] Lower bounds of continuous features of all input samples. If None, there are no continuous features.

##### $Xu$

[array-like of shape (n\_samples, n\_continuous\_features) or (n\_continuous\_features, )] Lower bounds of continuous features of all input samples. If None, there are no continuous features.

##### $Xd$

[array-like of shape (n\_samples, n\_cat\_features) or (n\_cat\_features, )] Bounds of categorical features of all input patterns. If None, there are no categorical features.

##### $V$

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] Minimum points of all continuous features of the existing hyperboxes in the trained model. If None, there are no continuous features.

##### $W$

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] Maximum points of all continuous features of the existing hyperboxes in the trained model. If None, there are no continuous features.

##### $D$

[array-like of shape (n\_hyperboxes, n\_cat\_features)] Bounds of all categorical features of the existing hyperboxes in the trained model. If None, there are no categorical features.

##### $C$

[array-like of shape (n\_hyperboxes,)] Class labels of all existing hyperboxes corresponding to the values stored in  $V$ ,  $W$ , and  $D$ .

##### $g$

[float or ndarray of shape (n\_continuous\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each continuous dimension.

#### Returns

##### **mem\_vals\_matrix**

[array-like of shape (n\_samples, n\_classes)] Membership values with regard to all class labels for each input sample. Each row is a vector of membership values. Each column represents an index of a class label sorted in an ascending order of class labels.

##### **hyperbox\_ids\_matrix**

[array-like of shape (n\_samples, n\_classes)] Storing the indices of hyperboxes corresponding to membership values for classes.

`hbbrain.utils.membership_calc.membership_cat_feature_eiol_gfmm( $x_{cat}$ ,  $D$ )`

Compute membership degrees between input categorical features and all bounds of categorical features of existing hyperboxes for the extended improved online learning algorithm of general fuzzy min-max neural network.

#### Parameters

**x\_cat**

[array-like of shape (n\_cat\_features, )] Categorical features of an input pattern.

**D**

[array-like of shape (n\_hyperboxes, n\_cat\_features)] A matrix stores all bounds of categorical features for all existing hyperboxes, in which each row stores a categorical features bound for a hyperbox. Each element  $d_{ij} \in D$  is a set of symbolic values with their cardinalities for the  $j$ -th categorical dimension of the hyperbox  $B_i$ . For example,  $d_{i1} = \{apple : 5, orange : 1\}$  means that the first categorical feature of the hyperbox  $B_i$  contains 5 values of apple and 1 value of orange.

**Returns****b**

[array-like of shape (n\_hyperboxes, .)] An array stores the degrees of membership from the input pattern to all existing hyperboxes which are computed based on categorical features.

`hbbrain.utils.membership_calc.membership_func_extended_iol_gfmm(xl, xu, x_cat, V, W, D, g=1, alpha=0.5)`

Compute fuzzy membership values between an input pattern and a list of existing hyperboxes of a general fuzzy min-max neural network with mixed-attribute data.

---

**Note:** This function provides the degrees of membership  $b$  of an input pattern  $x$  (in form of upper bound  $xu$  and lower bound  $xl$  for continuous features and categorical features  $x\_cat$ ) with respect to the existing hyperboxes represented by minimal points  $V$  and maximal points  $W$  for continuous features and the bound  $D$  for categorical features. The sensitivity parameter  $g$  regulates how fast the membership values decrease when an input continuous pattern is separated from hyperbox core. The parameter  $alpha$  is the trade-off factor between impacts of continuous features and categorical features on the output of membership values. Each element  $d_{ij} \in D$  is a set of symbolic values with their cardinalities for the  $j$ -th categorical dimension of the hyperbox  $B_i$ . For example,  $d_{i1} = \{apple : 5, orange : 1\}$  means that the first categorical feature of the hyperbox  $B_i$  contains 5 values of apple and 1 value of orange.

---

**Parameters****xl**

[array-like of shape (n\_continuous\_features,)] Lower bound of continuous features of an input pattern.

**xu**

[array-like of shape (n\_continuous\_features,)] Upper bound of continuous features of an input pattern.

**x\_cat**

[array-like of shape (n\_cat\_features, )] Categorical features of an input pattern.

**V**

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] A matrix stores all minimal points for continuous features of all existing hyperboxes, in which each row is a minimal point of a hyperbox.

**W**

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] A matrix stores all maximal points for continuous features of all existing hyperboxes, in which each row is a maximal point of a hyperbox.

**D**

[array-like of shape (n\_hyperboxes, n\_cat\_features)] A matrix stores a special structure for

categorical features of all existing hyperboxes. Each element in  $D$  stores a set of symbolic values with their cardinalities for the  $j$ -th categorical dimension of a given hyperbox.

**g**

[float or ndarray of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each continuous dimension.

**alpha**

[float, optional, default=0.5] The trade-off weighting factor between the impacts of categorical features and numerical features on the outputs of membership values.

#### Returns

**b**

[array-like of shape (n\_hyperboxes,)] Degrees of membership of the input pattern  $x=[x_l, x_u, x_{cat}]$  corresponding to each hyperbox in the current list of existing hyperboxes.

`hbbrain.utils.membership_calc.membership_func_fmnn(x, V, W, g=1)`

Compute fuzzy membership values between an input pattern and a list of existing hyperboxes of a fuzzy min-max neural network and its improved versions.

For more details regarding how to calculate fuzzy membership values, please refer to the publication [1].

---

**Note:** This function provides the degrees of membership  $b$  of an input pattern  $x$  with respect to the existing hyperboxes described by minimal points  $V$  and maximal points  $W$ . The sensitivity parameter  $g$  regulates how fast the membership values decrease when an input pattern is separated from hyperbox core.

---

#### Parameters

**x**

[array-like of shape (n\_features,)] An input pattern.

**V**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all minimal points of all existing hyperboxes, in which each row is a minimal point of a hyperbox.

**W**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all maximal points of all existing hyperboxes, in which each row is a maximal point of a hyperbox.

**g**

[float or ndarray of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each dimension.

#### Returns

**b**

[array-like of shape (n\_hyperboxes,)] Degrees of membership of the input pattern  $X=[X_l, X_u]$  corresponding to each hyperbox in the current list of existing hyperboxes.

## References

[1]

`hbbrain.utils.membership_calc.membership_func_free_range_gfmm(xl, xu, V, W, g=1)`

Compute fuzzy membership values between an input pattern and a list of existing hyperboxes of a general fuzzy min-max neural network. This membership function does not require the coordinates located in the range of [0, 1].

---

**Note:** This function provides the degrees of membership  $b$  of an input pattern  $x$  (in form of upper bound  $xu$  and lower bound  $xl$ ) with respect to the existing hyperboxes described by minimal points  $V$  and maximal points  $W$ . The sensitivity parameter  $g$  regulates how fast the membership values decrease when an input pattern is separated from hyperbox core.

---

### Parameters

**xl**

[array-like of shape (n\_features,)] Lower bound of an input pattern.

**xu**

[array-like of shape (n\_features,)] Upper bound of an input pattern.

**V**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all minimal points of all existing hyperboxes, in which each row is a minimal point of a hyperbox.

**W**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all maximal points of all existing hyperboxes, in which each row is a maximal point of a hyperbox.

**g**

[float or ndarray of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each dimension.

### Returns

**b**

[array-like of shape (n\_hyperboxes,)] Degrees of membership of the input pattern  $X=[Xl, Xu]$  corresponding to each hyperbox in the current list of existing hyperboxes.

`hbbrain.utils.membership_calc.membership_func_freq_cat_gfmm(xl, xu, x_cat, V, W, E, F, similarity_of_cat_vals, g=1)`

Compute the membership values between an input pattern with respect to all hyperboxes (including continuous and categorical features). The membership values for categorical features is computed based on the occurrence frequency values of different class labels with regards to each categorical values in each categorical feature.

For more details regarding how to calculate fuzzy membership values, please refer to the publications [1] and [2].

### Parameters

**xl**

[array-like of shape (n\_continuous\_features, )] Lower bounds of input continuous features of the input pattern.

**xu**

[array-like of shape (n\_continuous\_features, )] Upper bounds of input continuous features of the input pattern.

**x\_cat**

[array-like of shape (n\_cat\_features, )] Categorical features of an input pattern.

**V**

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] Minimum points of continuous features of the existing hyperboxes in the trained model.

**W**

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] Maximum points of continuous features of the existing hyperboxes in the trained model.

**E**

[array-like of shape (n\_hyperboxes, n\_cat\_features)] A matrix stores all lower bounds of categorical features for all existing hyperboxes, in which each row stores a lower categorical features bound for a hyperbox.

**F**

[array-like of shape (n\_hyperboxes, n\_cat\_features)] A matrix stores all upper bounds of categorical features for all existing hyperboxes, in which each row stores a upper categorical features bound for a hyperbox.

**similarity\_of\_cat\_vals**

[array-like of shape (n\_cat\_features,)] An array stores all similarity values among all pairs of categorical values for each categorical feature index. Each element in this array is an dictionary with keys being a hashed value of two categorical values and values of this dictionary being a similarity value.

**g**

[float or ndarray of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each continuous dimension.

**Returns****b**

[array-like of shape (n\_hyperboxes,)] Degrees of membership of the input pattern  $X=[Xl, Xu]$  corresponding to each hyperbox in the current list of existing hyperboxes.

**References**

[1], [2]

`hbbrain.utils.membership_calc.membership_func_gfmm(xl, xu, V, W, g=1)`

Compute fuzzy membership values between an input pattern and a list of existing hyperboxes of a general fuzzy min-max neural network.

For more details regarding how to calculate fuzzy membership values, please refer to the publications [1] and [2].

---

**Note:** This function provides the degrees of membership  $b$  of an input pattern  $x$  (in form of upper bound  $xu$  and lower bound  $xl$ ) with respect to the existing hyperboxes described by minimal points  $V$  and maximal points  $W$ . The sensitivity parameter  $g$  regulates how fast the membership values decrease when an input pattern is separated from hyperbox core.

---

**Parameters****xl**

[array-like of shape (n\_features,)] Lower bound of an input pattern.

**xu**

[array-like of shape (n\_features,)] Upper bound of an input pattern.

**V**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all minimal points of all existing hyperboxes, in which each row is a minimal point of a hyperbox.

**W**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all maximal points of all existing hyperboxes, in which each row is a maximal point of a hyperbox.

**g**

[float or ndarray of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each dimension.

**Returns****b**

[array-like of shape (n\_hyperboxes,)] Degrees of membership of the input pattern  $X=[Xl, Xu]$  corresponding to each hyperbox in the current list of existing hyperboxes.

**References**

[1], [2]

`hbbrain.utils.membership_calc.membership_func_onehot_gfmm(xl, xu, xd, V, W, D, g=1)`

Compute membership values between an input pattern of which continuous features are represented by the lower bound  $xl$  and the upper bound  $xu$  while categorical features are presented by the bound  $xd$  and all existing hyperboxes with lower and upper bounds stored in  $V$  and  $W$  and the categorical bound stored in  $D$ .

**Parameters****xl**

[array-like of shape (n\_continuous\_features,)] Lower bound of continuous features of an input pattern.

**xu**

[array-like of shape (n\_continuous\_features,)] Upper bound of continuous features of an input pattern.

**xd**

[array-like of shape (n\_cat\_features,)] Categorical features of an input pattern.

**V**

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] A matrix stores all minimal points of continuous features for all existing hyperboxes, in which each row is a minimal point of a hyperbox.

**W**

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] A matrix stores all maximal points of continuous features for all existing hyperboxes, in which each row is a maximal point of a hyperbox.

**D**

[array-like of shape (n\_hyperboxes, n\_cat\_features)] A matrix stores all bounds of categorical features for all existing hyperboxes, in which each row contains the bound of a hyperbox.

**g**

[float or ndarray of shape (n\_continuous\_features,), optional, default=1] A sensitivity pa-



parameter describing the speed of decreasing of the membership function in continuous dimensions.

### Returns

**b**

[array-like of shape (n\_hyperboxes,)] Degrees of membership of the input pattern  $x=[x_l, x_u, x_d]$  corresponding to each hyperbox in the current list of existing hyperboxes.

`hbbrain.utils.membership_calc.membership_function_freq_cat(x_cat, E, F, similarity_of_cat_vals)`

Compute membership degrees between input categorical features and all lower and upper bounds of categorical features of existing hyperboxes.

### Parameters

**x\_cat**

[array-like of shape (n\_cat\_features,)] Categorical features of an input pattern.

**E**

[array-like of shape (n\_hyperboxes, n\_cat\_features)] A matrix stores all lower bounds of categorical features for all existing hyperboxes, in which each row stores a lower categorical features bound for a hyperbox.

**F**

[array-like of shape (n\_hyperboxes, n\_cat\_features)] A matrix stores all upper bounds of categorical features for all existing hyperboxes, in which each row stores a upper categorical features bound for a hyperbox.

**similarity\_of\_cat\_vals**

[array-like of shape (n\_cat\_features,)] An array stores all similarity values among all pairs of categorical values for each categorical feature index. Each element in this array is a dictionary with keys being a hashed value of two categorical values and values of this dictionary being a similarity value.

### Returns

**b**

[array-like of shape (n\_hyperboxes,)] An array stores the degrees of membership from the input pattern to all existing hyperboxes which are computed based on categorical features.

`hbbrain.utils.membership_calc.n_cat_features_containing_bit_one(v)`

This function is to count number of categorical features in v in which there is at least one bit 1

## 2.1.2 utils.adjust\_hyperbox

The `hbbrain.utils.adjust_hyperbox` submodule implements various functions for hyperbox adjustment, e.g., hyperbox overlap test, overlap resolving, and hyperbox contraction.

`hbbrain.utils.adjust_hyperbox.hyperbox_contraction_efmnn(V, W, case_contraction, id_extended_box, id_tested_box, alpha=1e-05)`

Adjust the coordinates of two hyperboxes for overlap resolving corresponding to nine overlap test cases.

### Parameters

**V**

[array-like of shape (n\_hyperboxes, n\_features)] Lower bounds (minimum points) of all existing hyperboxes in the trained model.

**W**

[array-like of shape (n\_hyperboxes, n\_features)] Upper bounds (maximum points) of all existing hyperboxes in the trained model.

**case\_contraction**

[a list of two elements] This is a special struct which is the outcomes of the `hyperbox_overlap_test_efmnn` function to determine the overlap test case and corresponding overlapped dimension.

**id\_extended\_box**

[int] id\_extended\_boxex of the extended hyperbox which needs to test for overlap.

**id\_tested\_box**

[int] id\_extended\_boxex of the hyperbox to test for overlap with the extended hyperbox.

**alpha**

[float] A very small value is used to avoid the overlap between two hyperboxes after contraction.

### Returns

**Vout**

[array-like of shape (n\_hyperboxes, n\_features)] Lower bounds (minimum points) of all existing hyperboxes with two hyperboxes adjusted.

**Wout**

[array-like of shape (n\_hyperboxes, n\_features)] Upper bounds (maximum points) of all existing hyperboxes with two hyperboxes adjusted.

`hbbrain.utils.adjust_hyperbox.hyperbox_contraction_fmnn(V, W, case_contraction, id_extended_box, id_tested_box, alpha=1e-05)`

Adjust the coordinates of two hyperboxes for overlap resolving.

### Parameters

**V**

[array-like of shape (n\_hyperboxes, n\_features)] Lower bounds (minimum points) of all existing hyperboxes in the trained model.

**W**

[array-like of shape (n\_hyperboxes, n\_features)] Upper bounds (maximum points) of all existing hyperboxes in the trained model.

**case\_contraction**

[a list of two elements] This is a special struct which is the outcomes of the [`hyperbox\_overlap\_test\_fmnn\(\)`](#) function to determine the overlap test case and corresponding overlapped dimension.

**id\_extended\_box**

[int] id\_extended\_boxex of the extended hyperbox which needs to test for overlap.

**id\_tested\_box**

[int] id\_extended\_boxex of the hyperbox to test for overlap with the extended hyperbox.

**alpha**

[float, optional, default=0.00001] A very small value is used to avoid the overlap between two hyperboxes after contraction.

### Returns

**Vout**

[array-like of shape (n\_hyperboxes, n\_features)] Lower bounds (minimum points) of all existing hyperboxes with two hyperboxes adjusted.

**Wout**

[array-like of shape (n\_hyperboxes, n\_features)] Upper bounds (maximum points) of all existing hyperboxes with two hyperboxes adjusted.

`hbbrain.utils.adjust_hyperbox.hyperbox_contraction_freq_cat_gfmm(Ei, Fi, case_contraction)`

Perform hyperbox contraction in categorical features for a given hyperbox.

**Parameters****Ei**

[array-like of shape (n\_cat\_features,)] Lower bounds for categorical features of the hyperbox which need to do contraction.

**Fi**

[array-like of shape (n\_cat\_features,)] Upper bounds for categorical features of the hyperbox which need to do contraction.

**case\_contraction**

[a list of two elements] This is a special struct which is the outcomes of the [hyperbox\\_overlap\\_test\\_freq\\_cat\\_gfmm\(\)](#) function to determine the overlap test case and corresponding overlapped dimension for categorical features.

**Returns****E\_out**

[array-like of shape (n\_cat\_features, )] Lower bounds for categorical features of the hyperboxes contracted.

**F\_out**

[array-like of shape (n\_cat\_features, )] Upper bounds for categorical features of the hyperboxes contracted.

`hbbrain.utils.adjust_hyperbox.hyperbox_contraction_rfmnn(V, W, C, ids_parent_box, id_child_box, overlap_dim, scale=0.001)`

Adjusting or splitting min-max points of overlapping clusters in the refined fuzzy min-max neural network classifier. The detailed information of this procedure can be found in [1].

**Parameters****V**

[array-like of shape (n\_hyperboxes, n\_features)] Lower bounds (minimum points) of all existing hyperboxes in the trained model.

**W**

[array-like of shape (n\_hyperboxes, n\_features)] Upper bounds (minimum points) of all existing hyperboxes in the trained model.

**C**

[array-like of shape (n\_hyperboxes,)] Class labels of all existing hyperboxes.

**ids\_parent\_box**

[array-like of shape (n\_parent\_hyperboxes,)] List of indices of parent hyperboxes fully containing the child hyperbox.

**id\_child\_box**

[int] Index of a child hyperbox fully contained in parent hyperboxes

**overlap\_dim**

[array-like of shape (n\_parent\_hyperboxes,)] The overlapped dimensions between parent hyperboxes and the child hyperbox need to make contraction.

**scale**

[float, optional, default=0.001] A buffer value is used to avoid overlap on the edges between hyperboxes after contraction

**Returns****Vout**

[array-like of shape (n\_hyperboxes, n\_features)] Lower bounds (minimum points) of all existing hyperboxes after doing contraction.

**Wout**

[array-like of shape (n\_hyperboxes, n\_features)] Upper bounds (minimum points) of all existing hyperboxes after doing contraction.

**Cout**

[array-like of shape (n\_hyperboxes,)] Class labels of all existing hyperboxes after doing contraction.

**References**

[1]

hbbrain.utils.adjust\_hyperbox.**hyperbox\_overlap\_test\_efmn**(V, W, id\_extended\_box, id\_tested\_box, X)

Check the overlap of two input hyperboxes

**Parameters****V**

[array-like of shape (n\_hyperboxes, n\_features)] Lower bounds (minimum points) of all existing hyperboxes in the trained model.

**W**

[array-like of shape (n\_hyperboxes, n\_features)] Upper bounds (maximum points) of all existing hyperboxes in the trained model.

**id\_extended\_box**

[int] Index of the extended hyperbox which needs to test for overlap.

**id\_tested\_box**

[int] Index of the hyperbox to test for overlap with the extended hyperbox.

**X**

[array-like of shape (n\_features, )] Current input sample leads to the extension of the existing hyperbox (only be used for contraction case 9)

**Returns****dim**

[list with two integer elements] The first element contains the overlap test case which two hyperboxes overlap with each other. The second element contains the corresponding dimension where two hyperboxes overlap with each other.

hbbrain.utils.adjust\_hyperbox.**hyperbox\_overlap\_test\_fmnn**(V, W, id\_extended\_box, id\_tested\_box)

Check the overlap of two input hyperboxes

**Parameters**

**V**

[array-like of shape (n\_hyperboxes, n\_features)] Lower bounds (minimum points) of all existing hyperboxes in the trained model.

**W**

[array-like of shape (n\_hyperboxes, n\_features)] Upper bounds (maximum points) of all existing hyperboxes in the trained model.

**id\_extended\_box**

[int] id\_extended\_boxex of the extended hyperbox which needs to test for overlap.

**id\_tested\_box**

[int] id\_extended\_boxex of the hyperbox to test for overlap with the extended hyperbox.

**Returns****dim**

[list with two integer elements] The first element contains the overlap test case which two hyperboxes overlap with each other. The second element contains the corresponding dimension where two hyperboxes overlap with each other.

```
hbbrain.utils.adjust_hyperbox.hyperbox_overlap_test_freq_cat_gfmm(E, F, id_extended_box,
                                                                    id_tested_box, X_cat,
                                                                    similarity_of_cat_vals,
                                                                    cat_overlap_resolved_hyperbox_id)
```

Test overlap in categorical features between two input hyperboxes.

**Parameters****E**

[array-like of shape (n\_hyperboxes, n\_cat\_features)] Lower bounds for categorical features of all existing hyperboxes.

**F**

[array-like of shape (n\_hyperboxes, n\_cat\_features)] Upper bounds for categorical features of all existing hyperboxes.

**id\_extended\_box**

[int] Index of the extended hyperbox which needs to test for overlap.

**id\_tested\_box**

[int] Index of the hyperbox to test for overlap with the extended hyperbox.

**X\_cat**

[array-like of shape (n\_samples, n\_cat\_features)] Categorical features of all training data.

**similarity\_of\_cat\_vals**

[array-like of shape (n\_cat\_features,)] An array stores all similarity values among all pairs of categorical values for each categorical feature index. Each element in this array is a dictionary with keys being a hashed value of two categorical values and values of this dictionary being a similarity value.

**cat\_overlap\_resolved\_hyperbox\_id**

[a list of int] Indices of hyperboxes overlapping with the extended hyperbox but the overlapping regions among them were resolved by changing values on only categorical features. When replacing an overlap area by other values, we are not allowed to create the overlapping regions with the hyperboxes having indices stored in this list.

**Returns****dim**

[a list of two element in the form of [dimension, replaced values]] If this list is empty, there

is no overlapping area among categorical features in two hyperboxes. Otherwise, this list shows the categorical dimension where the overlap occurs. If the first element in this list (dimension) gets the value of -1, it means that there is an overlapping region but we cannot find a suitable value to replace for any dimension to resolve the overlap in the categorical features. The second element in this list contains two new values for lower and upper bounds in the categorical dimension shown in *dimension*. If the second value of this list is None, then no change happens in the bounds in that categorical dimension.

`hbbrain.utils.adjust_hyperbox.is_overlap_cat_features_one_by_one(E1, F1, E2, F2)`

Check whether all categorical features of two input hyperboxes represented by lower bounds *E1*, *E2* and upper bounds *F1*, *F2* overlap with each other.

**Parameters**

**E1**

[array-like of shape (n\_cat\_features, )] Lower bound for categorical features of the first hyperbox.

**F1**

[array-like of shape (n\_cat\_features, )] Upper bound for categorical features of the first hyperbox.

**E2**

[array-like of shape (n\_cat\_features, )] Lower bound for categorical features of the second hyperbox.

**F2**

[array-like of shape (n\_cat\_features, )] Upper bound for categorical features of the second hyperbox.

**Returns**

**bool**

True if all categorical features in bounds overlap with each other. Otherwise, return False.

`hbbrain.utils.adjust_hyperbox.is_overlap_cat_features_one_vs_many(E1, F1, E, F, tested_box_ids=[])`

Check for overlap in categorical features between an input hyperbox and a list of existing hyperboxes.

**Parameters**

**E1**

[array-like of shape (n\_cat\_features, )] Lower bound for categorical features of the hyperbox which needs to check for overlap.

**F1**

[array-like of shape (n\_cat\_features, )] Upper bound for categorical features of the hyperbox which needs to check for overlap.

**E**

[array-like of shape (n\_hyperboxes, n\_cat\_features)] Lower bounds for categorical features of all existing hyperboxes.

**F**

[array-like of shape (n\_hyperboxes, n\_cat\_features)] Upper bounds for categorical features of all existing hyperboxes.

**tested\_box\_ids**

[a list of int, optional, default=[]] The indices of existing hyperboxes with which the input hyperbox needs to check overlap.

**Returns****bool**

Return True if the categorical features of the input hyperbox overlap with any existing hyperboxes that are checked for.

`hbbrain.utils.adjust_hyperbox.is_overlap_diff_labels_num_data_rfmnn(V, W, V_cmp, W_cmp, find_dim_min_overlap=True)`

Check whether there is any overlapping region between the hyperbox represented by minimum point *V\_cmp* and maximum point *W\_cmp* and any hyperbox in the existing list of hyperboxes belonging to other classes. The detailed information of this procedure can be found in [1].

**Parameters****V**

[array-like of shape (n\_hyperboxes, n\_features)] Lower bounds (minimum points) of hyperboxes representing other classes compared to *V\_cmp*.

**W**

[array-like of shape (n\_hyperboxes, n\_features)] Upper bounds (maximum points) of hyperboxes representing other classes compared to *W\_cmp*.

**V\_cmp**

[array-like of shape (n\_features,)] Minimum point of the compared hyperbox.

**W\_cmp**

[array-like of shape (n\_features,)] Maximum point of the compared hyperbox.

**find\_dim\_min\_overlap**

[boolean, optional, default=True] If True, then find the dimension causing the minimum overlap between the hyperbox [*V\_cmp*, *W\_cmp*] and any hyperboxes in the list of hyperboxes represented by [*V*, *W*]. Otherwise, only test whether there is any existing overlap zone.

**Returns**

**if find\_dim\_min\_overlap == False:**

return False - no overlap, True - overlap

**else:**

**return:**

- *is\_overlap*: False - no overlap, True - overlap
- *hyperbox\_ids\_overlap*: indices of hyperboxes overlap with [*V\_cmp*, *W\_cmp*] - numpy array
- *min\_overlap\_dimension*: dimension with minimum overlap value > 0 corresponding to hyperboxes with id located in *hyperbox\_id\_overlap*

**if is\_overlap == False:**

*hyperbox\_ids\_overlap* = *min\_overlap\_dimension* = None

## References

[1]

```
hbbrain.utils.adjust_hyperbox.is_overlap_one_many_diff_label_hyperboxes_mixed_data_general(V,  
                                                                                          W,  
                                                                                          D,  
                                                                                          N_samples,  
                                                                                          V_cmp,  
                                                                                          W_cmp,  
                                                                                          D_cmp,  
                                                                                          N_samples_cmp)
```

Check whether an input hyperbox overlaps with any hyperboxes representing different classes with the input hyperbox

### Parameters

**V**

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] Lower continuous bounds (minimum points) of hyperboxes representing other classes compared to V\_cmp.

**W**

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] Upper continuous bounds (maximum points) of hyperboxes representing other classes compared to W\_cmp.

**D**

[array-like of shape (n\_hyperboxes, n\_cat\_features)] A matrix stores a special structure for categorical features of all hyperboxes representing other classes compared to D\_cmp. Each element in *D* stores a set of symbolic values with their cardinalities for the j-th categorical dimension of a given hyperbox.

**N\_samples**

[array-like of shape (n\_hyperboxes,)] A vector save the number of samples contained in each hyperbox stored in the lists of V, W, D

**V\_cmp**

[array-like of shape (n\_continuous\_features,)] Minimum point of the compared hyperbox.

**W\_cmp**

[array-like of shape (n\_continuous\_features,)] Maximum point of the compared hyperbox.

**D\_cmp**

[array-like of shape (n\_cat\_features,)] Categorical bound of the compared hyperbox. It contains a set of symbolic values with their cardinalities for the j-th categorical dimension of the compared hyperbox.

**N\_samples\_cmp**

[int] A scalar storing the number of hyperboxes included in the hyperbox represented by [V\_cmp, W\_cmp, D\_cmp]

### Returns

**bool**

Show if the input hyperbox overlaps with any hyperbox in the list of hyperboxes representing the classes other than the input hyperbox.

```
hbbrain.utils.adjust_hyperbox.is_overlap_one_many_diff_label_hyperboxes_num_data_general(V,  
                                                                                          W,  
                                                                                          V_cmp,  
                                                                                          W_cmp)
```



Check whether an input hyperbox overlaps with any hyperboxes representing different classes with the input hyperbox

**Parameters****V**

[array-like of shape (n\_hyperboxes, n\_features)] Lower bounds (minimum points) of hyperboxes representing other classes compared to V\_cmp.

**W**

[array-like of shape (n\_hyperboxes, n\_features)] Upper bounds (maximum points) of hyperboxes representing other classes compared to W\_cmp.

**V\_cmp**

[array-like of shape (n\_features,)] Minimum point of the compared hyperbox.

**W\_cmp**

[array-like of shape (n\_features,)] Maximum point of the compared hyperbox.

**Returns****bool**

Show if the input hyperbox overlaps with any hyperbox in the list of hyperboxes representing the classes other than the input hyperbox.

`hbbrain.utils.adjust_hyperbox.is_overlap_one_many_hyperboxes_num_data_general(V, W, C, id_box)`

Check overlap between the hyperbox at the position *id\_box* and remaning hyperboxes in the current list.

---

**Note:** The current input list of hyperboxes contains all existing hyperboxes including the hyperboxes representing the same class as the hyperbox at the position *id\_box*. Therefore, to perform overlap testing, the list of hyperboxes representing the class labels other than the class label of the *id\_box*-th hyperbox should be first filtered. Finally, the overlap test is only conducted on this filtered list.

---

**Parameters****V**

[array-like of shape (n\_hyperboxes, n\_features)] Lower bounds (minimum points) of all existing hyperboxes in the trained model.

**W**

[array-like of shape (n\_hyperboxes, n\_features)] Upper bounds (minimum points) of all existing hyperboxes in the trained model.

**C**

[array-like of shape (n\_hyperboxes,)] Class labels of all existing hyperboxes.

**id\_box**

[int] id\_extended\_boxex of the hyperbox to be checked for overlap.

**Returns****bool**

Show if the input hyperbox overlaps with any hyperbox in the list of hyperboxes representing the classes other than its class label.

`hbbrain.utils.adjust_hyperbox.is_two_hyperboxes_overlap_num_data_free_range_general(Vi, Wi, Vk, Wk)`

Check if two hyperboxes  $Bi$  and  $Bk$  overlap with each other or not. This function uses a general formula concerning the determination of an hyperbox within the overlapping region. If this hyperbox exists for all dimensions, two hyperboxes  $Bi$  and  $Bk$  overlap, else no overlap occurs.

**Parameters**

**Vi**

[array-like of shape (n\_features,)] Minimum coordinate of the hyperbox  $Bi$ .

**Wi**

[array-like of shape (n\_features,)] Maximum coordinate of the hyperbox  $Bi$ .

**Vk**

[array-like of shape (n\_features,)] Minimum coordinate of the hyperbox  $Bk$ .

**Wk**

[array-like of shape (n\_features,)] Maximum coordinate of the hyperbox  $Bk$ .

**Returns**

**is\_overlap**

[boolean] Show if two hyperboxes  $Bi$  and  $Bk$  overlap or not.

`hbbrain.utils.adjust_hyperbox.is_two_hyperboxes_overlap_num_data_general(Vi, Wi, Vk, Wk)`

Check if two hyperboxes  $Bi$  and  $Bk$  overlap with each other or not. This function uses a general formula built from a shortest gap-based similarity measure. If this measure returns a value of 1, it means that these two hyperboxes overlap with each other. Otherwise, two hyperboxes  $Bi$  and  $Bk$  do not overlap. See the references [1] and [2] for more details.

**Parameters**

**Vi**

[array-like of shape (n\_features,)] Minimum coordinate of the hyperbox  $Bi$ .

**Wi**

[array-like of shape (n\_features,)] Maximum coordinate of the hyperbox  $Bi$ .

**Vk**

[array-like of shape (n\_features,)] Minimum coordinate of the hyperbox  $Bk$ .

**Wk**

[array-like of shape (n\_features,)] Maximum coordinate of the hyperbox  $Bk$ .

**Returns**

**is\_overlap**

[boolean] Show if two hyperboxes  $Bi$  and  $Bk$  overlap or not.

## References

For more details regarding the way of checking the overlap between two hyperboxes, please see the following articles:

[1], [2]

`hbbrain.utils.adjust_hyperbox.overlap_resolving_num_data(Vi, Wi, ci, Vk, Wk, ck, alpha=1e-05)`

Resolve overlap between two hyperboxes  $B_i$  and  $B_k$  with coordinates being numerical features. For more details regarding the way of contracting two overlapping hyperboxes, please see the article [1]:

### Parameters

**$V_i$**

[array-like of shape (n\_features,)] Minimum coordinate of the hyperbox  $B_i$ .

**$W_i$**

[array-like of shape (n\_features,)] Maximum coordinate of the hyperbox  $B_i$ .

**$c_i$**

[int] Class label of the hyperbox  $B_i$ .

**$V_k$**

[array-like of shape (n\_features,)] Minimum coordinate of the hyperbox  $B_k$ .

**$W_k$**

[array-like of shape (n\_features,)] Maximum coordinate of the hyperbox  $B_k$ .

**$c_k$**

[int] Class label of the hyperbox  $B_k$ .

**$\alpha$**

[float, optional, default=0.00001] A very small value is used to avoid the overlap between two hyperboxes after contraction.

### Returns

**$V_i$**

[array-like of shape (n\_features,)] Minimum coordinate of the hyperbox  $B_i$  after contraction.

**$W_i$**

[array-like of shape (n\_features,)] Maximum coordinate of the hyperbox  $B_i$  after contraction.

**$V_k$**

[array-like of shape (n\_features,)] Minimum coordinate of the hyperbox  $B_k$  after contraction.

**$W_k$**

[array-like of shape (n\_features,)] Maximum coordinate of the hyperbox  $B_k$  after contraction.

## References

[1]

`hbbrain.utils.adjust_hyperbox.overlap_resolving_num_data_free_range(Vi, Wi, ci, Vk, Wk, ck, alpha=1e-05)`

Resolve overlap between two hyperboxes  $B_i$  and  $B_k$  with coordinates being numerical features with unlimited ranges. For more details regarding the way of contracting two overlapping hyperboxes, please see [1].

### Parameters

**$V_i$**

[array-like of shape (n\_features,)] Minimum coordinate of the hyperbox  $B_i$ .

**Wi**[array-like of shape (n\_features,)] Maximum coordinate of the hyperbox  $B_i$ .**ci**[int] Class label of the hyperbox  $B_i$ .**Vk**[array-like of shape (n\_features,)] Minimum coordinate of the hyperbox  $B_k$ .**Wk**[array-like of shape (n\_features,)] Maximum coordinate of the hyperbox  $B_k$ .**ck**[int] Class label of the hyperbox  $B_k$ .**alpha**

[float] A very small value is used to avoid the overlap between two hyperboxes after contraction.

**Returns****Vi**[array-like of shape (n\_features,)] Minimum coordinate of the hyperbox  $B_i$  after contraction.**Wi**[array-like of shape (n\_features,)] Maximum coordinate of the hyperbox  $B_i$  after contraction.**Vk**[array-like of shape (n\_features,)] Minimum coordinate of the hyperbox  $B_k$  after contraction.**Wk**[array-like of shape (n\_features,)] Maximum coordinate of the hyperbox  $B_k$  after contraction.**References**

[1]

**2.1.3 utils.matrix\_transformation**

The `hbbrain.utils.matrix_transformation` submodule implements various functions for matrix transformation measures.

`hbbrain.utils.matrix_transformation.hashing(a, b)`

Transform a pair of positive integer numbers into a unique number and this value is have commutation ability.

**Parameters****a**

[positive int] The first value.

**b**

[positive int] The second value.

**Returns****c**

[postivie int] A unique transformed value from the two input values.

`hbbrain.utils.matrix_transformation.hashing_mat(A, B)`

Transform each pair of items in two matrices A and B to a unique number

**Parameters**

**A**

[array-like of shape (n\_samples, n\_features)] The first matrix.

**B**

[array-like of shape (n\_samples, n\_features)] The second matrix.

**Returns**

**C**

[array-like of shape (n\_samples, n\_features)] A matrix that each element is a combination of two corresponding elements in two input matrices at the same position.

`hbbrain.utils.matrix_transformation.split_matrix(A, asimil_type='max', is_sort=True)`

Split an input matrix A into a maxtrix with three columns:

- The first column contains the row indices of A
- The second column contains the column indices of A
- The third column contains the values corresponding to each row and column

**Parameters**

**A**

[ndarray of shape (n\_samples, n\_features)] Input matrix needs to be split.

**asimil\_type**

[str, optional, default='max'] Use the minimum or maximum values of  $a_{ij}$  or  $a_{ij}$  for the third column if the matrix A is assymetric. Get a value of 'max' or 'min'.

**is\_sort**

[boolean, optional, default=True] Sort the values of the third column in a descending order or not.

**Returns**

**X**

[ndarray of shape (n\_samples, n\_features)] The outcome of the input matrix A after transformation.

## 2.1.4 utils.drawing\_func

The `hbbrain.utils.drawing_func` submodule implements various functions to support for drawing the hyperboxes.

`hbbrain.utils.drawing_func.draw_box(drawing_canvas, lw_bound, up_bound, color, linewidth=1)`

Drawing rectangular (2 dimensional inputs) or cube (3 and more dimensional inputs) shapes

**Parameters**

**drawing\_canvas**

[`axes.SubplotBase`, or another subclass of `Axes` in the matplotlib library] Plotting object of matplotlib.

**lw\_bound**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix storing lower bounds of all hyperboxes that we want to show in the canvas.

**up\_bound**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix storing upper bounds of all hyperboxes that we want to show in the canvas.

**color**

[int, tuple, or array-like of shape (n\_hyperboxes,)] A constant value or a tuple showing the same color for all hyperboxes or a vector storing the colors corresponding to the hyperboxes represented by *lw\_bound* and *up\_bound*

**linewidth**

[float, default=1] The width of hyperbox lines

**Returns****handler**

[list of Line2D or Line3D] A list of Line2D or Line3D depending on the number of dimensions initialised in *drawing\_canvas* to show the plotted objects.

`hbbrain.utils.drawing_func.draw_box_parallel_coordinate(X, y, y_pred, plot_width=800,  
plot_height=480,  
file_path='par_coor.html')`

Draw input samples in the form of parallel coordinates.

**Parameters****X**

[array like of shape (n\_samples, n\_features)] A matrix of samples needs to display in the parallel coordinates.

**y**

[array like of shape (n\_samples, )] Class labels of samples stored in *X*.

**y\_pred**

[int] The samples with the same label as *y\_pred* will be highlighted.

**plot\_width**

[int, optional, default=800] Width of the window to show graphs.

**plot\_height**

[int, optional, default=480] Height of the window to show graphs.

**file\_path**

[str, optional, default="par\_cord.html"] Path including a file name to the location storing the parallel coordinates graph.

**Returns**

None.

`hbbrain.utils.drawing_func.draw_decision_boundary_2D(drawing_canvas, XX, YY, yhat)`

Draw decision boundary in a 2-D plane

**Parameters****drawing\_canvas**

[*axes.SubplotBase*, or another subclass of *Axes* in the matplotlib library] A plotting object of matplotlib.

**XX**

[array-like of shape (Ny, Nx)] A coordinate matrix of the values on the X-axis created via **numpy.meshgrid**. The values of X must be ordered monotonically.

**YY**

[array-like of shape (Ny, Nx)] A coordinate matrix of the values on the Y-axis created via **numpy.meshgrid**. The values of X must be ordered monotonically.

**yhat**

[array-like of shape (n\_points,)] Predicted class labels for all points in the grid generated by XX and YY.

**Returns****None.**

hbbrain.utils.drawing\_func.**generate\_grid\_decision\_boundary\_2D**(*min\_x=0, max\_x=1, min\_y=0, max\_y=1, step=0.01*)

Generate a grid of points on the 2-D plane to determine the class label of these points from which decision boundary can be deduced.

**Parameters****min\_x**

[float, optional, default = 0] Starting coordinate of the 2-D grid on the X-axis.

**max\_x**

[float, optional, default = 0] Ending coordinate of the 2-D grid on the X-axis.

**min\_y**

[float, optional, default = 0] Starting coordinate of the 2-D grid on the Y-axis.

**max\_y**

[float, optional, default = 0] Ending coordinate of the 2-D grid on the Y-axis.

**step**

[float, optional, default = 0.01] The distance between two next points.

**Returns****a grid of points and coordinate matrices from coordinate vectors****grid**

[array-like of shape (n\_points, 2)] A matrix contains all pairs of points of a 2-D grid.

**XX**

[array-like of shape (Ny, Nx)] A coordinate matrix generated from a coordinate vector on the X-axis defined by *min\_x*, *max\_x*, and *step*.  $N_y = (max\_y - min\_y)/step$  and  $N_x = (max\_x - min\_x)/step$ .

**YY**

[array-like of shape (Ny, Nx)] A coordinate matrix generated from a coordinate vector on the Y-axis defined by *min\_y*, *max\_y*, and *step*.  $N_y = (max\_y - min\_y)/step$  and  $N_x = (max\_x - min\_x)/step$ .

---

**Note:** The number of elements *n\_points* in the matrix *grid* is computed by  $\frac{max_x - min_x}{step} \cdot \frac{max_y - min_y}{step}$ .

---

`hbbrain.utils.drawing_func.get_cmap(n, name='brg')`

Get a colormap instance mapping each index in 0, 1, ..., n-1 to a distinct RGB color.

**Parameters**

**n**

[int or None, default: None] If name is not already a Colormap instance and n is not None, the colormap will be resampled to have n entries in the lookup table.

**name**

[matplotlib.colors.Colormap or str or None, default: 'brg'] If a Colormap instance, it will be returned. Otherwise, the name of a colormap known to Matplotlib, which will be resampled by n.

**Returns**

**Return a function that maps each index in 0, 1, ..., n-1 to a distinct RGB color.**

**Examples**

```
>>> from hbbrain.utils.drawing_func import get_cmap
>>> cmap = get_cmap(2)
>>> cmap(0)
(0.0, 0.0, 1.0, 1.0)...
```

## 2.1.5 utils.dist\_metrics

The `hbbrain.utils.dist_metrics` submodule implements various functions to compute distance-based metrics.

`hbbrain.utils.dist_metrics.manhattan_distance(X, Y)`

Compute Manhattan distance between two points X and Y

**Parameters**

**X**

[ndarray of shape (n\_features,) or (n\_samples, n\_features)] Vector or matrix contains the coordinates of the first point.

**Y**

[ndarray of shape (n\_features,) or (n\_samples, n\_features)] Vector or matrix contains the coordinates of the second point.

**Returns**

**d**

[float or ndarray of shape (n\_samples,)] A scalar value or a vector stores the resulting Manhattan distance values.

`hbbrain.utils.dist_metrics.manhattan_distance_with_missing_val(X1, X2, Y1, Y2)`

Compute Manhattan distance between the central points of X1, X2 and Y1, Y2.

---

**Note:**  $X1$ ,  $X2$ ,  $Y1$ ,  $Y2$  can contain missing values. In that case,  $X1j=1+EPSILON\_MISSING\_VAL > X2j=-EPSILON\_MISSING\_VAL$  and  $Y1j=1+EPSILON\_MISSING\_VAL > Y2j=-EPSILON\_MISSING\_VAL$ . The Manhattan distance is only computed for the dimensions without missing values.

---



**Parameters****X1**

[ndarray of shape (n\_features,) or (n\_samples, n\_features)] Vector or matrix contains the lower bounds of the first point.

**X2**

[ndarray of shape (n\_features,) or (n\_samples, n\_features)] Vector or matrix contains the upper bounds of the first point.

**Y1**

[ndarray of shape (n\_features,) or (n\_samples, n\_features)] Vector or matrix contains the lower bounds of the second point.

**Y2**

[ndarray of shape (n\_features,) or (n\_samples, n\_features)] Vector or matrix contains the upper bounds of the second point.

**Returns****result**

[ndarray of shape (n\_samples,)] A vector stores the resulting Manhattan distance values.

```
hbbrain.utils.dist_metrics.manhattan_distance_with_missing_val_free_range(X1, X2, Y1, Y2,
                                                                           MIN_RANGE,
                                                                           MAX_RANGE)
```

Compute Manhattan distance between the central points of X1, X2 and Y1, Y2. The coordinates are not limited by ranges.

---

**Note:** *X1*, *X2*, *Y1*, *Y2* can contain missing values. In that case,  $X1j=MAX\_RANGE > X2j=MIN\_RANGE$  and  $Y1j=MAX\_RANGE > Y2j=MIN\_RANGE$ . The Manhattan distance is only computed for the dimensions without missing values.

---

**Parameters****X1**

[ndarray of shape (n\_features,) or (n\_samples, n\_features)] Vector or matrix contains the lower bounds of the first point.

**X2**

[ndarray of shape (n\_features,) or (n\_samples, n\_features)] Vector or matrix contains the upper bounds of the first point.

**Y1**

[ndarray of shape (n\_features,) or (n\_samples, n\_features)] Vector or matrix contains the lower bounds of the second point.

**Y2**

[ndarray of shape (n\_features,) or (n\_samples, n\_features)] Vector or matrix contains the upper bounds of the second point.

**MIN\_RANGE**

[float] The minimum value of floating numbers for missing features.

**MAX\_RANGE**

[float] The maximum values of floating numbers for missing features.

**Returns**

**result**

[ndarray of shape (n\_samples,)] A vector stores the resulting Manhattan distance values.

`hbbrain.utils.dist_metrics.rfmn_distance(X, V, W)`

Compute the distance from the input pattern to the list of existing hyperboxes represented by minimum points *V* and maximum points *W*.

**Parameters****X**

[ndarray of shape (n\_features,) or (n\_hyperboxes, n\_features)] Vector or matrix contains the coordinates of the input pattern.

**V**

[ndarray of shape (n\_hyperboxes, n\_features)] Lower bounds of all existing hyperboxes.

**W**

[ndarray of shape (n\_hyperboxes, n\_features)] Upper bounds of all existing hyperboxes.

**Returns****dist**

[ndarray of shape (n\_hyperboxes,)] The distance values from the input pattern to all existing hyperboxes.

## 2.1.6 `utils.model_storage`

The `hbbrain.utils.model_storage` submodule implements various functions to store a trained model to the local file and load it from the file.

`hbbrain.utils.model_storage.load_model(filename)`

Load a stored model from a file

**Parameters****filename**

[str] The path to file storing the trained model.

**Returns****model**

[object] An model stored in the file.

`hbbrain.utils.model_storage.load_multi_models(filename)`

Deserialize a file containing many trained models

**Parameters****filename**

[str] The path to file storing the trained model.

**Yields****objects**

An iterator through many models stored in the file.

`hbbrain.utils.model_storage.store_model(model, filename)`

Store an trained model or a list of trained models to the file

**Parameters**

**model**

[object] A trained model or a list of the trained models needs to store.

**filename**

[str] The path to file storing the trained models.

**Returns**

None.

## 2.2 base

### 2.2.1 base.base\_estimator

Base class for all hyperbox-based estimators.

**class** hbbrain.base.base\_estimator.**BaseHyperboxClassifier**(*theta=0.5, is\_draw=False, V=None, W=None, C=None*)

Base class for all hyperbox-based estimators in hyperbox-brain.

---

**Note:** All estimators should specify all the parameters that can be set at the class level in their `__init__` as explicit keyword arguments (no `*args` or `**kwargs`). This class only initialises all common parameters for hyperbox-based estimators.

---

**Parameters****theta**

[float or ndarray of shape (n\_features,), optional, default = 0.5] A maximum hyperbox size parameter for each dimension.

**gamma**

[float or ndarray of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each dimension.

**is\_draw**

[boolean, optional, default = False] A parameter is used to indicate whether the process of hyperbox building can be dynamically displayed on a canvas or not. This functionality displays hyperboxes in the form of 2D or 3D. In the case that the number of dimensions is higher than 3, only the three features are shown.

**V**

[array-like of shape (n\_hyperboxes, n\_features), default = an empty ndarray] A matrix stores all minimal coordinates of all existing hyperboxes, in which each row is a minimal coordinate of a hyperbox.

**W**

[array-like of shape (n\_hyperboxes, n\_features), default = an empty ndarray] A matrix stores all maximal coordinates of all existing hyperboxes, in which each row is a maximal coordinate of a hyperbox.

**C**

[ndarray of shape (n\_hyperboxes,), default = an empty ndarray] An array contains all class labels for all existing hyperboxes.

**Attributes**

**n\_hyperboxes**

[int] Number of hyperboxes built during fit.

**Methods**

<code>delay([delay_constant])</code>	Delay a time period to display hyperboxes
<code>draw_hyperbox_and_boundary([window_name, ...])</code>	Draw the existing hyperboxes and their decision boundaries among classes
<code>fit(X, y)</code>	Fit the model according to the given training data.
<code>get_n_hyperboxes()</code>	Get number of hyperboxes in the trained hyperbox-based model
<code>get_params([deep])</code>	Get parameters for this estimator.
<code>initialise_canvas_graph([n_dims, ...])</code>	Initialise a canvas to draw hyperboxes
<code>score(X, y[, sample_weight])</code>	Return the mean accuracy on the given test data and labels.
<code>set_params(**params)</code>	Set the parameters of this estimator.
<code>show_sample_explanation(xl, xu, ...[, ...])</code>	Show explanation for predicted results of an input pattern under the form of parallel coordinates or hyperboxes in 2D or 3D planes.

**delay**(*delay\_constant*=0.01)

Delay a time period to display hyperboxes

**Parameters****delay\_constant**

[float] Delay time period to display hyperboxes on the canvas

**draw\_hyperbox\_and\_boundary**(*window\_name*='Hyperbox-based classifier and its decision boundaries',  
*min\_range*=0, *max\_range*=1)

Draw the existing hyperboxes and their decision boundaries among classes

---

**Note:** This function only works on 2-dimensional datasets

---

**Parameters****window\_name**

[str, optional, default="Hyperbox-based classifier and its decision boundaries"] Name of plotting window showing hyperboxes and their decision boundaries.

**min\_range**

[float, optional, default=0] Minimum value in each axis.

**max\_range**

[float, optional, default=1] Maximum value in each axis.

**Returns**

None.

**fit**(*X*, *y*)

Fit the model according to the given training data.

**Parameters**

**X**

[array-like of shape (n\_samples, n\_features)] Training vector, where *n\_samples* is the number of samples and *n\_features* is the number of features.

**y**

[array-like of shape (n\_samples,)] Target vector relative to X.

**Returns****self**

Fitted estimator.

**get\_n\_hyperboxes()**

Get number of hyperboxes in the trained hyperbox-based model

**Returns****int**

Number of hyperboxes in the trained hyperbox-based classifier.

**initialise\_canvas\_graph**(*n\_dims=2, figure\_name='A trained hyperbox-based learning model', min\_range=0, max\_range=1*)

Initialise a canvas to draw hyperboxes

**Parameters****n\_dims**

[int, optional, default=2] The number of dimensions of hyperboxes shown in the canvas (2D or 3D).

**figure\_name**

[str, optional, default='A trained hyperbox-based learning model'] Title name of the window containing hyperboxes.

**fig\_num**

[int, optional, default=1] Index of canvas.

**min\_range**

[float, optional, default=0] Minimum value in each axis.

**max\_range**

[float, optional, default=1] Maximum value in each axis.

**Returns****drawing\_canvas**

[*axes.SubplotBase*, or another subclass of *Axes* in the matplotlib library] Plotting object of matplotlib.

**show\_sample\_explanation**(*xl, xu, dict\_min\_point\_classes, dict\_max\_point\_classes, y\_pred, type\_plot='par\_cord', plot\_width=800, plot\_height=480, min\_range=0, max\_range=1, file\_path='par\_cord.html'*)

Show explanation for predicted results of an input pattern under the form of parallel coordinates or hyperboxes in 2D or 3D planes.

---

**Note:** This function only works on numerical features.

---

**Parameters**

**xl**

[array-like of shape (n\_features,)] Lower bound of numerical features of the input pattern which needs to show explanation.

**xu**

[array-like of shape (n\_features,)] Upper bound of numerical features of an input pattern which needs to show explanation.

**dict\_min\_point\_classes**

[dictionary] A dictionary stores all minimal points of hyperboxes having the maximum membership value for each class. The key is the class label and the value is the minimal points of all hyperboxes corresponding to each class.

**dict\_max\_point\_classes**

[dictionary] A dictionary stores all maximal points of hyperboxes having the maximum membership value for each class. The key is the class label and the value is the maximal points of all hyperboxes corresponding to each class.

**y\_pred**

[int] The predicted class of the input pattern.

**type\_plot**

[str, optional, default="par\_cord"] Type of graph to show explanation. If the value is *par\_cord*, a parallel coordinate is used. Otherwise, hyperboxes in 2D or 3D planes are shown.

**plot\_width**

[int, optional, default=800] Width of the window to show parallel coordinates.

**plot\_height**

[int, optional, default=480] Height of the window to show parallel coordinates.

**min\_range**

[float, optional, default=0] Minimum value in the axes to show hyperboxes in 2D or 3D planes.

**max\_range**

[float, optional, default=1] Maximum value in the axes to show hyperboxes in 2D or 3D planes.

**file\_path**

[str, optional, default="par\_cord.html"] Path including a file name to the location storing the parallel coordinates graph.

**Returns**

None.

## 2.2.2 base.base\_gfmm\_estimator

Base class and functions for all general fuzzy min-max neural network estimators.

```
class hbbrain.base.base_gfmm_estimator.BaseGFMMClassifier(theta=0.5, gamma=1, is_draw=False,  
                                                         V=None, W=None, C=None)
```

Base class for all hyperbox-based estimators in the hyperbox-brain.

---

**Note:** All estimators should specify all the parameters that can be set at the class level in their `__init__` as explicit keyword arguments (no `*args` or `**kwargs`). This class only initialises all common parameters for

hyperbox-based estimators.

---

### Parameters

#### **theta**

[float or ndarray of shape (n\_features,), optional, default = 0.5] A maximum hyperbox size parameter for each dimension.

#### **gamma**

[float or ndarray of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each dimension.

#### **is\_draw**

[boolean, optional, default = False] A parameter is used to indicate whether the process of hyperbox building can be dynamically displayed on a canvas or not. This functionality displays hyperboxes in the form of 2D or 3D. In the case that the number of dimensions is higher than 3, only the three features are shown.

#### **V**

[array-like of shape (n\_hyperboxes, n\_features), default = an empty ndarray] A matrix stores all minimal coordinates of all existing hyperboxes, in which each row is a minimal coordinate of a hyperbox.

#### **W**

[array-like of shape (n\_hyperboxes, n\_features), default = an empty ndarray] A matrix stores all maximal coordinates of all existing hyperboxes, in which each row is a maximal coordinate of a hyperbox.

#### **C**

[ndarray of shape (n\_hyperboxes,), default = an empty ndarray] An array contains all class labels for all existing hyperboxes.

### Attributes

#### **n\_hyperboxes**

[int] Number of hyperboxes built during fit.

## Methods

<code>delay([delay_constant])</code>	Delay a time period to display hyperboxes
<code>draw_hyperbox_and_boundary([window_name, ...])</code>	Draw the existing hyperboxes and their decision boundaries among classes
<code>fit(X, y)</code>	Fit the model according to the given training data.
<code>get_n_hyperboxes()</code>	Get number of hyperboxes in the trained hyperbox-based model
<code>get_params([deep])</code>	Get parameters for this estimator.
<code>initialise_canvas_graph([n_dims, ...])</code>	Initialise a canvas to draw hyperboxes
<code>predict(X)</code>	Predict class labels for samples in $X$ .
<code>predict_proba(X)</code>	Predict class probabilities of the input samples $X$ .
<code>predict_with_membership(X)</code>	Predict class membership values of the input samples $X$ .
<code>score(X, y[, sample_weight])</code>	Return the mean accuracy on the given test data and labels.
<code>set_params(**params)</code>	Set the parameters of this estimator.
<code>show_sample_explanation(xl, xu, ...[, ...])</code>	Show explanation for predicted results of an input pattern under the form of parallel coordinates or hyperboxes in 2D or 3D planes.

### `predict(X)`

Predict class labels for samples in  $X$ .

---

**Note:** In the case there are many winner hyperboxes representing different class labels but with the same membership value with respect to the input pattern  $X_i$ , an additional criterion based on the minimum Manhattan distance between the input pattern  $X_i$  and the central points of winner hyperboxes are used to find the final winner hyperbox that its class label is used for predicting the class label of the input pattern  $X_i$ .

---

#### Parameters

##### $X$

[array-like of shape (n\_samples, n\_features)] The data matrix for which we want to predict the targets.

#### Returns

##### $y_{\text{pred}}$

[ndarray of shape (n\_samples,)] Vector containing the predictions. In binary and multiclass problems, this is a vector containing  $n_{\text{samples}}$ .

### `predict_proba(X)`

Predict class probabilities of the input samples  $X$ .

The predicted class probability is the fraction of the membership value of the representative hyperbox of that class and the sum of all membership values of all representative hyperboxes of all classes.

#### Parameters

##### $X$

[array-like of shape (n\_samples, n\_features)] The input samples.

#### Returns



**proba**

[ndarray of shape (n\_samples, n\_classes)] The class probabilities of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

**predict\_with\_membership(X)**

Predict class membership values of the input samples X.

The predicted class membership value is the membership value of the representative hyperbox of that class.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The input samples.

**Returns****mem\_vals**

[ndarray of shape (n\_samples, n\_classes)] The class membership values of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

`hbbrain.base.base_gfmm_estimator.convert_format_missing_input_zero_one(Xl, Xu, y=None)`

Convert missing values in the features and labels under the form of NaN values to the form used in the algorithms

**Parameters****Xl**

[array-like of shape (n\_samples, n\_features)] A matrix containing lower bound values of features and samples, where *n\_samples* is the number of samples and *n\_features* is the number of features.

**Xu**

[array-like of shape (n\_samples, n\_features)] A matrix containing upper bound values of features and samples, where *n\_samples* is the number of samples and *n\_features* is the number of features.

**y**

[array-like of shape (n\_samples,)] Target vector relative to [Xl, Xu].

**Returns****Xl\_out**

[array-like of shape (n\_samples, n\_features)] The transformed matrix of the input matrix Xl.

**Xu\_out**

[array-like of shape (n\_samples, n\_features)] The transformed matrix of the input matrix Xu.

**y\_out**

[array-like of shape (n\_samples, n\_features)] The transformed vector of the input vector y.

`hbbrain.base.base_gfmm_estimator.is_contain_missing_value(X)`

Check whether an input vector X contains any missing values.

**Parameters****X**

[array-like of shape (n\_features,) or (n\_samples, n\_features)] A input vector for which we want to check the existence of missing values.

**Returns****bool**

The output value showing whether the input vector X contains missing values or not.

`hbbrain.base.base_gfmm_estimator.predict_with_manhattan(V, W, C, Xl, Xu, g=1)`

Predict class labels for samples in  $X$  represented in the form of intervals  $[Xl, Xu]$ . This is a common function to determine the right class labels for  $X$  wrt. a trained hyperbox-based classifier represented by  $[V, W, C]$ . It uses the winner-takes-all principle to predict class labels for each sample in  $X$  by assigning the class label of the sample to the class label of the hyperbox with the maximum membership value to that sample. It will use a Manhattan distance in the case of many hyperboxes with different classes having the same maximum membership value.

#### Parameters

**V**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all minimal points of all hyperboxes of a trained hyperbox-based model, in which each row is a minimal point of a hyperbox.

**W**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all maximal points of all hyperboxes of a trained hyperbox-based model, in which each row is a maximal point of a hyperbox.

**C**

[ndarray of shape (n\_hyperboxes,)] An array contains all class labels for all hyperboxes of a trained hyperbox-based model.

**Xl**

[array-like of shape (n\_samples, n\_features)] The data matrix contains lower bounds of input patterns for which we want to predict the targets.

**Xu**

[array-like of shape (n\_samples, n\_features)] The data matrix contains upper bounds of input patterns for which we want to predict the targets.

**g**

[float or array-like of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each dimension.

#### Returns

**y\_pred**

[ndarray of shape (n\_samples,)] A vector contains the predictions. In binary and multiclass problems, this is a vector containing  $n\_samples$ .

`hbbrain.base.base_gfmm_estimator.predict_with_probability(V, W, C, N_samples, Xl, Xu, g=1)`

Predict class labels for samples in  $X$  represented in the form of intervals  $[Xl, Xu]$ . This is a common function to determine the right class labels for  $X$  wrt. a trained hyperbox-based classifier represented by  $[V, W, C]$ . It uses the winner-takes-all principle to predict class labels for each sample in  $X$  by assigning the class label of the sample to the class label of the hyperbox with the maximum membership value to that sample. It will use a probability formula based on the number of samples included in each winner hyperbox in the case of many hyperboxes with different classes having the same maximum membership value.

#### Parameters

**V**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all minimal points of all hyperboxes of a trained hyperbox-based model, in which each row is a minimal point of a hyperbox.

**W**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all maximal points of all hyperboxes of a trained hyperbox-based model, in which each row is a maximal point of a hyperbox.

**C**

[ndarray of shape (n\_hyperboxes,)] An array contains all class labels for all hyperboxes of a trained hyperbox-based model.

**N\_samples**

[ndarray of shape (n\_hyperboxes,)] An array contains number of samples included in each hyperbox of a trained hyperbox-based model.

**Xl**

[array-like of shape (n\_samples, n\_features)] The data matrix contains lower bounds of input patterns for which we want to predict the targets.

**Xu**

[array-like of shape (n\_samples, n\_features)] The data matrix contains upper bounds of input patterns for which we want to predict the targets.

**g**

[float or array-like of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each dimension.

**Returns****y\_pred**

[ndarray of shape (n\_samples,)] A vector contains the predictions. In binary and multiclass problems, this is a vector containing *n\_samples*.

### 2.2.3 base.base\_fmnn\_estimator

Base classes for all fuzzy min-max neural network estimators and their improved versions.

**class** hbbrain.base.base\_fmnn\_estimator.**BaseFMNNClassifier**(*theta=0.5, gamma=1, is\_draw=False, V=None, W=None, C=None*)

Base class for all hyperbox-based estimators in hyperbox-brain.

---

**Note:** All estimators should specify all the parameters that can be set at the class level in their `__init__` as explicit keyword arguments (no `*args` or `**kwargs`). This class only initialises all common parameters for hyperbox-based estimators.

---

**Parameters****theta**

[float or ndarray of shape (n\_features,), optional, default = 0.5] A maximum hyperbox size parameter for each dimension.

**gamma**

[float or ndarray of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each dimension.

**is\_draw**

[boolean, optional, default = False] A parameter is used to indicate whether the process of hyperbox building can be dynamically displayed on a canvas or not. This functionality displays hyperboxes in the form of 2D or 3D. In the case that the number of dimensions is higher than 3, only the three features are shown.

**V**

[array-like of shape (n\_hyperboxes, n\_features), default = an empty ndarray] A matrix

stores all minimal coordinates of all existing hyperboxes, in which each row is a minimal coordinate of a hyperbox.

**W**

[array-like of shape (n\_hyperboxes, n\_features), default = an empty ndarray] A matrix stores all maximal coordinates of all existing hyperboxes, in which each row is a maximal coordinate of a hyperbox.

**C**

[ndarray of shape (n\_hyperboxes,), default = an empty ndarray] An array contains all class labels for all existing hyperboxes.

**Attributes****n\_hyperboxes**

[int] Number of hyperboxes built during fit.

**Methods**

<code>delay([delay_constant])</code>	Delay a time period to display hyperboxes
<code>draw_hyperbox_and_boundary([window_name, ...])</code>	Draw the existing hyperboxes and their decision boundaries among classes
<code>fit(X, y)</code>	Fit the model according to the given training data.
<code>get_n_hyperboxes()</code>	Get number of hyperboxes in the trained hyperbox-based model
<code>get_params([deep])</code>	Get parameters for this estimator.
<code><i>get_sample_explanation</i>(x)</code>	Get useful information for explaining the reason behind the predicted result for the input pattern
<code>initialise_canvas_graph([n_dims, ...])</code>	Initialise a canvas to draw hyperboxes
<code><i>predict</i>(X)</code>	Predict class labels for samples in X.
<code><i>predict_proba</i>(X)</code>	Predict class probabilities of the input samples X.
<code><i>predict_with_membership</i>(X)</code>	Predict class membership values of the input samples X.
<code>score(X, y[, sample_weight])</code>	Return the mean accuracy on the given test data and labels.
<code>set_params(**params)</code>	Set the parameters of this estimator.
<code>show_sample_explanation(xl, xu, ...[, ...])</code>	Show explanation for predicted results of an input pattern under the form of parallel coordinates or hyperboxes in 2D or 3D planes.
<code><i>simple_pruning</i>(X_val, y_val[, ...])</code>	Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox

**`get_sample_explanation(x)`**

Get useful information for explaining the reason behind the predicted result for the input pattern

**Parameters****x**

[ndarray of shape (n\_feature,)] The input pattern which needs to be explained.

**Returns****y\_pred**

[int] The predicted class of the input pattern

**dict\_mem\_val\_classes**

[dictionary] A dictionary stores all membership values for all classes. The key is class label and the value is the corresponding membership value.

**dict\_min\_point\_classes**

[dictionary] A dictionary stores all minimal points of hyperboxes having the maximum membership value for each class. The key is the class label and the value is the minimal points of all hyperboxes corresponding to each class

**dict\_max\_point\_classes**

[dictionary] A dictionary stores all maximal points of hyperboxes having the maximum membership value for each class. The key is the class label and the value is the maximal points of all hyperboxes corresponding to each class

**predict(X)**

Predict class labels for samples in  $X$ .

---

**Note:** In the case there are many winner hyperboxes representing different class labels but with the same membership value with respect to the input pattern  $X_i$ , an additional criterion based on the minimum Manhattan distance between the input pattern  $X_i$  and the central points of winner hyperboxes are used to find the final winner hyperbox that its class label is used for predicting the class label of the input pattern  $X_i$ .

---

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The data matrix for which we want to predict the targets.

**Returns****y\_pred**

[ndarray of shape (n\_samples,)] Vector containing the predictions. In binary and multiclass problems, this is a vector containing  $n\_samples$ .

**predict\_proba(X)**

Predict class probabilities of the input samples  $X$ .

The predicted class probability is the fraction of the membership value of the representative hyperbox of that class and the sum of all membership values of all representative hyperboxes of all classes.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The input samples.

**Returns****proba**

[ndarray of shape (n\_samples, n\_classes)] The class probabilities of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

**predict\_with\_membership(X)**

Predict class membership values of the input samples  $X$ .

The predicted class membership value is the membership value of the representative hyperbox of that class.

**Parameters**

**X**

[array-like of shape (n\_samples, n\_features)] The input samples.

**Returns****mem\_vals**

[ndarray of shape (n\_samples, n\_classes)] The class membership values of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

**simple\_pruning**(*X\_val*, *y\_val*, *acc\_threshold*=0.5, *keep\_empty\_boxes*=False)

Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox

**Parameters****X\_val**

[array-like of shape (n\_samples, n\_features)] The data matrix contains validation patterns.

**y\_val**

[ndarray of shape (n\_samples,)] A vector contains the true class label corresponding to each validation pattern.

**acc\_threshold**

[float, optional, default=0.5] The minimum accuracy for each hyperbox to be kept unchanged.

**keep\_empty\_boxes**

[boolean, optional, default=False] Whether to keep the hyperboxes which do not join the prediction process on the validation set. If True, keep them, else the decision for keeping or removing based on the classification accuracy on the validation dataset

**Returns****self**

A hyperbox-based model with the low-qualified hyperboxes pruned.

**hbbrain.base.base\_fmnn\_estimator.predict\_with\_manhattan\_fmnn**(*V*, *W*, *C*, *X*, *g*=1)

Predict class labels for samples in *X*.

---

**Note:** This is a common function to determine the right class labels for *X* with regard to a trained hyperbox-based classifier represented by [*V*, *W*, *C*]. It uses the winner-takes-all principle to predict class labels for each sample in *X* by assigning the class label of the sample to the class label of the hyperbox with the maximum membership value to that sample. It will use a Manhattan distance in the case of many hyperboxes with different classes having the same maximum membership value.

---

**Parameters****V**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all minimal points of all hyperboxes of a trained hyperbox-based model, in which each row is a minimal point of a hyperbox.

**W**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all maximal points of all hyperboxes of a trained hyperbox-based model, in which each row is a maximal point of a hyperbox.

**C**

[ndarray of shape (n\_hyperboxes,)] An array contains all class labels for all hyperboxes of a trained hyperbox-based model.

**X**

[array-like of shape (n\_samples, n\_features)] The data matrix contains input patterns for which we want to predict the targets.

**g**

[float or array-like of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each dimension.

**Returns****y\_pred**

[ndarray of shape (n\_samples,)] A vector contains the predictions. In binary and multiclass problems, this is a vector containing *n\_samples*.

## 2.2.4 base.base\_ensemble

Base functions and classes for ensemble models using hyperbox-based models.

**class** hbbrain.base.base\_ensemble.**BaseEnsemble**(*base\_estimator*, \*, *n\_estimators*=10, *estimator\_params*=())

Base class for all ensemble classes. Warning: This class should not be used directly. Use derived classes instead.

**Parameters****base\_estimator**

[object] The base estimator from which the ensemble is built.

**n\_estimators**

[int, default=10] The number of estimators in the ensemble.

**estimator\_params**

[list of str, default=tuple()] The list of attributes to use as parameters when instantiating a new base estimator. If none are given, default parameters are used.

**Attributes****base\_estimator\_**

[estimator] The base estimator from which the ensemble is grown.

**estimators\_**

[list of estimators] The collection of fitted base estimators.

**Methods**

<code>get_params([deep])</code>	Get parameters for this estimator.
<code>set_params(**params)</code>	Set the parameters of this estimator.

## 2.3 mixed-data learners

### 2.3.1 mixed\_data.eiol\_gfmm

General fuzzy min-max neural network trained by the extended improved incremental learning algorithm for mixed attribute data.

```
class hbbrain.mixed_data.eiol_gfmm.ExtendedImprovedOnlineGFM(theta=0.5, gamma=1, delta=0.5,  
                                                             alpha=0.5, V=None, W=None,  
                                                             D=None, C=None,  
                                                             N_samples=None)
```

Bases: *BaseGFMMClassifier*

Extended improved online learning algorithm for a general fuzzy min-max neural network with mixed-attribute data.

This algorithm can handle the datasets with both continuous and categorical features. It uses the change in the entropy values of categorical features of the samples contained in a hyperbox to determine if the current hyperbox can be expanded to include the categorical values of a new training instance. An extended architecture of the original general fuzzy min-max neural network and its new membership function are also introduced for mixed-attribute data.

See [1] for more detailed information regarding this extended improved online learning algorithm.

#### Parameters

##### **theta**

[float, optional, default=0.5] Maximum hyperbox size for continuous features.

##### **gamma**

[float or ndarray of shape (n\_continuous\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each continuous feature.

##### **delta**

[float, optional, default=0.5] A maximum entropy changing threshold for categorical values after expansion of the existing hyperbox to cover an input pattern.

##### **alpha**

[float, optional, default=0.5] A trade-off factor regulating the contribution level of continuous features part and categorical features part to the membership score.

##### **V**

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] A matrix stores all minimal points for continuous features of all existing hyperboxes, in which each row is a minimal point of a hyperbox.

##### **W**

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] A matrix stores all maximal points for continuous features of all existing hyperboxes, in which each row is a maximal point of a hyperbox.

##### **D**

[array-like of shape (n\_hyperboxes, n\_cat\_features)] A matrix stores a special structure for categorical features of all existing hyperboxes. Each element in *D* stores a set of symbolic values with their cardinalities for the j-th categorical dimension of a given hyperbox.



**C**

[array-like of shape (n\_hyperboxes,)] A vector stores all class labels corresponding to existing hyperboxes.

**N\_samples**

[array-like of shape (n\_hyperboxes,)] A vector stores the number of samples fully included in each existing hyperbox.

## References

[1]

## Examples

```
>>> from hbbrain.mixed_data.eiol_gfmm import ExtendedImprovedOnlineGFMM
>>> from hbbrain.datasets import load_japanese_credit
>>> X, y = load_japanese_credit()
>>> from sklearn.preprocessing import MinMaxScaler
>>> scaler = MinMaxScaler()
>>> numerical_features = [1, 2, 7, 10, 13, 14]
>>> categorical_features = [0, 3, 4, 5, 6, 8, 9, 11, 12]
>>> scaler.fit(X[:, numerical_features])
MinMaxScaler()
>>> X[:, numerical_features] = scaler.transform(X[:, numerical_features])
>>> clf = ExtendedImprovedOnlineGFMM(theta=0.1, delta=0.6)
>>> clf.fit(X, y, categorical_features)
>>> print("Number of hyperboxes = %d"%clf.get_n_hyperboxes())
Number of hyperboxes = 613
>>> clf.predict(X[[10, 100]])
array([1, 0])
```

## Attributes

**categorical\_features\_**

[int array of shape (n\_cat\_features,)] Indices of categorical features in the training data and hyperboxes.

**continuous\_features\_**

[int array of shape (n\_continuous\_features,)] Indices of continuous features in the training data and hyperboxes.

**is\_exist\_continuous\_missing\_value**

[boolean] Is there any missing values in continuous features in the training data.

**elapsed\_training\_time**

[float] Training time in seconds.

## Methods

<code>compute_increasing_entropy(...)</code>	Compute the increasing degree in the entropy for each categorical feature in both the current hyperbox and that hyperbox after extended.
<code>delay([delay_constant])</code>	Delay a time period to display hyperboxes
<code>draw_hyperbox_and_boundary([window_name, ...])</code>	Draw the existing hyperboxes and their decision boundaries among classes
<code>fit(X, y[, categorical_features, ...])</code>	Build a general fuzzy min-max neural network from the training set (X, y) using the extended improved online learning algorithm.
<code>get_n_hyperboxes()</code>	Get number of hyperboxes in the trained hyperbox-based model
<code>get_params([deep])</code>	Get parameters for this estimator.
<code>get_sample_explanation(x)</code>	Get useful information for explaining the reason behind the predicted result for the input pattern represented by upper and lower bounds for continuous features together with the categorical bounds for the categorical features.
<code>initialise_canvas_graph([n_dims, ...])</code>	Initialise a canvas to draw hyperboxes
<code>predict(X[, type_boundary_handling])</code>	Predict class labels for samples in X.
<code>predict_proba(X)</code>	Predict class probabilities of the input samples X including both continuous and categorical features.
<code>predict_with_membership(X)</code>	Predict class membership values of the input samples X including both categorical and continuous features.
<code>score(X, y[, sample_weight])</code>	Return the mean accuracy on the given test data and labels.
<code>set_params(**params)</code>	Set the parameters of this estimator.
<code>show_sample_explanation(xl, xu, ...[, ...])</code>	Show explanation for predicted results of an input pattern under the form of parallel coordinates or hyperboxes in 2D or 3D planes.
<code>simple_pruning(X_val, y_val[, ...])</code>	Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox.

### `compute_increasing_entropy(cat_extended_hyperbox, cat_cur_hyperbox)`

Compute the increasing degree in the entropy for each categorical feature in both the current hyperbox and that hyperbox after extended.

#### Parameters

##### `cat_extended_hyperbox`

[array-like of shape (n\_cat\_features,)] Categorical features in the current hyperbox after extended. Each dimension contains a dictionary with the key being categorical values and value being the number of samples in the hyperbox containing the given categorical value in that dimension.

##### `cat_cur_hyperbox`

[array-like of shape (n\_cat\_features,)] Categorical features in the current hyperbox. Each dimension contains a dictionary with the key being categorical values and value being the number of samples in the hyperbox containing the given categorical value in that dimension.

#### Returns

**increased\_entropy**

[array-like of shape (n\_cat\_features,)] The increased entropy value for each categorical dimension after extended.

**fit**(X, y, categorical\_features=None, N\_incl\_samples=None, type\_cat\_expansion=0)

Build a general fuzzy min-max neural network from the training set (X, y) using the extended improved online learning algorithm.

**Parameters****X**

[array-like of shape (n\_samples, n\_features) or (2\*n\_samples, n\_features)] The training input samples including both continuous and categorical features. If the number of rows in X is 2\*n\_samples, the first n\_samples rows contain lower bounds of input patterns and the rest n\_samples rows contain upper bounds.

**y**

[array-like of shape (n\_samples,)] The class labels.

**categorical\_features**

[a list of int, optional, default=None] Indices of categorical features in the training set. If None, there is no categorical feature.

**N\_incl\_samples**

[array-like of shape (n\_samples,), optional, default=None] A vector stores numbers of samples fully contained in the input patterns in the case that input patterns form hyperboxes.

**type\_cat\_expansion**

[int, optional, default=0] Type of the expansion condition for categorical features. If *type\_cat\_expansion* gets the value of 0, then the categorical feature expansion condition regarding the maximum entropy changing threshold will be applied for every categorical dimension. Otherwise, this expansion condition will be applied for the average entropy changing values of all categorical features.

**Returns****self**

[object] Fitted estimator.

**get\_n\_hyperboxes()**

Get number of hyperboxes in the trained hyperbox-based model

**Returns****int**

Number of hyperboxes in the trained hyperbox-based classifier.

**get\_sample\_explanation(x)**

Get useful information for explaining the reason behind the predicted result for the input pattern represented by upper and lower bounds for continuous features together with the categorical bounds for the categorical features.

**Parameters****x**

[ndarray of shape (n\_feature,)] The input pattern which needs to be explained includes both continuous features and categorical features.

**Returns****y\_pred**

[int] The predicted class of the input pattern

**dict\_mem\_val\_classes**

[dictionary] A dictionary stores all membership values for all classes. The key is class label and the value is the corresponding membership value.

**dict\_min\_point\_classes**

[dictionary] A dictionary stores all minimal points of hyperboxes having the maximum membership value for each class. The key is the class label and the value is the minimal points of the hyperbox corresponding to that class.

**dict\_max\_point\_classes**

[dictionary] A dictionary stores all maximal points of hyperboxes having the maximum membership value for each class. The key is the class label and the value is the maximal points of the hyperbox corresponding to that class.

**dict\_cat\_bound\_classes: dictionary**

A dictionary stores all categorical bounds of categorical features for the hyperboxes having the maximum membership value for each class. The key is the class label and the value is the categorical bound of categorical features for the hyperboxes corresponding to each class.

**predict**(*X*, *type\_boundary\_handling=1*)

Predict class labels for samples in *X*.

---

**Note:** In the case there are many winner hyperboxes representing different class labels but with the same membership value with respect to the input pattern  $X_i$ , an additional criterion based on the probability generated by number of samples included in winner hyperboxes and membership values or the Manhattan distance between the central point of winner hyperboxes and the input sample is used to find the final winner hyperbox that its class label is used for predicting the class label of the input pattern  $X_i$ .

---

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The data matrix for which we want to predict the targets.

**type\_boundary\_handling**

[int, optional, default=PROBABILITY\_MEASURE (aka 1)] The way of handling many winner hyperboxes, i.e., PROBABILITY\_MEASURE or MANHATTAN\_DIS

**Returns****y\_pred**

[ndarray of shape (n\_samples,)] Vector containing the predictions. In binary and multiclass problems, this is a vector containing *n\_samples*.

**predict\_proba**(*X*)

Predict class probabilities of the input samples *X* including both continuous and categorical features.

The predicted class probability is the fraction of the membership value of the representative hyperbox of that class and the sum of all membership values of all representative hyperboxes of all classes.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The input samples.

**Returns**

**proba**

[ndarray of shape (n\_samples, n\_classes)] The class probabilities of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

**predict\_with\_membership(X)**

Predict class membership values of the input samples X including both categorical and continuous features.

The predicted class membership value is the membership value of the representative hyperbox of that class.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The input samples.

**Returns****mem\_vals**

[ndarray of shape (n\_samples, n\_classes)] The class membership values of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

**simple\_pruning(X\_val, y\_val, acc\_threshold=0.5, keep\_empty\_boxes=False, type\_boundary\_handling=1)**

Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox.

**Parameters****X\_val**

[array-like of shape (n\_samples, n\_features)] The data matrix contains both continuous and categorical features of validation patterns.

**y\_val**

[ndarray of shape (n\_samples,)] A vector contains the true class label corresponding to each validation pattern.

**acc\_threshold**

[float, optional, default=0.5] The minimum accuracy for each hyperbox to be kept unchanged.

**keep\_empty\_boxes**

[boolean, optional, default=False] Whether to keep the hyperboxes which do not join the prediction process on the validation set. If True, keep them, otherwise the decision for keeping or removing based on the classification accuracy on the validation dataset.

**type\_boundary\_handling**

[int, optional, default=PROBABILITY\_MEASURE (aka 1)] The way of handling samples located on the boundary.

**Returns****self**

A hyperbox-based model with the low-qualified hyperboxes pruned.

**hbbrain.mixed\_data.eiol\_gfmm.impute\_missing\_categorical\_features(X\_cat)**

Impute missing values in categorical features by a default value.

**Parameters****X\_cat**

[array-like of shape (n\_samples, n\_cat\_features)] Input matrix contains categorical features only.

**Returns**

**X\_cat**

[array-like of shape (n\_samples, n\_cat\_features)] The resulting matrix contains categorical features of which missing categorical values have been imputed by a default value.

`hbbrain.mixed_data.eiol_gfmm.predict_with_manhattan_mixed_data(V, W, D, C, Xl, Xu, X_cat, g=1, alpha=0.5)`

Predict class labels for samples in  $X$  represented in the form of intervals  $[Xl, Xu]$  for continuous features and  $X\_cat$  for categorical features. This is a common function to determine the right class labels for  $X$  wrt. a trained hyperbox-based classifier represented by  $[V, W, D, C]$ . It uses the winner-takes-all principle to predict class labels for each sample in  $X$  by assigning the class label of the sample to the class label of the hyperbox with the maximum membership value to that sample. It will use a Manhattan distance for continuous features in the case of many hyperboxes with different classes having the same maximum membership value.

**Parameters****V**

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] A matrix stores all minimal points for all continuous features of all hyperboxes of a trained hyperbox-based model, in which each row is a minimal point of a hyperbox.

**W**

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] A matrix stores all maximal points for all continuous features of all hyperboxes of a trained hyperbox-based model, in which each row is a maximal point of a hyperbox.

**D**

[array-like of shape (n\_hyperboxes, n\_cat\_features)] A matrix stores all categorical bounds for categorical features of all hyperboxes of a trained hyperbox-based model, in which each row is a categorical bound of a hyperbox.

**C**

[ndarray of shape (n\_hyperboxes,)] An array contains all class labels for all hyperboxes of a trained hyperbox-based model.

**Xl**

[array-like of shape (n\_samples, n\_continuous\_features)] The data matrix contains lower bounds for continuous features of input patterns for which we want to predict the targets.

**Xu**

[array-like of shape (n\_samples, n\_continuous\_features)] The data matrix contains upper bounds for continuous features of input patterns for which we want to predict the targets.

**X\_cat**

[array-like of shape (n\_samples, n\_cat\_features)] The data matrix contains categorical bounds for categorical features of input patterns for which we want to predict the targets.

**g**

[float or array-like of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each continuous dimension.

**alpha**

[float, optional, default=0.5] The trade-off weighting factor between the impacts of categorical features and numerical features on the outputs of membership values.

**Returns****y\_pred**

[ndarray of shape (n\_samples,)] A vector contains the predictions. In binary and multiclass problems, this is a vector containing  $n\_samples$ .

---

```
hbbrain.mixed_data.eiol_gfmm.predict_with_probability_mixed_data(V, W, D, C, N_samples, Xl, Xu,
                                                                X_cat, g=1, alpha=0.5)
```

Predict class labels for samples in  $X$  represented in the form of intervals  $[Xl, Xu]$ . This is a common function to determine the right class labels for  $X$  wrt. a trained hyperbox-based classifier represented by  $[V, W, C]$ . It uses the winner-takes-all principle to predict class labels for each sample in  $X$  by assigning the class label of the sample to the class label of the hyperbox with the maximum membership value to that sample. It will use a probability formula based on the number of samples included in each winner hyperbox in the case of many hyperboxes with different classes having the same maximum membership value.

#### Parameters

**V**

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] A matrix stores all minimal points of all hyperboxes of a trained hyperbox based model, each row is a minimal point for continuous features of a hyperbox.

**W**

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] A matrix stores all maximal points of all hyperboxes of a trained hyperbox-based model, each row is a maximal point for continuous features of a hyperbox.

**D**

[array-like of shape (n\_hyperboxes, n\_cat\_features)] A matrix stores all maximal points of all hyperboxes of a trained hyperbox based model, each row is a categorical bound for a hyperbox.

**C**

[ndarray of shape (n\_hyperboxes,)] An array contains all class labels for all hyperboxes of a trained hyperbox-based model.

**N\_samples**

[ndarray of shape (n\_hyperboxes,)] An array contains number of samples included in each hyperbox of a trained hyperbox-based model.

**Xl**

[array-like of shape (n\_samples, n\_continuous\_features)] The data matrix contains lower bounds of input patterns for which we want to predict the targets.

**Xu**

[array-like of shape (n\_samples, n\_continuous\_features)] The data matrix contains upper bounds of input patterns for which we want to predict the targets.

**X\_cat**

[array-like of shape (n\_samples, n\_cat\_features)] The data matrix contains categorical bounds of input categorical patterns for which we want to predict the targets.

**g**

[float or array-like of shape (n\_continuous\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each continuous dimension.

**alpha**

[float, optional, default=0.5] The trade-off weighting factor between the impacts of categorical features and numerical features on the outputs of membership values.

#### Returns

**y\_pred**

[ndarray of shape (n\_samples,)] A vector contains the predictions. In binary and multiclass problems, this is a vector containing  $n\_samples$ .

### 2.3.2 mixed\_data.freq\_cat\_onln\_gfmm

General fuzzy min-max neural network trained by the batch incremental learning algorithm, in which categorical features are encoded using the ordinal encoding method and the similarity among categorical values are computed using their frequency of occurrence with respect to all class labels in a training set.

```
class hbbrain.mixed_data.freq_cat_onln_gfmm.FreqCatOnlineGFM(theta=0.5, theta_min=1, eta=0.5,  
                                                             gamma=1, alpha=0.9, V=None,  
                                                             W=None, E=None, F=None,  
                                                             C=None)
```

Bases: *BaseHyperboxClassifier*

Batch Incremental learning algorithm with mixed-attribute data for a general fuzzy min-max neural network, in which categorical features are encoded using the ordinal encoding method and the similarity degrees among categorical values are computed using their frequency of occurrence with respect to all class labels in a training set.

This algorithm uses a distance measure between any two values of a categorical variable based on the occurrence probability of such categorical values with respect to the values of the class variable. This distance is then normalised and used to compute the membership values for categorical features in conjunction with membership values of continuous features to generate the final membership values for mixed-attribute data.

See [1] for more detailed information regarding this batch incremental learning algorithm.

#### Parameters

##### **theta**

[float, optional, default=0.5] Maximum hyperbox size for continuous features.

##### **theta\_min**

[float, optional, default=1] Minimum value of the maximum hyperbox size for continuous features so that the training loop is still performed. If the value of *theta\_min* is larger than the value of *theta*, it will be automatically assigned a value equal to *theta*.

##### **gamma**

[float or ndarray of shape (n\_continuous\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each continuous feature.

##### **eta**

[float, optional, default=0.5] Maximum hyperbox size for the categorical features.

##### **alpha**

[float, optional, default=0.9] Multiplier factor to reduce the value of maximum hyperbox size after each training loop.

##### **V**

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] A matrix stores all minimal points for continuous features of all existing hyperboxes, in which each row is a minimal point of a hyperbox.

##### **W**

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] A matrix stores all maximal points for continuous features of all existing hyperboxes, in which each row is a maximal point of a hyperbox.

##### **E**

[array-like of shape (n\_hyperboxes, n\_cat\_features)] A matrix stores all lower bounds for categorical features of all existing hyperboxes, in which each row is a lower bound of a hyperbox.



**F**

[array-like of shape (n\_hyperboxes, n\_cat\_features)] A matrix stores all upper bounds for categorical features of all existing hyperboxes, in which each row is an upper bound of a hyperbox.

**C**

[array-like of shape (n\_hyperboxes,)] A vector stores all class labels corresponding to existing hyperboxes.

## References

[1]

## Examples

```
>>> from hbbrain.mixed_data.freq_cat_onln_gfmm import FreqCatOnlineGFMM
>>> from hbbrain.datasets import load_japanese_credit
>>> X, y = load_japanese_credit()
>>> from sklearn.preprocessing import MinMaxScaler
>>> scaler = MinMaxScaler()
>>> numerical_features = [1, 2, 7, 10, 13, 14]
>>> categorical_features = [0, 3, 4, 5, 6, 8, 9, 11, 12]
>>> scaler.fit(X[:, numerical_features])
MinMaxScaler()
>>> X[:, numerical_features] = scaler.transform(X[:, numerical_features])
>>> clf = FreqCatOnlineGFMM(theta=0.1, eta=0.6)
>>> clf.fit(X, y, categorical_features)
>>> print("Number of hyperboxes = %d"%clf.get_n_hyperboxes())
Number of hyperboxes = 416
>>> clf.predict(X[[10, 100]])
array([1, 0])
```

## Attributes

### **similarity\_of\_cat\_vals**

[array-like of shape (n\_cat\_features,)] An array stores all similarity values among all pairs of categorical values for each categorical feature index. Each element in this array is a dictionary with keys being a hashed value of two categorical values and values of this dictionary being a similarity value.

### **categorical\_features\_**

[int array of shape (n\_cat\_features,)] Indices of categorical features in the training data and hyperboxes.

### **continuous\_features\_**

[int array of shape (n\_continuous\_features,)] Indices of continuous features in the training data and hyperboxes.

### **encoder\_**

[sklearn.preprocessing.OrdinalEncoder] An ordinal encoder was used to encode categorical features.

### **is\_exist\_continuous\_missing\_value**

[boolean] Is there any missing values in continuous features in the training data.

**elapsed\_training\_time**  
[float] Training time in seconds.

**n\_passes**  
[int] Number of training loops.

## Methods

<code>delay([delay_constant])</code>	Delay a time period to display hyperboxes
<code>draw_hyperbox_and_boundary([window_name, ...])</code>	Draw the existing hyperboxes and their decision boundaries among classes
<code>fit(X, y[, categorical_features])</code>	Build a general fuzzy min-max neural network from the training set (X, y) using the original incremental learning algorithm, in which categorical features are encoded using the ordinal encoding method and the similarity among categorical values are computed using their frequency of occurrence with respect to all class labels in a training set.
<code>get_n_hyperboxes()</code>	Get number of hyperboxes in the trained hyperbox-based model
<code>get_params([deep])</code>	Get parameters for this estimator.
<code>get_sample_explanation(x)</code>	Get useful information for explaining the reason behind the predicted result for the input pattern represented by upper and lower bounds for continuous features together with the lower and upper bounds for the categorical features.
<code>initialise_canvas_graph([n_dims, ...])</code>	Initialise a canvas to draw hyperboxes
<code>is_satisfied_cat_expansion_conds(Ej, Fj, x_cat)</code>	Check whether the expansion condition for categorical features $x_{cat}$ of an input pattern can be covered by categorical bounds of the hyperbox $B_j$ with the categorical features stored in the lower bound $E_j$ and the upper bound $F_j$ .
<code>predict(X)</code>	Predict class labels for samples in X.
<code>predict_proba(X)</code>	Predict class probabilities of the input samples X including both continuous and categorical features.
<code>predict_with_membership(X)</code>	Predict class membership values of the input samples X including both categorical and continuous features.
<code>score(X, y[, sample_weight])</code>	Return the mean accuracy on the given test data and labels.
<code>set_params(**params)</code>	Set the parameters of this estimator.
<code>show_sample_explanation(xl, xu, ...[, ...])</code>	Show explanation for predicted results of an input pattern under the form of parallel coordinates or hyperboxes in 2D or 3D planes.
<code>simple_pruning(X_val, y_val[, ...])</code>	Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox.

**fit(X, y, categorical\_features=None)**

Build a general fuzzy min-max neural network from the training set (X, y) using the original incremental learning algorithm, in which categorical features are encoded using the ordinal encoding method and the similarity among categorical values are computed using their frequency of occurrence with respect to all class labels in a training set.

## Parameters

**X**

[array-like of shape (n\_samples, n\_features) or (2\*n\_samples, n\_features)] The training input samples including both continuous and categorical features. If the number of rows in X is 2\*n\_samples, the first n\_samples rows contain lower bounds of input patterns and the rest n\_samples rows contain upper bounds.

**y**

[array-like of shape (n\_samples,)] The class labels.

**categorical\_features**

[a list of int, optional, default=None] Indices of categorical features in the training set. If None, there is no categorical feature.

**Returns****self**

[object] Fitted estimator.

**get\_n\_hyperboxes()**

Get number of hyperboxes in the trained hyperbox-based model

**Returns****int**

Number of hyperboxes in the trained hyperbox-based classifier.

**get\_sample\_explanation(x)**

Get useful information for explaining the reason behind the predicted result for the input pattern represented by upper and lower bounds for continuous features together with the lower and upper bounds for the categorical features.

**Parameters****x**

[ndarray of shape (n\_feature,)] The input pattern which needs to be explained includes both continuous features and categorical features.

**Returns****y\_pred**

[int] The predicted class of the input pattern

**dict\_mem\_val\_classes**

[dictionary] A dictionary stores all membership values for all classes. The key is class label and the value is the corresponding membership value.

**dict\_min\_point\_classes**

[dictionary] A dictionary stores all minimal points of hyperboxes having the maximum membership value for each class. The key is the class label and the value is the minimal points of the hyperbox corresponding to that class.

**dict\_max\_point\_classes**

[dictionary] A dictionary stores all maximal points of hyperboxes having the maximum membership value for each class. The key is the class label and the value is the maximal points of the hyperbox corresponding to that class.

**dict\_min\_point\_cat\_classes: dictionary**

A dictionary stores all lower bounds of categorical features for the hyperboxes having the maximum membership value for each class. The key is the class label and the value is the lower bound of categorical features for the hyperboxes corresponding to each class.

**dict\_max\_point\_cat\_classes: dictionary**

A dictionary stores all upper bounds of categorical features for the hyperboxes having the maximum membership value for each class. The key is the class label and the value is the upper bound of categorical features for the hyperboxes corresponding to each class.

**is\_satisfied\_cat\_expansion\_conds(*Ej*, *Fj*, *x\_cat*)**

Check whether the expansion condition for categorical features *x\_cat* of an input pattern can be covered by categorical bounds of the hyperbox *Bj* with the categorical features stored in the lower bound *Ej* and the upper bound *Fj*.

**Parameters*****Ej***

[array-like of shape (n\_cat\_features,)] Lower bound of categorical features in the hyperbox *Bj* which can be extended to cover the input pattern.

***Fj***

[array-like of shape (n\_cat\_features,)] Upper bound of categorical features in the hyperbox *Bj* which can be extended to cover the input pattern.

***x\_cat***

[array-like of shape (n\_cat\_features,)] Categorical features of an input pattern.

**Returns****bool**

If True, the categorical features in *Dj* are satisfied with the expansion conditions for the categorical feature so that it can be expanded to cover the input pattern. Otherwise, the conditions for the categorical features are not met.

**predict(*X*)**

Predict class labels for samples in *X*.

---

**Note:** In the case there are many winner hyperboxes representing different class labels but with the same membership value with respect to the input pattern  $X_i$ , an additional criterion based on the minimum Manhattan distance between continuous features of  $X_i$  and the central points of continuous features of winner hyperboxes are used to find the final winner hyperbox that its class label is used for predicting the class label of the input pattern  $X_i$ . If there are only categorical features but many winner hyperboxes belonging to different classes, a random selection will be used to choose the final class label.

---

**Parameters*****X***

[array-like of shape (n\_samples, n\_features)] The data matrix for which we want to predict the targets.

**Returns*****y\_pred***

[ndarray of shape (n\_samples,)] Vector containing the predictions. In binary and multiclass problems, this is a vector containing *n\_samples*.

**predict\_proba(*X*)**

Predict class probabilities of the input samples *X* including both continuous and categorical features.

The predicted class probability is the fraction of the membership value of the representative hyperbox of that class and the sum of all membership values of all representative hyperboxes of all classes.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The input samples.

**Returns****proba**

[ndarray of shape (n\_samples, n\_classes)] The class probabilities of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

**predict\_with\_membership(X)**

Predict class membership values of the input samples X including both categorical and continuous features.

The predicted class membership value is the membership value of the representative hyperbox of that class.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The input samples.

**Returns****mem\_vals**

[ndarray of shape (n\_samples, n\_classes)] The class membership values of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

**simple\_pruning(X\_val, y\_val, acc\_threshold=0.5, keep\_empty\_boxes=False)**

Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox.

**Parameters****X\_val**

[array-like of shape (n\_samples, n\_features)] The data matrix contains both continuous and categorical features of validation patterns.

**y\_val**

[ndarray of shape (n\_samples,)] A vector contains the true class label corresponding to each validation pattern.

**acc\_threshold**

[float, optional, default=0.5] The minimum accuracy for each hyperbox to be kept unchanged.

**keep\_empty\_boxes**

[boolean, optional, default=False] Whether to keep the hyperboxes which do not join the prediction process on the validation set. If True, keep them, otherwise the decision for keeping or removing based on the classification accuracy on the validation dataset.

**Returns****self**

A hyperbox-based model with the low-qualified hyperboxes pruned.

`hbbrain.mixed_data.freq_cat_onln_gfmm.compute_similarity_among_categorical_values(X_cat, y)`

Compute the similarity among pairs of categorical values for each categorical feature.

**Parameters****X\_cat**

[array-like of shape (n\_samples, n\_cat\_features)] Input patterns contain only categorical features.

**y**  
[array-like of shape (n\_samples, )] The class label corresponds to each input pattern.

#### Returns

**similarity\_of\_cat\_vals**  
[array-like of shape (n\_cat\_features,)] An array stores all similarity values among all pairs of categorical values for each categorical feature index. Each element in this array is a dictionary with keys being a hashed value of two categorical values and values of this dictionary being a similarity value.

`hbbrain.mixed_data.freq_cat_onln_gfmm.ordinal_encode_categorical_features(X, categorical_features, encoder=None)`

Encode categorical features as an integer array.

#### Parameters

**X**  
[array-like of shape (n\_samples, n\_features)] An input data matrix includes both continuous and categorical features.

**categorical\_features**  
[a list of integer] Indices of categorical features in X.

**encoder**  
[sklearn.preprocessing.OrdinalEncoder, optional, default=None] An existing ordinal encoder is used to encode categorical features.

#### Returns

**X**  
[array-like of shape (n\_samples, n\_features)] An input data matrix with the encoded categorical features.

**encoder**  
[sklearn.preprocessing.OrdinalEncoder] An ordinal encoder was used to encode categorical features.

`hbbrain.mixed_data.freq_cat_onln_gfmm.predict_freq_cat_feature_manhatttan(V, W, E, F, C, Xl, Xu, X_cat, similarity_of_cat_vals, g=1)`

Predict class labels for samples in the form of hyperboxes with continuous features represented by low bounds  $Xl$  and upper bounds  $Xu$  and categorical features stored in  $X_{cat}$ . The predicted results will be computed from existing hyperboxes with continuous features matrices for lower bounds  $V$  and upper bounds  $W$  and categorical features matrices for lower bounds  $E$  and upper bounds  $F$ .

---

**Note:** In the case there are many winner hyperboxes representing different class labels but with the same membership value with respect to the input pattern  $X_i$  in the form of an hyperbox represented by a lower bound  $Xl_i$  and an upper bound  $Xu_i$  for continous features and a matrix  $Xcat_i$  for categorical features, an additional criterion based on the minimum Manhattan distance between the central point of continous features in the input hyperbox  $X_i = [Xl_i, Xu_i]$  and the central points of continous features in winner hyperboxes are used to find the final winner hyperbox that its class label is used for predicting the class label of the input hyperbox  $X_i$ .

---

**Warning:** Another important point to pay attention is that the categorical features storing in  $X_{cat}$  need to be encoded by using the function `ordinal_encode_categorical_features()` before pushing the values to this method.

### Parameters

**V**

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] A matrix stores all minimal points for all continuous features of all hyperboxes of a trained hyperbox-based model, in which each row is a minimal point of a hyperbox.

**W**

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] A matrix stores all maximal points for all continuous features of all hyperboxes of a trained hyperbox-based model, in which each row is a maximal point of a hyperbox.

**E**

[array-like of shape (n\_hyperboxes, n\_cat\_features)] A matrix stores all lower bounds for all categorical features of all hyperboxes of a trained hyperbox-based model, in which each row is a lower bound for categorical features of a hyperbox.

**F**

[array-like of shape (n\_hyperboxes, n\_cat\_features)] A matrix stores all upper bounds for all categorical features of all hyperboxes of a trained hyperbox-based model, in which each row is an upper bound for categorical features of a hyperbox.

**C**

[array-like of shape (n\_hyperboxes,)] An array contains all class labels for all hyperboxes of a trained hyperbox-based model.

**Xl**

[array-like of shape (n\_samples, n\_continuous\_features)] The data matrix contains lower bounds for continuous features of input patterns for which we want to predict the targets.

**Xu**

[array-like of shape (n\_samples, n\_continuous\_features)] The data matrix contains upper bounds for continuous features of input patterns for which we want to predict the targets.

**X\_cat**

[array-like of shape (n\_samples, n\_cat\_features)] The data matrix contains categorical bounds for categorical features of input patterns for which we want to predict the targets.

**similarity\_of\_cat\_vals**

[array-like of shape (n\_cat\_features,)] An array stores all similarity values among all pairs of categorical values for each categorical feature index. Each element in this array is a dictionary with keys being a hashed value of two categorical values and values of this dictionary being a similarity value.

**g**

[float or array-like of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each continuous dimension.

### Returns

**y\_pred**

[array-like of shape (n\_samples,)] Predicted class labels for all input patterns.

### 2.3.3 mixed\_data.onehot\_onln\_gfmm

General fuzzy min-max neural network trained by the batch incremental learning algorithm for mixed attribute data, in which categorical features are encoded using one-hot encoding.

```
class hbbrain.mixed_data.onehot_onln_gfmm.OneHotOnlineGFMM(theta=0.5, theta_min=1,
                                                         min_percent_overlap_cat=0.5,
                                                         gamma=1, alpha=0.9, V=None,
                                                         W=None, D=None, C=None)
```

Bases: *BaseHyperboxClassifier*

Batch incremental learning algorithm with mixed-attribute data for a general fuzzy min-max neural network, in which categorical features are encoded using the one-hot encoding method and the similarity degrees among categorical values are computed using one-hot encoding values with logical operators. The final membership value is the average of membership values for continuous features and membership values for categorical features.

See [1] for more detailed information regarding this batch incremental learning algorithm.

#### Parameters

##### **theta**

[float, optional, default=0.5] Maximum hyperbox size for continuous features.

##### **theta\_min**

[float, optional, default=1] Minimum value of the maximum hyperbox size for continuous features so that the training loop is still performed. If the value of *theta\_min* is larger than the value of *theta*, it will be automatically assigned a value equal to *theta*.

##### **gamma**

[float or ndarray of shape (n\_continuous\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each continuous feature.

##### **min\_percent\_overlap\_cat**

[float, optional, default=0.5] The minimum number of categorical values in the categorical features of the input pattern that match the values in the categorical dimensions of the winner hyperbox to be expansion.

##### **alpha**

[float, optional, default=0.9] Multiplier factor to reduce the value of maximum hyperbox size after each training loop.

##### **V**

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] A matrix stores all minimal points for continuous features of all existing hyperboxes, in which each row is a minimal point of a hyperbox.

##### **W**

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] A matrix stores all maximal points for continuous features of all existing hyperboxes, in which each row is a maximal point of a hyperbox.

##### **D**

[array-like of shape (n\_hyperboxes, n\_cat\_features)] A matrix stores all bounds for categorical features of all existing hyperboxes, in which each row is a lower bound of a hyperbox. Elements in this matrix are binary strings.

##### **C**

[array-like of shape (n\_hyperboxes,)] A vector stores all class labels corresponding to existing hyperboxes.



## References

[1]

## Examples

```
>>> from hbbrain.mixed_data.onehot_onln_gfmm import OneHotOnlineGFMM
>>> from hbbrain.datasets import load_japanese_credit
>>> X, y = load_japanese_credit()
>>> from sklearn.preprocessing import MinMaxScaler
>>> scaler = MinMaxScaler()
>>> numerical_features = [1, 2, 7, 10, 13, 14]
>>> categorical_features = [0, 3, 4, 5, 6, 8, 9, 11, 12]
>>> scaler.fit(X[:, numerical_features])
MinMaxScaler()
>>> X[:, numerical_features] = scaler.transform(X[:, numerical_features])
>>> clf = OneHotOnlineGFMM(theta=0.1, min_percent_overlap_cat=0.6)
>>> clf.fit(X, y, categorical_features)
>>> print("Number of hyperboxes = %d"%clf.get_n_hyperboxes())
Number of hyperboxes = 236
>>> clf.predict(X[[10, 100]])
array([0, 0])
```

## Attributes

### **categorical\_features\_**

[int array of shape (n\_cat\_features,)] Indices of categorical features in the training data and hyperboxes.

### **continuous\_features\_**

[int array of shape (n\_continuous\_features,)] Indices of continuous features in the training data and hyperboxes.

### **encoder\_**

[sklearn.preprocessing.OneHotEncoder] An one-hot encoder was used to encode categorical features.

### **is\_exist\_continuous\_missing\_value**

[boolean] Is there any missing values in continuous features in the training data.

### **elapsed\_training\_time**

[float] Training time in seconds.

### **n\_passes**

[int] Number of training loops.

## Methods

<code>delay([delay_constant])</code>	Delay a time period to display hyperboxes
<code>draw_hyperbox_and_boundary([window_name, ...])</code>	Draw the existing hyperboxes and their decision boundaries among classes
<code>fit(X, y[, categorical_features])</code>	Build a general fuzzy min-max neural network from the training set (X, y) using the original incremental learning algorithm for mixed attribute data, in which categorical features are encoded using one-hot encoding.
<code>get_n_hyperboxes()</code>	Get number of hyperboxes in the trained hyperbox-based model
<code>get_params([deep])</code>	Get parameters for this estimator.
<code>get_sample_explanation(x)</code>	Get useful information for explaining the reason behind the predicted result for the input pattern represented by upper and lower bounds for continuous features together with the bound for categorical feature.
<code>initialise_canvas_graph([n_dims, ...])</code>	Initialise a canvas to draw hyperboxes
<code>is_satisfied_cat_expansion_conds(xd, Dj, ...)</code>	Check whether the expansion condition for categorical features $xd$ of an input pattern can be covered by categorical features of the hyperbox $B_j$ with the categorical features stored in $D_j$ .
<code>predict(X)</code>	Predict class labels for samples in X.
<code>predict_proba(X)</code>	Predict class probabilities of the input samples X including both continuous and categorical features.
<code>predict_with_membership(X)</code>	Predict class membership values of the input samples X including both categorical and continuous features.
<code>score(X, y[, sample_weight])</code>	Return the mean accuracy on the given test data and labels.
<code>set_params(**params)</code>	Set the parameters of this estimator.
<code>show_sample_explanation(xl, xu, ...[, ...])</code>	Show explanation for predicted results of an input pattern under the form of parallel coordinates or hyperboxes in 2D or 3D planes.
<code>simple_pruning(X_val, y_val[, ...])</code>	Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox.

**fit**(X, y, categorical\_features=None)

Build a general fuzzy min-max neural network from the training set (X, y) using the original incremental learning algorithm for mixed attribute data, in which categorical features are encoded using one-hot encoding.

### Parameters

#### X

[array-like of shape (n\_samples, n\_features) or (2\*n\_samples, n\_features)] The training input samples including both continuous and categorical features. If the number of rows in X is 2\*n\_samples, the first n\_samples rows contain lower bounds of input patterns and the rest n\_samples rows contain upper bounds.

#### y

[array-like of shape (n\_samples,)] The class labels.

#### categorical\_features

[a list of int, optional, default=None] Indices of categorical features in the training set. If

None, there is no categorical feature.

#### Returns

**self**  
[object.] Fitted estimator.

#### **get\_n\_hyperboxes()**

Get number of hyperboxes in the trained hyperbox-based model

#### Returns

**int**  
Number of hyperboxes in the trained hyperbox-based classifier.

#### **get\_sample\_explanation(*x*)**

Get useful information for explaining the reason behind the predicted result for the input pattern represented by upper and lower bounds for continuous features together with the bound for categorical feature.

#### Parameters

**x**  
[ndarray of shape (n\_feature,)] The input pattern which needs to be explained includes both continuous features and categorical features.

#### Returns

**y\_pred**  
[int] The predicted class of the input pattern

**dict\_mem\_val\_classes**  
[dictionary] A dictionary stores all membership values for all classes. The key is class label and the value is the corresponding membership value.

**dict\_min\_point\_classes**  
[dictionary] A dictionary stores all minimal points of hyperboxes having the maximum membership value for each class. The key is the class label and the value is the minimal points of the hyperbox corresponding to that class

**dict\_max\_point\_classes**  
[dictionary] A dictionary stores all maximal points of hyperboxes having the maximum membership value for each class. The key is the class label and the value is the maximal points of the hyperbox corresponding to that class.

**dict\_cat\_point\_classes: dictionary**  
A dictionary stores all categorical features of hyperboxes having the maximum membership value for each class. The key is the class label and the value is the bound of categorical features of the hyperbox corresponding to that class.

#### **is\_satisfied\_cat\_expansion\_conds(*xd*, *Dj*, *n\_cat\_features*)**

Check whether the expansion condition for categorical features *xd* of an input pattern can be covered by categorical features of the hyperbox  $B_j$  with the categorical features stored in *Dj*.

#### Parameters

**xd**  
[array-like of shape (n\_cat\_features,)] Categorical features of an input pattern.

**Dj**  
[array-like of shape (n\_cat\_features,)] Categorical features bounds of the hyperbox  $B_j$  which can be extended to cover the input pattern.

**n\_cat\_features**

[int] Number of categorical features in the training set.

**Returns****bool**

If True, the categorical features in  $D_j$  are satisfied with the expansion conditions for the categorical feature so that it can be expanded to cover the input pattern. Otherwise, the conditions for the categorical features are not met.

**predict(X)**

Predict class labels for samples in  $X$ .

---

**Note:** In the case there are many winner hyperboxes representing different class labels but with the same membership value with respect to the input pattern  $X_i$ , an additional criterion based on the minimum Manhattan distance between continuous features of  $X_i$  and the central points of continuous features of winner hyperboxes are used to find the final winner hyperbox that its class label is used for predicting the class label of the input pattern  $X_i$ . If there are only categorical features but many winner hyperboxes belonging to different classes, a random selection will be used to choose the final class label.

---

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The data matrix for which we want to predict the targets.

**Returns****y\_pred**

[ndarray of shape (n\_samples,)] Vector containing the predictions. In binary and multiclass problems, this is a vector containing  $n\_samples$ .

**predict\_proba(X)**

Predict class probabilities of the input samples  $X$  including both continuous and categorical features.

The predicted class probability is the fraction of the membership value of the representative hyperbox of that class and the sum of all membership values of all representative hyperboxes of all classes.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The input samples.

**Returns****proba**

[ndarray of shape (n\_samples, n\_classes)] The class probabilities of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

**predict\_with\_membership(X)**

Predict class membership values of the input samples  $X$  including both categorical and continuous features.

The predicted class membership value is the membership value of the representative hyperbox of that class.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The input samples.

**Returns**

**mem\_vals**

[ndarray of shape (n\_samples, n\_classes)] The class membership values of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

**simple\_pruning**(*X\_val*, *y\_val*, *acc\_threshold=0.5*, *keep\_empty\_boxes=False*)

Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox.

**Parameters****X\_val**

[array-like of shape (n\_samples, n\_features)] The data matrix contains both continuous and categorical features of validation patterns.

**y\_val**

[ndarray of shape (n\_samples,)] A vector contains the true class label corresponding to each validation pattern.

**acc\_threshold**

[float, optional, default=0.5] The minimum accuracy for each hyperbox to be kept unchanged.

**keep\_empty\_boxes**

[boolean, optional, default=False] Whether to keep the hyperboxes which do not join the prediction process on the validation set. If True, keep them, otherwise the decision for keeping or removing based on the classification accuracy on the validation dataset.

**Returns****self**

A hyperbox-based model with the low-qualified hyperboxes pruned.

**hbbrain.mixed\_data.onehot\_onln\_gfmm.impute\_missing\_value\_cat\_feature**(*Xd*)

Impute missing values of categorical features in *Xd* by a constant value.

**Parameters****Xd**

[array-like of shape (n\_samples, n\_cat\_features)] Categorical features.

**Returns****Xd**

[array-like of shape (n\_samples, n\_cat\_features)] Categorical features after doing data imputation.

**hbbrain.mixed\_data.onehot\_onln\_gfmm.one\_hot\_encoding\_cat\_feature**(*X*, *categorical\_features*, *encodings=None*)

Encode categorical features by the one-hot encoding method.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] Input patterns.

**categorical\_features**

[array-like of shape (n\_cat\_features, )] Indices of categorical features.

**encodings**

[a list of objects, optional, default=None] Storing a list of one-hot encoders each for a categorical feature.

**Returns**

**X\_out**

[array-like of shape (n\_samples, n\_features)] An input data matrix with the encoded categorical features.

**encodings\_out**

[TYPE] An one-hot encoder was used to encode categorical features.

`hbbrain.mixed_data.onehot_onln_gfmm.predict_onehot_cat_feature_manhattan(V, W, D, C, Xl, Xu, Xd, g=1)`

Predict class labels for mixed-class samples in  $X$  represented in the form of intervals  $[Xl, Xu, Xd]$ . This is a common function to determine the right class labels for  $X$  wrt a trained hyperbox-based classifier represented by  $[V, W, D, C]$ . It uses the winner-takes-all principle to predict class labels for each sample in  $X$  by assigning the class label of the sample to the class label of the hyperbox with the maximum membership value to that sample. It will use a Manhattan distance for continuous features in the case of many hyperboxes with different classes having the same maximum membership value. If there is no continuous feature the random selection will be used for the case of many winner hyperboxes.

**Parameters****Xl**

[array-like of shape (n\_samples, n\_continuous\_features)] Lower bounds of continuous features of all input samples. If None, there are no continuous features.

**Xu**

[array-like of shape (n\_samples, n\_continuous\_features)] Lower bounds of continuous features of all input samples. If None, there are no continuous features.

**Xd**

[array-like of shape (n\_samples, n\_cat\_features)] Bounds of categorical features of all input patterns. If None, there are no categorical features.

**V**

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] Minimum points of all continuous features of the existing hyperboxes in the trained model. If None, there are no continuous features.

**W**

[array-like of shape (n\_hyperboxes, n\_continuous\_features)] Maximum points of all continuous features of the existing hyperboxes in the trained model. If None, there are no continuous features.

**D**

[array-like of shape (n\_hyperboxes, n\_cat\_features)] Bounds of all categorical features of the existing hyperboxes in the trained model. If None, there are no categorical features.

**C**

[array-like of shape (n\_hyperboxes,)] Class labels of all existing hyperboxes corresponding to the values stored in  $V$ ,  $W$ , and  $D$ .

**g**

[float or ndarray of shape (n\_continuous\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each continuous dimension.

**Returns****y\_pred**

[ndarray of shape (n\_samples,)] A vector contains the predictions. In binary and multiclass problems, this is a vector containing  $n\_samples$ .

## 2.4 batch learners

### 2.4.1 batch\_learner.agglo\_gfmm

General fuzzy min-max neural network trained by the agglomerative learning algorithm with full similarity matrix.

```
class hbbrain.numerical_data.batch_learner.agglo_gfmm.AgglomerativeLearningGFMM(theta=0.5,
                                                                              gamma=1,
                                                                              min_simil=0.5,
                                                                              simil_measure='mid',
                                                                              asimil_type='max',
                                                                              is_draw=False)
```

Bases: *BaseGFMMClassifier*

Agglomerative learning algorithm with full similarity matrix for a general fuzzy min-max neural network with numerical data.

See [1] for more detailed information regarding this learning algorithm.

---

**Note:** Note that this implementation uses the accelerated mechanism presented in [2] to accelerate the improved online learning algorithm.

---

#### Parameters

##### **theta**

[float, optional, default=0.5] Maximum hyperbox size for numerical features.

##### **gamma**

[float or ndarray of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each continuous feature.

##### **min\_simil**

[float, optional, default=0.5] Minimum similarity threshold so that two hyperboxes are agglomerated.

##### **simil\_measure**

[{'short', 'long', 'mid'}, optional, default='mid'] Type of similarity measures is used to compute similarity between two hyperboxes. It can get values of shorted gap, middel gap or longest gap between two hyperboxes.

##### **asimil\_type**

[{'max', 'min'}, optional, default='max'] Type of similarity measures is used in the case of *simil\_measure* getting a value of *mid*. It can be the maximum or minimum values of two dissimilar values of a similarity measure based on middle distance.

##### **is\_draw**

[boolean, optional, default=False] Whether the construction of hyperboxes can be progressively shown during the training process on a canvas window.

## References

[1], [2]

## Examples

```
>>> from hbbrain.numerical_data.batch_learner.agglo_gfmm import _
↳ AgglomerativeLearningGFMM
>>> from sklearn.datasets import load_iris
>>> X, y = load_iris(return_X_y=True)
>>> from sklearn.preprocessing import MinMaxScaler
>>> scaler = MinMaxScaler()
>>> scaler.fit(X)
MinMaxScaler()
>>> X = scaler.transform(X)
>>> clf = AgglomerativeLearningGFMM(theta=0.1, min_simil=0.8, simil_measure='short')
>>> clf.fit(X, y)
>>> print("Number of hyperboxes = %d"%clf.get_n_hyperboxes())
Number of hyperboxes = 65
>>> clf.predict(X[[10, 50, 100]])
array([0, 1, 2])
```

## Attributes

### V

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all minimal points for numerical features of all existing hyperboxes, in which each row is a minimal point of a hyperbox.

### W

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all maximal points for numerical features of all existing hyperboxes, in which each row is a minimal point of a hyperbox.

### C

[array-like of shape (n\_hyperboxes,)] A vector stores all class labels corresponding to existing hyperboxes.

### N\_samples

[array-like of shape (n\_hyperboxes,)] A vector stores the number of samples fully included in each existing hyperbox.

### is\_exist\_missing\_value

[boolean] Is there any missing values in continuous features in the training data.

### elapsed\_training\_time

[float] Training time in seconds.



## Methods

<code>delay([delay_constant])</code>	Delay a time period to display hyperboxes
<code>draw_hyperbox_and_boundary([window_name, ...])</code>	Draw the existing hyperboxes and their decision boundaries among classes
<code><i>fit</i>(X, y)</code>	Fit the model according to the given training data using the agglomerative learning algorithm using full similarity matrix.
<code>get_n_hyperboxes()</code>	Get number of hyperboxes in the trained hyperbox-based model
<code>get_params([deep])</code>	Get parameters for this estimator.
<code><i>get_sample_explanation</i>(xl, xu[, ...])</code>	Get useful information for explaining the reason behind the predicted result for the input pattern
<code>initialise_canvas_graph([n_dims, ...])</code>	Initialise a canvas to draw hyperboxes
<code><i>predict</i>(X[, type_boundary_handling])</code>	Predict class labels for samples in X.
<code>predict_proba(X)</code>	Predict class probabilities of the input samples X.
<code>predict_with_membership(X)</code>	Predict class membership values of the input samples X.
<code>score(X, y[, sample_weight])</code>	Return the mean accuracy on the given test data and labels.
<code>set_params(**params)</code>	Set the parameters of this estimator.
<code>show_sample_explanation(xl, xu, ...[, ...])</code>	Show explanation for predicted results of an input pattern under the form of parallel coordinates or hyperboxes in 2D or 3D planes.
<code><i>simple_pruning</i>(Xl_val, Xu_val, y_val[, ...])</code>	Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox

### ***fit*(X, y)**

Fit the model according to the given training data using the agglomerative learning algorithm using full similarity matrix.

#### Parameters

##### **X**

[array-like of shape (n\_samples, n\_features)] Training vector, where *n\_samples* is the number of samples and *n\_features* is the number of features.

##### **y**

[array-like of shape (n\_samples,)] Target vector relative to X.

#### Returns

##### **self**

[object] Fitted hyperbox-based model.

### ***get\_sample\_explanation*(xl, xu, type\_boundary\_handling=1)**

Get useful information for explaining the reason behind the predicted result for the input pattern

#### Parameters

##### **xl**

[ndarray of shape (n\_feature,)] Minimum point of the input pattern which needs to be explained.

##### **xu**

[ndarray of shape (n\_feature,)] Maximum point of the input pattern which needs to be explained.

**type\_boundary\_handling**

[int, optional, default=PROBABILITY\_MEASURE (aka 1)] The way of handling samples located on the boundary.

**Returns****y\_pred**

[int] The predicted class of the input pattern

**dict\_mem\_val\_classes**

[dictionary] A dictionary stores all membership values for all classes. The key is class label and the value is the corresponding membership value.

**dict\_min\_point\_classes**

[dictionary] A dictionary stores all minimal points of hyperboxes having the maximum membership value for each class. The key is the class label and the value is the minimal points of all hyperboxes corresponding to each class

**dict\_max\_point\_classes**

[dictionary] A dictionary stores all maximal points of hyperboxes having the maximum membership value for each class. The key is the class label and the value is the maximal points of all hyperboxes corresponding to each class

**predict**(*X*, *type\_boundary\_handling=1*)

Predict class labels for samples in *X*.

---

**Note:** In the case there are many winner hyperboxes representing different class labels but with the same membership value with respect to the input pattern  $X_i$ , an additional criterion based on the probability generated by number of samples included in winner hyperboxes and membership values or the Manhattan distance between the central point of winner hyperboxes and the input sample is used to find the final winner hyperbox that its class label is used for predicting the class label of the input pattern  $X_i$ .

---

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The data matrix for which we want to predict the targets.

**type\_boundary\_handling**

[int, optional, default=PROBABILITY\_MEASURE (aka 1)] The way of handling many winner hyperboxes, i.e., PROBABILITY\_MEASURE or MANHATTAN\_DIS

**Returns****y\_pred**

[ndarray of shape (n\_samples,)] Vector containing the predictions. In binary and multiclass problems, this is a vector containing *n\_samples*.

**simple\_pruning**(*Xl\_val*, *Xu\_val*, *y\_val*, *acc\_threshold=0.5*, *keep\_empty\_boxes=False*,  
*type\_boundary\_handling=1*)

Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox

**Parameters****Xl\_val**

[array-like of shape (n\_samples, n\_features)] The data matrix contains lower bounds of validation patterns.

**Xu\_val**

[array-like of shape (n\_samples, n\_features)] The data matrix contains upper bounds of validation patterns.

**y\_val**

[ndarray of shape (n\_samples,)] A vector contains the true class label corresponding to each validation pattern.

**acc\_threshold**

[float, optional, default=0.5] The minimum accuracy for each hyperbox to be kept unchanged.

**keep\_empty\_boxes**

[boolean, optional, default=False] Whether to keep the hyperboxes which do not join the prediction process on the validation set. If True, keep them, else the decision for keeping or removing based on the classification accuracy on the validation dataset

**type\_boundary\_handling**

[int, optional, default=PROBABILITY\_MEASURE (aka 1)] The way of handling samples located on the boundary.

**Returns****self**

A hyperbox-based model with the low-qualified hyperboxes pruned.

## 2.4.2 batch\_learner.accel\_agglo\_gfmm

General fuzzy min-max neural network trained by the accelerated agglomerative learning algorithm.

```
class hbbrain.numerical_data.batch_learner.accel_agglo_gfmm.AccelAgglomerativeLearningGFMM(theta=0.5,
                                                                                       gamma=1,
                                                                                       min_simil=0.5,
                                                                                       simil_measure=
                                                                                       asimil_type='m
                                                                                       is_draw=False)
```

Bases: *BaseGFMMClassifier*

Accelerated agglomerative learning algorithm for a general fuzzy min-max neural network with numerical data.

See [1] for more detailed information regarding this learning algorithm.

---

**Note:** This implementation uses the accelerated mechanism presented in [2] to accelerate the improved online learning algorithm.

---

**Parameters****theta**

[float, optional, default=0.5] Maximum hyperbox size for numerical features.

**gamma**

[float or ndarray of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each continuous feature.

**min\_simil**

[float, optional, default=0.5] Minimum similarity threshold so that two hyperboxes are agglomerated.

**simil\_measure**

[{'short', 'long', 'mid'}, optional, default='mid'] Type of similarity measures is used to compute similarity between two hyperboxes. It can get values of shorted gap, middel gap or longest gap between two hyperboxes.

**asimil\_type**

[{'max', 'min'}, optional, default='max'] Type of similarity measures is used in the case of *simil\_measure* getting a value of *mid*. It can be the maximum or minimum values of two dissimilar values of a similarity measure based on middle distance.

**is\_draw**

[boolean, optional, default=False] Whether the construction of hyperboxes can be progressively shown during the training process on a canvas window.

**References**

[1], [2]

**Examples**

```
>>> from hbbrain.numerical_data.batch_learner.accel_agglo_gfmm import _
↳ AccelAgglomerativeLearningGFMM
>>> from sklearn.datasets import load_iris
>>> X, y = load_iris(return_X_y=True)
>>> from sklearn.preprocessing import MinMaxScaler
>>> scaler = MinMaxScaler()
>>> scaler.fit(X)
MinMaxScaler()
>>> X = scaler.transform(X)
>>> clf = AccelAgglomerativeLearningGFMM(theta=0.1, min_simil=0.8, simil_measure=
↳ 'short')
>>> clf.fit(X, y)
>>> print("Number of hyperboxes = %d"%clf.get_n_hyperboxes())
Number of hyperboxes = 69
>>> clf.predict(X[[10, 50, 100]])
array([0, 1, 2])
```

**Attributes****V**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all minimal points for numerical features of all existing hyperboxes, in which each row is a minimal point of a hyperbox.

**W**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all maximal points for numerical features of all existing hyperboxes, in which each row is a minimal point of a hyperbox.

**C**

[array-like of shape (n\_hyperboxes,)] A vector stores all class labels corresponding to existing hyperboxes.

**N\_samples**

[array-like of shape (n\_hyperboxes,)] A vector stores the number of samples fully included in each existing hyperbox.

**is\_exist\_missing\_value**

[boolean] Is there any missing values in continuous features in the training data.

**elapsed\_training\_time**

[float] Training time in seconds.

**Methods**

<code>delay([delay_constant])</code>	Delay a time period to display hyperboxes
<code>draw_hyperbox_and_boundary([window_name, ...])</code>	Draw the existing hyperboxes and their decision boundaries among classes
<code><i>fit</i>(X, y)</code>	Fit the model according to the given training data using the accelerated agglomerative learning algorithm.
<code>get_n_hyperboxes()</code>	Get number of hyperboxes in the trained hyperbox-based model
<code>get_params([deep])</code>	Get parameters for this estimator.
<code><i>get_sample_explanation</i>(xl, xu[, ...])</code>	Get useful information for explaining the reason behind the predicted result for the input pattern
<code>initialise_canvas_graph([n_dims, ...])</code>	Initialise a canvas to draw hyperboxes
<code><i>predict</i>(X[, type_boundary_handling])</code>	Predict class labels for samples in X.
<code>predict_proba(X)</code>	Predict class probabilities of the input samples X.
<code>predict_with_membership(X)</code>	Predict class membership values of the input samples X.
<code>score(X, y[, sample_weight])</code>	Return the mean accuracy on the given test data and labels.
<code>set_params(**params)</code>	Set the parameters of this estimator.
<code>show_sample_explanation(xl, xu, ...[, ...])</code>	Show explanation for predicted results of an input pattern under the form of parallel coordinates or hyperboxes in 2D or 3D planes.
<code><i>simple_pruning</i>(Xl_val, Xu_val, y_val[, ...])</code>	Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox

***fit*(X, y)**

Fit the model according to the given training data using the accelerated agglomerative learning algorithm.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] Training vector, where *n\_samples* is the number of samples and *n\_features* is the number of features.

**y**

[array-like of shape (n\_samples,)] Target vector relative to X.

**Returns****self**

[object] Fitted hyperbox-based model.

***get\_sample\_explanation*(xl, xu, type\_boundary\_handling=1)**

Get useful information for explaining the reason behind the predicted result for the input pattern

**Parameters****xl**

[ndarray of shape (n\_feature,)] Minimum point of the input pattern which needs to be explained.

**xu**

[ndarray of shape (n\_feature,)] Maximum point of the input pattern which needs to be explained.

**type\_boundary\_handling**

[int, optional, default=PROBABILITY\_MEASURE (aka 1)] The way of handling samples located on the boundary.

**Returns****y\_pred**

[int] The predicted class of the input pattern

**dict\_mem\_val\_classes**

[dictionary] A dictionary stores all membership values for all classes. The key is class label and the value is the corresponding membership value.

**dict\_min\_point\_classes**

[dictionary] A dictionary stores all minimal points of hyperboxes having the maximum membership value for each class. The key is the class label and the value is the minimal points of all hyperboxes corresponding to each class

**dict\_max\_point\_classes**

[dictionary] A dictionary stores all maximal points of hyperboxes having the maximum membership value for each class. The key is the class label and the value is the maximal points of all hyperboxes corresponding to each class

**predict**(X, type\_boundary\_handling=1)

Predict class labels for samples in X.

---

**Note:** In the case there are many winner hyperboxes representing different class labels but with the same membership value with respect to the input pattern  $X_i$ , an additional criterion based on the probability generated by number of samples included in winner hyperboxes and membership values or the Manhattan distance between the central point of winner hyperboxes and the input sample is used to find the final winner hyperbox that its class label is used for predicting the class label of the input pattern  $X_i$ .

---

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The data matrix for which we want to predict the targets.

**type\_boundary\_handling**

[int, optional, default=PROBABILITY\_MEASURE (aka 1)] The way of handling many winner hyperboxes, i.e., PROBABILITY\_MEASURE or MANHATTAN\_DIS

**Returns****y\_pred**

[ndarray of shape (n\_samples,)] Vector containing the predictions. In binary and multiclass problems, this is a vector containing  $n\_samples$ .

**simple\_pruning**(*Xl\_val*, *Xu\_val*, *y\_val*, *acc\_threshold=0.5*, *keep\_empty\_boxes=False*,  
*type\_boundary\_handling=1*)

Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox

#### Parameters

##### **Xl\_val**

[array-like of shape (n\_samples, n\_features)] The data matrix contains lower bounds of validation patterns.

##### **Xu\_val**

[array-like of shape (n\_samples, n\_features)] The data matrix contains upper bounds of validation patterns.

##### **y\_val**

[ndarray of shape (n\_samples,)] A vector contains the true class label corresponding to each validation pattern.

##### **acc\_threshold**

[float, optional, default=0.5] The minimum accuracy for each hyperbox to be kept unchanged.

##### **keep\_empty\_boxes**

[boolean, optional, default=False] Whether to keep the hyperboxes which do not join the prediction process on the validation set. If True, keep them, else the decision for keeping or removing based on the classification accuracy on the validation dataset

##### **type\_boundary\_handling**

[int, optional, default=PROBABILITY\_MEASURE (aka 1)] The way of handling samples located on the boundary.

#### Returns

##### **self**

A hyperbox-based model with the low-qualified hyperboxes pruned.

## 2.5 ensemble learners

### 2.5.1 ensemble\_learner.base\_bagging

Base functions and classes for bagging models using hyperbox-based models.

**class** hbbrain.numerical\_data.ensemble\_learner.base\_bagging.**BaseBagging**(*base\_estimator=None*,  
*n\_estimators=10*, \*,  
*max\_samples=0.5*,  
*bootstrap=False*,  
*class\_balanced=False*,  
*n\_jobs=None*,  
*random\_state=None*)

Bases: [BaseEnsemble](#)

Base class for Bagging meta-estimator. Warning: This class should not be used directly. Use derived classes instead.

#### Attributes

##### ***estimators\_samples\_***

The subset of drawn samples for each base estimator.

## Methods

<code>fit(X, y)</code>	Build a Bagging ensemble of estimators from the training set (X, y).
<code>get_n_hyperboxes()</code>	Get total number of hyperboxes in all base learners.
<code>get_params([deep])</code>	Get parameters for this estimator.
<code>set_params(**params)</code>	Set the parameters of this estimator.
<code>simple_pruning_base_estimators(X_val, y_val)</code>	Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox.

### **property estimators\_samples\_**

The subset of drawn samples for each base estimator. Returns a dynamically generated list of indices identifying the samples used for fitting each member of the ensemble, i.e., the in-bag samples.

---

**Note:** The list is re-created at each call to the property in order to reduce the object memory footprint by not storing the sampling data. Thus fetching the property may be slower than expected.

---

### **fit(X, y)**

Build a Bagging ensemble of estimators from the training set (X, y).

#### **Parameters**

##### **X**

[array-like of shape (n\_samples, n\_features)] The training input samples.

##### **y**

[array-like of shape (n\_samples,)] The real class labels

#### **Returns**

##### **self**

[object] Fitted estimator.

### **get\_n\_hyperboxes()**

Get total number of hyperboxes in all base learners.

#### **Returns**

##### **n\_hyperboxes**

[int] Total number of hyperboxes in all base learners.

### **simple\_pruning\_base\_estimators(X\_val, y\_val, acc\_threshold=0.5, keep\_empty\_boxes=False)**

Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox. This operation is applied for all base estimators.

#### **Parameters**

##### **X\_val**

[array-like of shape (n\_samples, n\_features)] The data matrix contains validation patterns.

##### **y\_val**

[ndarray of shape (n\_samples,)] A vector contains the true class label corresponding to each validation pattern.

##### **acc\_threshold**

[float, optional, default=0.5] The minimum accuracy for each hyperbox to be kept unchanged.



**keep\_empty\_boxes**

[boolean, optional, default=False] Whether to keep the hyperboxes which do not join the prediction process on the validation set. If True, keep them, else the decision for keeping or removing based on the classification accuracy on the validation dataset

**Returns****self**

A bagging model with base estimators pruned.

## 2.5.2 ensemble\_learner.base\_cross\_val\_bagging

Base functions and classes for k-fold cross validation bagging models using hyperbox-based models.

```
class hbbrain.numerical_data.ensemble_learner.base_cross_val_bagging.BaseCrossValBagging(base_estimator=None,
base_estimator_params=None,
n_estimators=10,
max_samples=0.5,
bootstrap=True,
strap=False,
class_balanced=False,
n_iter=10,
scoring='accuracy',
k_fold=5,
n_jobs=None,
random_state=None)
```

Bases: [BaseEnsemble](#)

Base class for cross validation Bagging meta-estimator, in which base estimators are built using k-fold cross-validation and random search for parameter tuning. Warning: This class should not be used directly. Use derived classes instead.

**Attributes****[estimators\\_samples\\_](#)**

The subset of drawn samples for each base estimator.

**Methods**

<a href="#">fit</a> (X, y)	Build a Bagging ensemble of estimators from the training set (X, y).
<a href="#">get_n_hyperboxes</a> ()	Get total number of hyperboxes in all base learners.
<a href="#">get_params</a> ([deep])	Get parameters for this estimator.
<a href="#">set_params</a> (**params)	Set the parameters of this estimator.
<a href="#">simple_pruning_base_estimators</a> (X_val, y_val)	Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox.

**property [estimators\\_samples\\_](#)**

The subset of drawn samples for each base estimator. Returns a dynamically generated list of indices identifying the samples used for fitting each member of the ensemble, i.e., the in-bag samples.

---

**Note:** The list is re-created at each call to the property in order to reduce the object memory footprint by not storing the sampling data. Thus fetching the property may be slower than expected.

---

**fit**(X, y)

Build a Bagging ensemble of estimators from the training set (X, y).

**Parameters**

**X**

[array-like of shape (n\_samples, n\_features)] The training input samples.

**y**

[array-like of shape (n\_samples,)] The real class labels

**Returns**

**self**

[object] Fitted estimator.

**get\_n\_hyperboxes**()

Get total number of hyperboxes in all base learners.

**Returns**

**n\_hyperboxes**

[int] Total number of hyperboxes in all base learners.

**simple\_pruning\_base\_estimators**(X\_val, y\_val, acc\_threshold=0.5, keep\_empty\_boxes=False)

Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox. This operation is applied for all base estimators.

**Parameters**

**X\_val**

[array-like of shape (n\_samples, n\_features)] The data matrix contains validation patterns.

**y\_val**

[ndarray of shape (n\_samples,)] A vector contains the true class label corresponding to each validation pattern.

**acc\_threshold**

[float, optional, default=0.5] The minimum accuracy for each hyperbox to be kept unchanged.

**keep\_empty\_boxes**

[boolean, optional, default=False] Whether to keep the hyperboxes which do not join the prediction process on the validation set. If True, keep them, else the decision for keeping or removing based on the classification accuracy on the validation dataset

**Returns**

**self**

A bagging model with base estimators pruned.

### 2.5.3 ensemble\_learner.decision\_comb\_bagging

A bagging of many base hyperbox-based models trained on the full set of features and a subset of samples. The predicted class is computed based on the voting mechanism of decisions of base models.

**class** `hbbrain.numerical_data.ensemble_learner.decision_comb_bagging.DecisionCombinationBagging`(*base\_estimator*, *n\_estimators*, *max\_samples*, *bootstrap*, *class\_balanced*, *n\_jobs*, *random\_state*)

Bases: `ClassifierMixin`, [\*BaseBagging\*](#)

A Bagging classifier of base hyperbox-based models trained on a full set of features and a subset of samples.

A decision combination Bagging classifier of hyperbox-based models is an ensemble meta-estimator that fits base hyperbox-based classifiers each on random subsets of the original samples and then aggregate their individual predictions by voting to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a single estimator, by introducing randomization into its construction procedure and then making an ensemble out of it. This algorithm encompasses several works from the literature. When random subsets of the dataset are drawn as random subsets of the samples, then this algorithm is known as Pasting [1]. If samples are drawn with replacement, then the method is known as Bagging [2]. See [3] for more detailed information regarding the combination of base hyperbox-based models.

#### Parameters

##### **base\_estimator**

[object, default=None] The base estimator to fit on random subsets of the dataset. If None, then the base estimator is a [\*OnlineGFMM\*](#).

##### **n\_estimators**

[int, default=10] The number of base estimators in the ensemble.

##### **max\_samples**

[int or float, default=0.5] The number of samples to draw from X to train each base estimator (with no replacement by default, see *bootstrap* for more details). - If int, then draw *max\_samples* samples. - If float, then draw *max\_samples* \* *X.shape[0]* samples.

##### **bootstrap**

[bool, default=False] Whether samples are drawn with replacement. If False, sampling without replacement is performed.

##### **class\_balanced**

[bool, default=False] Whether samples are drawn without replacement to build a final subset with the equal number of samples among classes.

##### **n\_jobs**

[int, default=1] The number of jobs to run in parallel for both [\*fit\(\)\*](#) and [\*predict\(\)\*](#). None means 1 unless in a `joblib.parallel_backend` context. -1 means using all processors.

##### **random\_state**

[int, RandomState instance or None, default=None] Controls the random resampling of the original dataset (sample wise and feature wise). If the base estimator accepts a *random\_state* attribute, a different seed is generated for each instance in the ensemble. Pass an int for reproducible output across multiple function calls.

## References

[1], [2], [3]

## Examples

```
>>> from hbbrain.numerical_data.incremental_learner.iol_gfmm import _
↳ ImprovedOnlineGFMM
>>> from hbbrain.numerical_data.ensemble_learner.decision_comb_bagging import _
↳ DecisionCombinationBagging
>>> from sklearn.datasets import make_classification
>>> X, y = make_classification(n_samples=100, n_features=4,
...                           n_informative=2, n_redundant=0,
...                           random_state=0, shuffle=False)
>>> from sklearn.preprocessing import MinMaxScaler
>>> scaler = MinMaxScaler()
>>> scaler.fit(X)
MinMaxScaler()
>>> X = scaler.transform(X)
>>> clf = DecisionCombinationBagging(base_estimator=ImprovedOnlineGFMM(0.1),
...                                  n_estimators=10, random_state=0).fit(X, y)
>>> clf.predict([[1, 0.6, 0.5, 0.2]])
array([1])
```

## Attributes

### **base\_estimator\_**

[estimator] The base estimator from which the ensemble is grown.

### **estimators\_**

[list of estimators] The collection of fitted base estimators.

### **estimators\_samples\_**

[list of arrays] The subset of drawn samples for each base estimator.

### **classes\_**

[ndarray of shape (n\_classes,)] The classes labels.

### **n\_classes\_**

[int or list] The number of classes.

## Methods

<code>fit(X, y)</code>	Build a Bagging ensemble of estimators from the training set (X, y).
<code>get_n_hyperboxes()</code>	Get total number of hyperboxes in all base learners.
<code>get_params([deep])</code>	Get parameters for this estimator.
<code>predict(X)</code>	Predict class for X.
<code>predict_proba(X)</code>	Predict class probabilities for X.
<code>predict_with_membership(X)</code>	Predict class memberships for X.
<code>score(X, y[, sample_weight])</code>	Return the mean accuracy on the given test data and labels.
<code>set_params(**params)</code>	Set the parameters of this estimator.
<code>simple_pruning_base_estimators(X_val, y_val)</code>	Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox.

### **fit(X, y)**

Build a Bagging ensemble of estimators from the training set (X, y).

#### **Parameters**

**X**

[array-like of shape (n\_samples, n\_features)] The training input samples.

**y**

[array-like of shape (n\_samples,)] The real class labels

#### **Returns**

**self**

[object] Fitted estimator.

### **predict(X)**

Predict class for X.

The predicted class of an input sample is computed as the class with the highest mean predicted probability using voting.

#### **Parameters**

**X**

[array-like of shape (n\_samples, n\_features)] The testing input samples.

#### **Returns**

**y**

[ndarray of shape (n\_samples,)] The predicted classes.

### **predict\_proba(X)**

Predict class probabilities for X.

The predicted class probabilities of an input sample are computed as the mean predicted class probabilities of the hyperbox-based learners in the ensemble model. The class probability of a single hyperbox-based learner is the fraction of the membership value of the representative hyperbox of that class and the sum of all membership values of all representative hyperboxes of all classes.

#### **Parameters**

**X**

[array-like of shape (n\_samples, n\_features)] The input samples for prediction.

**Returns****all\_probas**

[ndarray of shape (n\_samples, n\_classes)] The class probabilities of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

**predict\_with\_membership(X)**

Predict class memberships for X.

The predicted class memberships of an input sample are computed as the mean predicted class memberships of the hyperbox-based learners in the ensemble model. The class membership of a single hyperbox-based learner is the membership from the input X to the representative hyperbox of that class to join the prediction procedure.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The input samples for prediction.

**Returns****mem\_vals**

[ndarray of shape (n\_samples, n\_classes)] The class memberships of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

## 2.5.4 ensemble\_learner.decision\_comb\_cross\_val\_bagging

A bagging of many base hyperbox-based models trained on the full set of features and a subset of samples. Each base learner is trained by random search-based hyper-parameter tuning and k-fold cross-validation. The predicted class is computed based on the voting mechanism of decisions of base models.

**class** hbbrain.numerical\_data.ensemble\_learner.decision\_comb\_cross\_val\_bagging.**DecisionCombinationCrossV**

Bases: ClassifierMixin, [BaseCrossValBagging](#)

A Bagging classifier of base hyperbox-based models trained on a full set of features and a subset of samples, in which each base learner is trained by random search-based hyper-parameter tuning and k-fold cross-validation.

A decision combination cross-validation Bagging classifier of hyperbox-based models is an ensemble meta-estimator that fits base hyperbox-based classifiers each on random subsets of the original samples using random search-based hyper-parameter tuning and k-fold cross-validation, and then aggregate their individual predictions by voting to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a single estimator, by introducing randomization into its construction procedure and then making an ensemble out of it. This algorithm encompasses several works from the literature. When random subsets of the dataset are drawn as random subsets of the samples, then this algorithm is known as Pasting [1]. If samples are drawn

with replacement, then the method is known as Bagging [2]. See [3] for more detailed information regarding the combination of base hyperbox-based models.

### Parameters

#### **base\_estimator**

[object, default=None] The base estimator to fit on random subsets of the dataset. If None, then the base estimator is a [OnlineGFMM](#).

#### **base\_estimator\_params**

[dict or list of dicts, default={}] Dictionary with parameters names (str) as keys and distributions or lists of parameters to try. If a list is given, it is sampled uniformly. If a list of dicts is given, first a dict is sampled uniformly, and then a parameter is sampled using that dict as above.

#### **n\_estimators**

[int, default=10] The number of base estimators in the ensemble.

#### **max\_samples**

[int or float, default=0.5] The number of samples to draw from X to train each base estimator (with no replacement by default, see *bootstrap* for more details). - If int, then draw *max\_samples* samples. - If float, then draw *max\_samples* \* *X.shape[0]* samples.

#### **bootstrap**

[bool, default=False] Whether samples are drawn with replacement. If False, sampling without replacement is performed.

#### **class\_balanced**

[bool, default=False] Whether samples are drawn without replacement to build a final subset with the equal number of samples among classes.

#### **n\_iter**

[int, default=10] Number of parameter settings that are sampled. *n\_iter* trades off runtime vs quality of the solution.

#### **scoring**

[str or callable default='accuracy'] Strategy to evaluate the performance of the cross-validated model on the test set. If *scoring* represents a single score, one can use: - a single string (see [The scoring parameter: defining model evaluation rules in sklearn](#)). - a callable (see [Defining your scoring strategy from metric functions](#)) that returns a single value.

#### **n\_jobs**

[int, default=1] The number of jobs to run in parallel for both *fit()* and *predict()*. None means 1 unless in a *joblib.parallel\_backend* context. -1 means using all processors.

#### **k\_fold**

[int, default=5] Determines the cross-validation splitting strategy. Possible inputs for cv are: - None, to use the default 5-fold cross validation, - integer, to specify the number of folds in a (*Stratified*)*KFold*. For integer/None inputs, if the estimator is a classifier and y is either binary or multiclass, Stratified K-Fold is used.

#### **random\_state**

[int, RandomState instance or None, default=None] Controls the random resampling of the original dataset (sample wise and feature wise). If the base estimator accepts a *random\_state* attribute, a different seed is generated for each instance in the ensemble. Pass an int for reproducible output across multiple function calls.

## References

[1], [2], [3]

## Examples

```
>>> from hbbbrain.numerical_data.incremental_learner.onln_gfmm import OnlineGFMM
>>> from hbbbrain.numerical_data.ensemble_learner.decision_comb_cross_val_bagging_
↳ import DecisionCombinationCrossValBagging
>>> from sklearn.datasets import make_classification
>>> import numpy as np
>>> X, y = make_classification(n_samples=100, n_features=4,
...                           n_informative=2, n_redundant=0,
...                           random_state=0, shuffle=False)
>>> from sklearn.preprocessing import MinMaxScaler
>>> scaler = MinMaxScaler()
>>> scaler.fit(X)
MinMaxScaler()
>>> X = scaler.transform(X)
>>> clf = DecisionCombinationCrossValBagging(base_estimator=OnlineGFMM(0.1),
...     base_estimator_params = {'theta': np.arange(0.05, 1.01,
↳ 0.05), 'theta_min':[1], 'gamma':[0.5, 1, 2, 4, 8, 16]},
...     n_estimators=10, random_state=0).fit(X, y)
>>> clf.predict([[1, 0.6, 0.5, 0.2]])
array([1])
```

## Attributes

### **base\_estimator\_**

[estimator] The base estimator from which the ensemble is grown.

### **estimators\_**

[list of estimators] The collection of fitted base estimators.

### **estimators\_samples\_**

[list of arrays] The subset of drawn samples for each base estimator.

### **classes\_**

[ndarray of shape (n\_classes,)] The classes labels.

### **n\_classes\_**

[int or list] The number of classes.



## Methods

<code>fit(X, y)</code>	Build a Bagging ensemble of estimators from the training set (X, y).
<code>get_n_hyperboxes()</code>	Get total number of hyperboxes in all base learners.
<code>get_params([deep])</code>	Get parameters for this estimator.
<code>predict(X)</code>	Predict class for X.
<code>predict_proba(X)</code>	Predict class probabilities for X.
<code>predict_with_membership(X)</code>	Predict class memberships for X.
<code>score(X, y[, sample_weight])</code>	Return the mean accuracy on the given test data and labels.
<code>set_params(**params)</code>	Set the parameters of this estimator.
<code>simple_pruning_base_estimators(X_val, y_val)</code>	Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox.

### **fit(X, y)**

Build a Bagging ensemble of estimators from the training set (X, y).

#### **Parameters**

**X**

[array-like of shape (n\_samples, n\_features)] The training input samples.

**y**

[array-like of shape (n\_samples,)] The real class labels

#### **Returns**

**self**

[object] Fitted estimator.

### **predict(X)**

Predict class for X.

The predicted class of an input sample is computed as the class with the highest mean predicted probability using voting.

#### **Parameters**

**X**

[array-like of shape (n\_samples, n\_features)] The testing input samples.

#### **Returns**

**y**

[ndarray of shape (n\_samples,)] The predicted classes.

### **predict\_proba(X)**

Predict class probabilities for X.

The predicted class probabilities of an input sample are computed as the mean predicted class probabilities of the hyperbox-based learners in the ensemble model. The class probability of a single hyperbox-based learner is the fraction of the membership value of the representative hyperbox of that class and the sum of all membership values of all representative hyperboxes of all classes.

#### **Parameters**

**X**

[array-like of shape (n\_samples, n\_features)] The input samples for prediction.

**Returns****all\_probas**

[ndarray of shape (n\_samples, n\_classes)] The class probabilities of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

**predict\_with\_membership(X)**

Predict class memberships for X.

The predicted class memberships of an input sample are computed as the mean predicted class memberships of the hyperbox-based learners in the ensemble model. The class membership of a single hyperbox-based learner is the membership from the input X to the representative hyperbox of that class to join the prediction procedure.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The input samples for prediction.

**Returns****mem\_vals**

[ndarray of shape (n\_samples, n\_classes)] The class memberships of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

## 2.5.5 ensemble\_learner.model\_comb\_bagging

A bagging of many base hyperbox-based models trained on the full set of features and a subset of samples. After formulation of base learners models, their hyperboxes are combined into a single model. The predicted class is computed based on the final single model.

```
class hbbrain.numerical_data.ensemble_learner.model_comb_bagging.ModelCombinationBagging(base_estimator=None,
                                             model_level_estimator=None,
                                             n_estimators=10,
                                             max_samples=0.5,
                                             bootstrap=False,
                                             class_balanced=False,
                                             n_jobs=1,
                                             random_state=None)
```

Bases: `ClassifierMixin`, [\*BaseBagging\*](#)

A Bagging classifier of base hyperbox-based models trained on a full set of features and a subset of samples. Then, the hyperboxes from all base learners are combined to a single model.

A model-level combination Bagging classifier of hyperbox-based models is an ensemble meta-estimator that fits base hyperbox-based classifiers each on random subsets of the original samples and then aggregate their hyperboxes into a single model and use this model for prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a single estimator, by introducing randomization into its construction procedure and then making an ensemble out of it. This algorithm encompasses several works from the literature. When random subsets of the dataset are drawn as random subsets of the samples, then this algorithm is known as Pasting [1]. If samples are drawn with replacement, then the method is known as Bagging [2]. See [3] for more detailed information regarding the combination of hyperboxes from all base hyperbox-based models.

**Parameters**

**base\_estimator**

[object, default=None] The base estimator to fit on random subsets of the dataset. If None, then the base estimator is a [OnlineGFMM](#).

**model\_level\_estimator**

[object, default=None] The estimator is used to aggregate all resulting hyperboxes from all base learners into a single model. If None, then the base estimator is a [AccelAgglomerativeLearningGFMM](#).

**n\_estimators**

[int, default=10] The number of base estimators in the ensemble.

**max\_samples**

[int or float, default=0.5] The number of samples to draw from X to train each base estimator (with no replacement by default, see *bootstrap* for more details). - If int, then draw *max\_samples* samples. - If float, then draw *max\_samples* \* *X.shape[0]* samples.

**bootstrap**

[bool, default=False] Whether samples are drawn with replacement. If False, sampling without replacement is performed.

**class\_balanced**

[bool, default=False] Whether samples are drawn without replacement to build a final subset with the equal number of samples among classes.

**n\_jobs**

[int, default=1] The number of jobs to run in parallel for both *fit()* and *predict()*. None means 1 unless in a *joblib.parallel\_backend* context. -1 means using all processors.

**random\_state**

[int, RandomState instance or None, default=None] Controls the random resampling of the original dataset (sample wise and feature wise). If the base estimator accepts a *random\_state* attribute, a different seed is generated for each instance in the ensemble. Pass an int for reproducible output across multiple function calls.

## References

[1], [2], [3]

## Examples

```
>>> from hbbrain.numerical_data.incremental_learner.iol_gfmm import_
↳ ImprovedOnlineGFMM
>>> from hbbrain.numerical_data.ensemble_learner.model_comb_bagging import_
↳ ModelCombinationBagging
>>> from hbbrain.numerical_data.batch_learner.accel_agglo_gfmm import_
↳ AccelAgglomerativeLearningGFMM
>>> from sklearn.datasets import make_classification
>>> X, y = make_classification(n_samples=100, n_features=4,
...                           n_informative=2, n_redundant=0,
...                           random_state=0, shuffle=False)
>>> from sklearn.preprocessing import MinMaxScaler
>>> scaler = MinMaxScaler()
>>> scaler.fit(X)
MinMaxScaler()
```

(continues on next page)

(continued from previous page)

```

>>> X = scaler.transform(X)
>>> clf = ModelCombinationBagging(base_estimator=ImprovedOnlineGFMM(0.1),
...                               model_level_
-> estimator=AccelAgglomerativeLearningGFMM(0.1),
...                               n_estimators=10, random_state=0).fit(X, y)
>>> clf.predict([[1, 0.6, 0.5, 0.2]])
array([1])

```

### Attributes

- base\_estimator\_**  
[estimator] The base estimator from which the ensemble is grown.
- model\_level\_estimator\_**  
[estimator] The final estimator is the combination of hyperboxes from all base learners.
- estimators\_**  
[list of estimators] The collection of fitted base estimators.
- estimators\_samples\_**  
[list of arrays] The subset of drawn samples for each base estimator.
- classes\_**  
[ndarray of shape (n\_classes,)] The classes labels.
- n\_classes\_**  
[int or list] The number of classes.

### Methods

<i>fit</i> (X, y[, is_pruning_base_learners, X_val, ...])	Build a Bagging ensemble of estimators from the training set (X, y).
<i>get_n_hyperboxes</i> ()	Get total number of hyperboxes in all base learners.
<i>get_n_hyperboxes_comb_model</i> ()	Get number of hyperboxes in the final combined model from all hyperboxes of base learners
<i>get_params</i> ([deep])	Get parameters for this estimator.
<i>predict</i> (X[, type_boundary_handling])	Predict class for X.
<i>predict_proba</i> (X)	Predict class probabilities of the input samples X.
<i>predict_proba_all_base_learners</i> (X)	Predict mean class probabilities for X from all base learners.
<i>predict_voting</i> (X)	Predict class for X.
<i>predict_with_membership</i> (X)	Predict class membership values of the input samples X.
<i>predict_with_membership_all_base_learners</i> (X)	Predict mean class memberships for X from all base learners.
<i>score</i> (X, y[, sample_weight])	Return the mean accuracy on the given test data and labels.
<i>set_params</i> (**params)	Set the parameters of this estimator.
<i>simple_pruning</i> (X_val, y_val[, ...])	Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox.
<i>simple_pruning_base_estimators</i> (X_val, y_val)	Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox.

**fit**(*X*, *y*, *is\_pruning\_base\_learners*=False, *X\_val*=None, *y\_val*=None, *acc\_threshold*=0.5, *keep\_empty\_boxes*=False)

Build a Bagging ensemble of estimators from the training set (*X*, *y*).

#### Parameters

**X**

[array-like of shape (n\_samples, n\_features)] The training input samples.

**y**

[array-like of shape (n\_samples,)] The real class labels

**is\_pruning\_base\_learners**

[boolean, optional, default=False] Whether the pruning procedure can be applied for base learners or not

**X\_val**

[array-like of shape (n\_samples, n\_features)] The data matrix contains validation patterns.

**y\_val**

[ndarray of shape (n\_samples,)] A vector contains the true class label corresponding to each validation pattern.

**acc\_threshold**

[float, optional, default=0.5] The minimum accuracy for each hyperbox to be kept unchanged.

**keep\_empty\_boxes**

[boolean, optional, default=False] Whether to keep the hyperboxes which do not join the prediction process on the validation set. If True, keep them, else the decision for keeping or removing based on the classification accuracy on the validation dataset

#### Returns

**self**

[object] Fitted estimator.

**get\_n\_hyperboxes\_comb\_model()**

Get number of hyperboxes in the final combined model from all hyperboxes of base learners

#### Returns

**int**

Total number of hyperboxes in the final combined models from all resulting hyperboxes of all base learners.

**predict**(*X*, *type\_boundary\_handling*=-1)

Predict class for *X*.

The predicted class of an input sample is computed as the class with the highest mean predicted probability using voting.

#### Parameters

**X**

[array-like of shape (n\_samples, n\_features)] The testing input samples.

**type\_boundary\_handling**

[int, optional, default=-1] The way of handling many winner hyperboxes. This parameter is only used in the case of *model\_level\_estimator* being improved online learning algorithm or agglomerative learning algorithms.

#### Returns

**y\_pred**  
[ndarray of shape (n\_samples,)] The predicted classes.

### **predict\_proba(X)**

Predict class probabilities of the input samples X.

The predicted class probability is the fraction of the membership value of the representative hyperbox of that class and the sum of all membership values of all representative hyperboxes of all classes in the prediction procedure using the final combined model.

#### **Parameters**

**X**  
[array-like of shape (n\_samples, n\_features)] The input samples.

#### **Returns**

**proba**  
[ndarray of shape (n\_samples, n\_classes)] The class probabilities of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

### **predict\_proba\_all\_base\_learners(X)**

Predict mean class probabilities for X from all base learners.

The predicted class probabilities of an input sample are computed as the mean predicted class probabilities of all hyperbox-based learners in the ensemble model. The class probability of a single hyperbox-based learner is the fraction of the membership value of the representative hyperbox of that class and the sum of all membership values of all representative hyperboxes of all classes.

#### **Parameters**

**X**  
[array-like of shape (n\_samples, n\_features)] The input samples for prediction.

#### **Returns**

**all\_probabilities**  
[ndarray of shape (n\_samples, n\_classes)] The class probabilities of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

### **predict\_voting(X)**

Predict class for X.

The predicted class of an input sample is computed as the class with the highest mean predicted probability using voting from all base learners.

#### **Parameters**

**X**  
[array-like of shape (n\_samples, n\_features)] The testing input samples.

#### **Returns**

**y**  
[ndarray of shape (n\_samples,)] The predicted classes.

### **predict\_with\_membership(X)**

Predict class membership values of the input samples X.

The predicted class membership value is the membership value of the representative hyperbox of that class in the prediction procedure using the final combined model.

#### **Parameters**

**X**

[array-like of shape (n\_samples, n\_features)] The input samples.

**Returns****mem\_vals**

[ndarray of shape (n\_samples, n\_classes)] The class membership values of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

**predict\_with\_membership\_all\_base\_learners(X)**

Predict mean class memberships for X from all base learners.

The predicted class memberships of an input sample are computed as the mean predicted class memberships of the hyperbox-based learners in the ensemble model. The class membership of a single hyperbox-based learner is the membership from the input X to the representative hyperbox of that class to join the prediction procedure.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The input samples for prediction.

**Returns****mem\_vals**

[ndarray of shape (n\_samples, n\_classes)] The class memberships of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

**simple\_pruning(X\_val, y\_val, acc\_threshold=0.5, keep\_empty\_boxes=False)**

Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox. This operation is applied for the final combined model.

**Parameters****X\_val**

[array-like of shape (n\_samples, n\_features)] The data matrix contains validation patterns.

**y\_val**

[ndarray of shape (n\_samples,)] A vector contains the true class label corresponding to each validation pattern.

**acc\_threshold**

[float, optional, default=0.5] The minimum accuracy for each hyperbox to be kept unchanged.

**keep\_empty\_boxes**

[boolean, optional, default=False] Whether to keep the hyperboxes which do not join the prediction process on the validation set. If True, keep them, else the decision for keeping or removing based on the classification accuracy on the validation dataset

**Returns****self**

A final hyperbox-based model is pruned of low-quality hyperboxes.

## 2.5.6 ensemble\_learner.model\_comb\_cross\_val\_bagging

A bagging of many base hyperbox-based models trained on the full set of features and a subset of samples. Each base learner is trained by k-fold cross validation and random search based hyper-parameter tuning. After formulation of base learners models, their hyperboxes are combined into a single model. The predicted class is computed based on the final single model.

```
class hbbrain.numerical_data.ensemble_learner.model_comb_cross_val_bagging.ModelCombinationCrossValBagging
```

Bases: `ClassifierMixin`, [\*BaseCrossValBagging\*](#)

A Bagging classifier of base hyperbox-based models trained on a full set of features and a subset of samples. Each base learner is trained by k-fold cross validation and random search based hyper-parameter tuning. Then, the hyperboxes from all base learners are combined to a single model.

A model-level combination cross validation Bagging classifier of hyperbox-based models is an ensemble meta-estimator that fits base hyperbox-based classifiers each on random subsets of the original samples using k-fold cross validation and random search of hyper-paramters, and then aggregate their hyperboxes into a single model and use this model for prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a single estimator, by introducing randomization into its construction procedure and then making an ensemble out of it. This algorithm encompasses several works from the literature. When random subsets of the dataset are drawn as random subsets of the samples, then this algorithm is known as Pasting [1]. If samples are drawn with replacement, then the method is known as Bagging [2]. See [3] for more detailed information regarding the combination of hyperboxes from all base hyperbox-based models.

### Parameters

#### **base\_estimator**

[object, default=None] The base estimator to fit on random subsets of the dataset. If None, then the base estimator is a [\*OnlineGFMM\*](#).

#### **base\_estimator\_params**

[dict or list of dicts, default={}] Dictionary with parameters names (str) as keys and distributions or lists of parameters to try. If a list is given, it is sampled uniformly. If a list of dicts is given, first a dict is sampled uniformly, and then a parameter is sampled using that dict as above.

#### **model\_level\_estimator**

[object, default=None] The estimator is used to aggregate all resulting hyperboxes from all base learners into a single model. If None, then the base estimator is a [\*AccelAgglomerativeLearningGFMM\*](#).

#### **n\_estimators**

[int, default=10] The number of base estimators in the ensemble.



**max\_samples**

[int or float, default=0.5] The number of samples to draw from X to train each base estimator (with no replacement by default, see *bootstrap* for more details). - If int, then draw *max\_samples* samples. - If float, then draw *max\_samples* \* *X.shape[0]* samples.

**bootstrap**

[bool, default=False] Whether samples are drawn with replacement. If False, sampling without replacement is performed.

**class\_balanced**

[bool, default=False] Whether samples are drawn without replacement to build a final subset with the equal number of samples among classes.

**n\_iter**

[int, default=10] Number of parameter settings that are sampled. *n\_iter* trades off runtime vs quality of the solution.

**scoring**

[str or callable default='accuracy'] Strategy to evaluate the performance of the cross-validated model on the test set. If *scoring* represents a single score, one can use: - a single string (see [The scoring parameter: defining model evaluation rules in sklearn](#)). - a callable (see [Defining your scoring strategy from metric functions](#)) that returns a single value.

**k\_fold**

[int, default=5] Determines the cross-validation splitting strategy. Possible inputs for *cv* are: - None, to use the default 5-fold cross validation, - integer, to specify the number of folds in a (*Stratified*)*KFold*, For integer/None inputs, if the estimator is a classifier and *y* is either binary or multiclass, Stratified K-Fold is used.

**n\_jobs**

[int, default=1] The number of jobs to run in parallel for both *fit()* and *predict()*. None means 1 unless in a *joblib.parallel\_backend* context. -1 means using all processors.

**random\_state**

[int, RandomState instance or None, default=None] Controls the random resampling of the original dataset (sample wise and feature wise). If the base estimator accepts a *random\_state* attribute, a different seed is generated for each instance in the ensemble. Pass an int for reproducible output across multiple function calls.

## References

[1], [2], [3]

## Examples

```
>>> from hbbrain.numerical_data.incremental_learner.onln_gfmm import OnlineGFMM
>>> from hbbrain.numerical_data.ensemble_learner.model_comb_cross_val_bagging_
↳ import ModelCombinationCrossValBagging
>>> from hbbrain.numerical_data.batch_learner.accel_agglo_gfmm import_
↳ AccelAgglomerativeLearningGFMM
>>> from sklearn.datasets import make_classification
>>> X, y = make_classification(n_samples=100, n_features=4,
...                           n_informative=2, n_redundant=0,
...                           random_state=0, shuffle=False)
>>> from sklearn.preprocessing import MinMaxScaler
```

(continues on next page)

(continued from previous page)

```
>>> scaler = MinMaxScaler()
>>> scaler.fit(X)
MinMaxScaler()
>>> X = scaler.transform(X)
>>> clf = ModelCombinationCrossValBagging(base_estimator=OnlineGFMM(0.1),
...                                     base_estimator_params = {'theta': np.arange(0.05, 1.
→ 0.1, 0.05), 'theta_min':[1], 'gamma':[0.5, 1, 2, 4, 8, 16]},
...                                     model_level_
→ estimator=AccelAgglomerativeLearningGFMM(0.1),
...                                     n_estimators=10, random_state=0).fit(X, y)
>>> clf.predict([[1, 0.6, 0.5, 0.2]])
array([1])
```

### Attributes

#### **base\_estimator\_**

[estimator] The base estimator from which the ensemble is grown.

#### **model\_level\_estimator\_**

[estimator] The final estimator is the combination of hyperboxes from all base learners.

#### **estimators\_**

[list of estimators] The collection of fitted base estimators.

#### **estimators\_samples\_**

[list of arrays] The subset of drawn samples for each base estimator.

#### **classes\_**

[ndarray of shape (n\_classes,)] The classes labels.

#### **n\_classes\_**

[int or list] The number of classes.

## Methods

<code>fit(X, y[, is_pruning_base_learners, X_val, ...])</code>	Build a Bagging ensemble of estimators from the training set (X, y).
<code>get_n_hyperboxes()</code>	Get total number of hyperboxes in all base learners.
<code>get_n_hyperboxes_comb_model()</code>	Get number of hyperboxes in the final combined model from all hyperboxes of base learners
<code>get_params([deep])</code>	Get parameters for this estimator.
<code>predict(X[, type_boundary_handling])</code>	Predict class for X.
<code>predict_proba(X)</code>	Predict class probabilities of the input samples X.
<code>predict_proba_all_base_learners(X)</code>	Predict mean class probabilities for X from all base learners.
<code>predict_voting(X)</code>	Predict class for X.
<code>predict_with_membership(X)</code>	Predict class membership values of the input samples X.
<code>predict_with_membership_all_base_learners(X)</code>	Predict mean class memberships for X from all base learners.
<code>score(X, y[, sample_weight])</code>	Return the mean accuracy on the given test data and labels.
<code>set_params(**params)</code>	Set the parameters of this estimator.
<code>simple_pruning(X_val, y_val[, ...])</code>	Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox.
<code>simple_pruning_base_estimators(X_val, y_val)</code>	Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox.

**fit**(X, y, is\_pruning\_base\_learners=False, X\_val=None, y\_val=None, acc\_threshold=0.5, keep\_empty\_boxes=False)

Build a Bagging ensemble of estimators from the training set (X, y).

### Parameters

#### X

[array-like of shape (n\_samples, n\_features)] The training input samples.

#### y

[array-like of shape (n\_samples,)] The real class labels

#### is\_pruning\_base\_learners

[boolean, optional, default=False] Whether the pruning procedure can be applied for base learners or not

#### X\_val

[array-like of shape (n\_samples, n\_features)] The data matrix contains validation patterns.

#### y\_val

[ndarray of shape (n\_samples,)] A vector contains the true class label corresponding to each validation pattern.

#### acc\_threshold

[float, optional, default=0.5] The minimum accuracy for each hyperbox to be kept unchanged.

#### keep\_empty\_boxes

[boolean, optional, default=False] Whether to keep the hyperboxes which do not join the prediction process on the validation set. If True, keep them, else the decision for keeping or removing based on the classification accuracy on the validation dataset

**Returns****self**

[object] Fitted estimator.

**get\_n\_hyperboxes\_comb\_model()**

Get number of hyperboxes in the final combined model from all hyperboxes of base learners

**Returns****int**

Total number of hyperboxes in the final combined model from all resulting hyperboxes of all base learners.

**predict(X, type\_boundary\_handling=-1)**

Predict class for X.

The predicted class of an input sample is computed as the class with the highest mean predicted probability using voting.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The testing input samples.

**type\_boundary\_handling**[int, optional, default=-1] The way of handling many winner hyperboxes. This parameter is only used in the case of *model\_level\_estimator* being improved online learning algorithm or agglomerative learning algorithms.**Returns****y\_pred**

[ndarray of shape (n\_samples,)] The predicted classes.

**predict\_proba(X)**

Predict class probabilities of the input samples X.

The predicted class probability is the fraction of the membership value of the representative hyperbox of that class and the sum of all membership values of all representative hyperboxes of all classes in the prediction procedure using the final combined model.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The input samples.

**Returns****proba**

[ndarray of shape (n\_samples, n\_classes)] The class probabilities of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

**predict\_proba\_all\_base\_learners(X)**

Predict mean class probabilities for X from all base learners.

The predicted class probabilities of an input sample are computed as the mean predicted class probabilities of all hyperbox-based learners in the ensemble model. The class probability of a single hyperbox-based learner is the fraction of the membership value of the representative hyperbox of that class and the sum of all membership values of all representative hyperboxes of all classes.

**Parameters**

**X**

[array-like of shape (n\_samples, n\_features)] The input samples for prediction.

**Returns****all\_probas**

[ndarray of shape (n\_samples, n\_classes)] The class probabilities of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

**predict\_voting(X)**

Predict class for X.

The predicted class of an input sample is computed as the class with the highest mean predicted probability using voting from all base learners.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The testing input samples.

**Returns****y**

[ndarray of shape (n\_samples,)] The predicted classes.

**predict\_with\_membership(X)**

Predict class membership values of the input samples X.

The predicted class membership value is the membership value of the representative hyperbox of that class in the prediction procedure using the final combined model.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The input samples.

**Returns****mem\_vals**

[ndarray of shape (n\_samples, n\_classes)] The class membership values of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

**predict\_with\_membership\_all\_base\_learners(X)**

Predict mean class memberships for X from all base learners.

The predicted class memberships of an input sample are computed as the mean predicted class memberships of the hyperbox-based learners in the ensemble model. The class membership of a single hyperbox-based learner is the membership from the input X to the representative hyperbox of that class to join the prediction procedure.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The input samples for prediction.

**Returns****mem\_vals**

[ndarray of shape (n\_samples, n\_classes)] The class memberships of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

**simple\_pruning**(*X\_val*, *y\_val*, *acc\_threshold*=0.5, *keep\_empty\_boxes*=False)

Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox. This operation is applied for the final combined model.

**Parameters**

**X\_val**

[array-like of shape (n\_samples, n\_features)] The data matrix contains validation patterns.

**y\_val**

[ndarray of shape (n\_samples,)] A vector contains the true class label corresponding to each validation pattern.

**acc\_threshold**

[float, optional, default=0.5] The minimum accuracy for each hyperbox to be kept unchanged.

**keep\_empty\_boxes**

[boolean, optional, default=False] Whether to keep the hyperboxes which do not join the prediction process on the validation set. If True, keep them, else the decision for keeping or removing based on the classification accuracy on the validation dataset

**Returns**

**self**

A final hyperbox-based model is pruned of low-quality hyperboxes.

## 2.5.7 ensemble\_learner.random\_hyperboxes

Functions and classes for the random hyperboxes model.

**class** hbbrain.numerical\_data.ensemble\_learner.random\_hyperboxes.**RandomHyperboxesClassifier**(*base\_estimator*=*None*, *n\_estimators*=10, *max\_samples*=0.8, *max\_features*='sqrt', *class\_balanced*=False, *feature\_balanced*=False, *n\_jobs*=1, *random\_state*=None)

Bases: [ClassifierMixin](#), [BaseEnsemble](#)

A Random Hyperboxes classifier of base hyperbox-based models trained on a subset of features and a subset of samples.

A Random Hyperboxes classifier of hyperbox-based models is an ensemble meta-estimator that fits base hyperbox-based classifiers each on random subsets of both original samples and features, then aggregate their individual predictions by voting to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a single estimator, by introducing randomization into its construction procedures and then making an ensemble out of it. Subsets of features and samples of the random hyperboxes are built by random subsampling without replacement. See [1] for more detailed information regarding the random hyperboxes classifier.

**Parameters**

**base\_estimator**

[object, default=None] The base estimator to fit on random subsets of the dataset. If None, then the base estimator is a [OnlineGFMM](#).

**n\_estimators**

[int, default=10] The number of base estimators in the ensemble.

**max\_samples**

[int or float, default=0.5] The number of samples to draw from X to train each base estimator (with no replacement by default, see *bootstrap* for more details).

- If int, then draw *max\_samples* samples.
- If float, then draw *max\_samples* \* *X.shape[0]* samples.

**max\_features**

[{"sqrt", "log2"}, int or float, default="sqrt"] The maximum number of features to consider when building training data for base learners:

- If int, then consider *max\_features* features.
- If float, then *max\_features* is a fraction and *round(max\_features \* n\_features)* features are considered.
- If "sqrt", then *max\_features=sqrt(n\_features)*.
- If "log2", then *max\_features=log2(n\_features)*.

**class\_balanced**

[bool, default=False] Whether samples are drawn without replacement to build a final subset with the equal number of samples among classes.

**feature\_balanced: bool, default = False**

Whether number of features of training sets for all base learners are equal to each other or not.

**n\_jobs**

[int, default=1] The number of jobs to run in parallel for both *fit()* and *predict()*. None means 1 unless in a *joblib.parallel\_backend* context. -1 means using all processors.

**random\_state**

[int, RandomState instance or None, default=None] Controls the random resampling of the original dataset (sample wise and feature wise). If the base estimator accepts a *random\_state* attribute, a different seed is generated for each instance in the ensemble. Pass an int for reproducible output across multiple function calls.

## References

[1]

## Examples

```
>>> from hbbrain.numerical_data.incremental_learner.iol_gfmm import_
↳ ImprovedOnlineGFMM
>>> from hbbrain.numerical_data.ensemble_learner.random_hyperboxes import_
↳ RandomHyperboxesClassifier
>>> from sklearn.datasets import make_classification
>>> X, y = make_classification(n_samples=100, n_features=4,
...                           n_informative=2, n_redundant=0,
...                           random_state=0, shuffle=False)
>>> from sklearn.preprocessing import MinMaxScaler
```

(continues on next page)

(continued from previous page)

```

>>> scaler = MinMaxScaler()
>>> scaler.fit(X)
MinMaxScaler()
>>> X = scaler.transform(X)
>>> clf = RandomHyperboxesClassifier(base_estimator=ImprovedOnlineGFMM(0.1),
...                                n_estimators=10, random_state=0).fit(X, y)
>>> clf.predict([[1, 0.6, 0.5, 0.2]])
array([1])

```

**Attributes****base\_estimator\_**

[estimator] The base estimator from which the ensemble is grown.

**n\_features\_**

[int] Number of features seen during fit.

**estimators\_**

[list of estimators] The collection of fitted base estimators.

**estimators\_samples\_**

[list of arrays] The subset of drawn samples for each base estimator.

**estimators\_features\_**

[list of arrays] The subset of indices of the drawn features for each base estimator. Each subset is defined by an array of the indices selected.

**classes\_**

[ndarray of shape (n\_classes,)] The classes labels.

**n\_classes\_**

[int or list] The number of classes.

**Methods**

<i>fit</i> (X, y)	Build a random hyperbox model from the training set (X, y).
<i>get_n_hyperboxes</i> ()	Get total number of hyperboxes in all base learners.
<i>get_params</i> ([deep])	Get parameters for this estimator.
<i>predict</i> (X)	Predict class for X.
<i>predict_proba</i> (X)	Predict class probabilities for X.
<i>predict_with_membership</i> (X)	Predict class memberships for X.
<i>score</i> (X, y[, sample_weight])	Return the mean accuracy on the given test data and labels.
<i>set_params</i> (**params)	Set the parameters of this estimator.
<i>simple_pruning_base_estimators</i> (X_val, y_val)	Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox.

**property estimators\_samples\_**

The subset of drawn samples for each base estimator. Returns a dynamically generated list of indices identifying the samples used for fitting each member of the ensemble, i.e., the in-bag samples.



---

**Note:** The list is re-created at each call to the property in order to reduce the object memory footprint by not storing the sampling data. Thus fetching the property may be slower than expected.

---

**fit**(X, y)

Build a random hyperbox model from the training set (X, y).

**Parameters**

**X**

[array-like of shape (n\_samples, n\_features)] The training input samples.

**y**

[array-like of shape (n\_samples,)] The class labels.

**Returns**

**self**

[object] Fitted estimator.

**get\_n\_hyperboxes**()

Get total number of hyperboxes in all base learners.

**Returns**

**n\_hyperboxes**

[int] Total number of hyperboxes in all base learners.

**predict**(X)

Predict class for X.

The predicted class of an input sample is computed as the class with the highest mean predicted probability using voting.

**Parameters**

**X**

[array-like of shape (n\_samples, n\_features)] The testing input samples.

**Returns**

**y**

[ndarray of shape (n\_samples,)] The predicted classes.

**predict\_proba**(X)

Predict class probabilities for X.

The predicted class probabilities of an input sample are computed as the mean predicted class probabilities of the hyperbox-based learners in the ensemble model. The class probability of a single hyperbox-based learner is the fraction of the membership value of the representative hyperbox of that class and the sum of all membership values of all representative hyperboxes of all classes.

**Parameters**

**X**

[array-like of shape (n\_samples, n\_features)] The input samples for prediction.

**Returns**

**all\_probas**

[ndarray of shape (n\_samples, n\_classes)] The class probabilities of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

**predict\_with\_membership(X)**

Predict class memberships for X.

The predicted class memberships of an input sample are computed as the mean predicted class memberships of the hyperbox-based learners in the ensemble model. The class membership of a single hyperbox-based learner is the membership from the input X to the representative hyperbox of that class to join the prediction procedure.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The input samples for prediction.

**Returns****mem\_vals**

[ndarray of shape (n\_samples, n\_classes)] The class memberships of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

**simple\_pruning\_base\_estimators(X\_val, y\_val, acc\_threshold=0.5, keep\_empty\_boxes=False)**

Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox. This operation is applied for all base estimators.

**Parameters****X\_val**

[array-like of shape (n\_samples, n\_features)] The data matrix contains validation patterns.

**y\_val**

[ndarray of shape (n\_samples,)] A vector contains the true class label corresponding to each validation pattern.

**acc\_threshold**

[float, optional, default=0.5] The minimum accuracy for each hyperbox to be kept unchanged.

**keep\_empty\_boxes**

[boolean, optional, default=False] Whether to keep the hyperboxes which do not join the prediction process on the validation set. If True, keep them, else the decision for keeping or removing based on the classification accuracy on the validation dataset

**Returns****self**

A random hyperboxes model with base estimators pruned.

## 2.5.8 ensemble\_learner.cross\_val\_random\_hyperboxes

Functions and classes for the cross-validation random hyperboxes model.

---

```
class hbbrain.numerical_data.ensemble_learner.cross_val_random_hyperboxes.CrossValRandomHyperboxesClass:
```

Bases: `ClassifierMixin`, [\*BaseEnsemble\*](#)

A Corss-validation Random Hyperboxes classifier of base hyperbox-based models trained on a subset of features and a subset of samples together with random search-based hyper-parameter tuning and k-fold cross-validation.

A Random Hyperboxes classifier of hyperbox-based models is an ensemble meta-estimator that fits base hyperbox-based classifiers each on random subsets of both original samples and features using k-fold cross-validation and hyper-parameter tuning based on random search. Then, base learners are aggregated with their individual predictions by voting to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a single estimator, by introducing randomization into its construction procedures and then making an ensemble out of it. Subsets of features and samples of the random hyperboxes are built by random subsampling without replacement. See [1] for more detailed information regarding the random hyperboxes classifier.

### Parameters

#### **base\_estimator**

[object, default=None] The base estimator to fit on random subsets of the dataset. If None, then the base estimator is a [\*OnlineGFMM\*](#).

#### **base\_estimator\_params**

[dict or list of dicts, default={}] Dictionary with parameters names (str) as keys and distributions or lists of parameters to try. If a list is given, it is sampled uniformly. If a list of dicts is given, first a dict is sampled uniformly, and then a parameter is sampled using that dict as above.

#### **n\_estimators**

[int, default=10] The number of base estimators in the ensemble.

#### **max\_samples**

[int or float, default=0.5] The number of samples to draw from X to train each base estimator (with no replacement by default, see *bootstrap* for more details).

- If int, then draw *max\_samples* samples.
- If float, then draw *max\_samples* \* *X.shape[0]* samples.

#### **max\_features**

[{"sqrt", "log2"}, int or float, default="sqrt"] The maximum number of features to consider when building training data for base learners:

- If int, then consider *max\_features* features.
- If float, then *max\_features* is a fraction and *round(max\_features \* n\_features)* features are considered.

- If “sqrt”, then  $max\_features=sqrt(n\_features)$ .
- If “log2”, then  $max\_features=log2(n\_features)$ .

**class\_balanced**

[bool, default=False] Whether samples are drawn without replacement to build a final subset with the equal number of samples among classes.

**feature\_balanced: bool, default = False**

Whether number of features of training sets for all base learners are equal to each other or not.

**n\_iter**

[int, default=10] Number of parameter settings that are sampled. `n_iter` trades off runtime vs quality of the solution.

**scoring**

[str or callable default='accuracy'] Strategy to evaluate the performance of the cross-validated model on the test set. If *scoring* represents a single score, one can use: - a single string (see [The scoring parameter: defining model evaluation rules in sklearn](#)). - a callable (see [Defining your scoring strategy from metric functions](#)) that returns a single value.

**k\_fold**

[int, default=5] Determines the cross-validation splitting strategy. Possible inputs for *cv* are: - None, to use the default 5-fold cross validation, - integer, to specify the number of folds in a (*Stratified*)*KFold*, For integer/None inputs, if the estimator is a classifier and *y* is either binary or multiclass, Stratified K-Fold is used.

**n\_jobs**

[int, default=1] The number of jobs to run in parallel for both [fit\(\)](#) and [predict\(\)](#). None means 1 unless in a `joblib.parallel_backend` context. -1 means using all processors.

**random\_state**

[int, RandomState instance or None, default=None] Controls the random resampling of the original dataset (sample wise and feature wise). If the base estimator accepts a *random\_state* attribute, a different seed is generated for each instance in the ensemble. Pass an int for reproducible output across multiple function calls.

## References

[1]

## Examples

```
>>> from hbbrain.numerical_data.incremental_learner.iol_gfmm import_
↳ ImprovedOnlineGFMM
>>> from hbbrain.numerical_data.ensemble_learner.cross_val_random_hyperboxes import_
↳ CrossValRandomHyperboxesClassifier
>>> from sklearn.datasets import make_classification
>>> X, y = make_classification(n_samples=100, n_features=4,
...                           n_informative=2, n_redundant=0,
...                           random_state=0, shuffle=False)
>>> from sklearn.preprocessing import MinMaxScaler
>>> scaler = MinMaxScaler()
>>> scaler.fit(X)
```

(continues on next page)

(continued from previous page)

```

MinMaxScaler()
>>> X = scaler.transform(X)
>>> clf = CrossValRandomHyperboxesClassifier(base_estimator=ImprovedOnlineGFMM(0.1),
...                                           base_estimator_params={'theta': np.arange(0.05, 1.01, 0.
→05), 'gamma':[0.5, 1, 2, 4, 8, 16]}),
...                                           n_estimators=10, random_state=0).fit(X, y)
>>> clf.predict([[1, 0.6, 0.5, 0.2]])
array([1])

```

**Attributes**

- base\_estimator\_**  
[estimator] The base estimator from which the ensemble is grown.
- n\_features\_**  
[int] Number of features seen during fit.
- estimators\_**  
[list of estimators] The collection of fitted base estimators.
- estimators\_samples\_**  
[list of arrays] The subset of drawn samples for each base estimator.
- classes\_**  
[ndarray of shape (n\_classes,)] The classes labels.
- n\_classes\_**  
[int or list] The number of classes.

**Methods**

<i>fit</i> (X, y)	Build a random hyperbox model from the training set (X, y).
<i>get_n_hyperboxes</i> ()	Get total number of hyperboxes in all base learners.
<i>get_params</i> ([deep])	Get parameters for this estimator.
<i>predict</i> (X)	Predict class for X.
<i>predict_proba</i> (X)	Predict class probabilities for X.
<i>predict_with_membership</i> (X)	Predict class memberships for X.
<i>score</i> (X, y[, sample_weight])	Return the mean accuracy on the given test data and labels.
<i>set_params</i> (**params)	Set the parameters of this estimator.
<i>simple_pruning_base_estimators</i> (X_val, y_val)	Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox.

**property estimators\_samples\_**

The subset of drawn samples for each base estimator. Returns a dynamically generated list of indices identifying the samples used for fitting each member of the ensemble, i.e., the in-bag samples.

**Note:** The list is re-created at each call to the property in order to reduce the object memory footprint by not storing the sampling data. Thus fetching the property may be slower than expected.

**fit(X, y)**

Build a random hyperbox model from the training set (X, y).

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The training input samples.

**y**

[array-like of shape (n\_samples,)] The class labels.

**Returns****self**

[object] Fitted estimator.

**get\_n\_hyperboxes()**

Get total number of hyperboxes in all base learners.

**Returns****int**

Total number of hyperboxes in all base learners.

**predict(X)**

Predict class for X.

The predicted class of an input sample is computed as the class with the highest mean predicted probability using voting.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The testing input samples.

**Returns****y**

[ndarray of shape (n\_samples,)] The predicted classes.

**predict\_proba(X)**

Predict class probabilities for X.

The predicted class probabilities of an input sample are computed as the mean predicted class probabilities of the hyperbox-based learners in the ensemble model. The class probability of a single hyperbox-based learner is the fraction of the membership value of the representative hyperbox of that class and the sum of all membership values of all representative hyperboxes of all classes.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The input samples for prediction.

**Returns****all\_probab**

[ndarray of shape (n\_samples, n\_classes)] The class probabilities of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

**predict\_with\_membership(X)**

Predict class memberships for X.

The predicted class memberships of an input sample are computed as the mean predicted class memberships of the hyperbox-based learners in the ensemble model. The class membership of a single hyperbox-based

learner is the membership from the input X to the representative hyperbox of that class to join the prediction procedure.

#### Parameters

##### X

[array-like of shape (n\_samples, n\_features)] The input samples for prediction.

#### Returns

##### mem\_vals

[ndarray of shape (n\_samples, n\_classes)] The class memberships of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

**simple\_pruning\_base\_estimators**(X\_val, y\_val, acc\_threshold=0.5, keep\_empty\_boxes=False)

Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox. This operation is applied for all base estimators.

#### Parameters

##### X\_val

[array-like of shape (n\_samples, n\_features)] The data matrix contains validation patterns.

##### y\_val

[ndarray of shape (n\_samples,)] A vector contains the true class label corresponding to each validation pattern.

##### acc\_threshold

[float, optional, default=0.5] The minimum accuracy for each hyperbox to be kept unchanged.

##### keep\_empty\_boxes

[boolean, optional, default=False] Whether to keep the hyperboxes which do not join the prediction process on the validation set. If True, keep them, else the decision for keeping or removing based on the classification accuracy on the validation dataset

#### Returns

##### self

A random hyperboxes model with base estimators pruned.

## 2.6 incremental learners

### 2.6.1 incremental\_learner.onln\_gfmm

General fuzzy min-max neural network trained by the original incremental learning algorithm.

```
class hbbrain.numerical_data.incremental_learner.onln_gfmm.OnlineGFMM(theta=0.5, theta_min=1,
                                                                    gamma=1, alpha=0.9,
                                                                    is_draw=False, V=None,
                                                                    W=None, C=None)
```

Bases: *BaseGFMMClassifier*

General fuzzy min-max neural network model using the original incremental learning algorithm.

This class implements the original online learning algorithm to train the general fuzzy min-max neural network. The details of this algorithm can be found in [1].

---

**Note:** This implementation uses the accelerated mechanism presented in [2] to accelerate the improved online learning algorithm. Compared to the original online learning algorithm proposed in [1], this implementation uses the similarity measure between two hyperboxes by shortest gap distance presented in [3] for overlap test. In addition, we extend the number of hyperbox contraction cases from four in the original algorithm to eight cases aiming to cover more overlapping cases between two hyperboxes.

---

### Parameters

#### **theta**

[float, optional, default=0.5] Maximum hyperbox size for numerical features.

#### **theta\_min**

[float, optional, default=1] Minimum value of the maximum hyperbox size for continuous features so that the training loop is still performed. If the value of *theta\_min* is larger than the value of *theta*, it will be automatically assigned a value equal to *theta*.

#### **gamma**

[float or ndarray of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each continuous feature.

#### **alpha**

[float, optional, default=0.9] Multiplier factor to reduce the value of maximum hyperbox size after each training loop.

#### **is\_draw**

[boolean, optional, default=False] Whether the construction of hyperboxes can be progressively shown during the training process on a canvas window.

#### **V**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all minimal points for numerical features of all existing hyperboxes, in which each row is a minimal point of a hyperbox.

#### **W**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all maximal points for numerical features of all existing hyperboxes, in which each row is a maximal point of a hyperbox.

#### **C**

[array-like of shape (n\_hyperboxes,)] A vector stores all class labels corresponding to existing hyperboxes.



## References

[1], [2], [3]

## Examples

```
>>> from sklearn.datasets import load_iris
>>> from hbbrain.numerical_data.incremental_learner.onln_gfmm import OnlineGFMM
>>> X, y = load_iris(return_X_y=True)
>>> from sklearn.preprocessing import MinMaxScaler
>>> scaler = MinMaxScaler()
>>> scaler.fit(X)
MinMaxScaler()
>>> X = scaler.transform(X)
>>> clf = OnlineGFMM(theta=0.1).fit(X, y)
>>> clf.predict(X[[10, 50, 100]])
array([0, 1, 2])
```

## Attributes

### **is\_exist\_missing\_value**

[boolean] Is there any missing values in continuous features in the training data.

### **elapsed\_training\_time**

[float] Training time in seconds.

### **n\_passes**

[int] Number of training loops.

## Methods

<code>delay([delay_constant])</code>	Delay a time period to display hyperboxes
<code>draw_hyperbox_and_boundary([window_name, ...])</code>	Draw the existing hyperboxes and their decision boundaries among classes
<code>fit(X, y)</code>	Fit the model according to the given training data using the original incremental learning algorithm.
<code>get_n_hyperboxes()</code>	Get number of hyperboxes in the trained hyperbox-based model
<code>get_params([deep])</code>	Get parameters for this estimator.
<code>get_sample_explanation(xl, xu)</code>	Get useful information for explaining the reason behind the predicted result for the input pattern
<code>initialise_canvas_graph([n_dims, ...])</code>	Initialise a canvas to draw hyperboxes
<code>predict(X)</code>	Predict class labels for samples in <i>X</i> .
<code>predict_proba(X)</code>	Predict class probabilities of the input samples <i>X</i> .
<code>predict_with_membership(X)</code>	Predict class membership values of the input samples <i>X</i> .
<code>score(X, y[, sample_weight])</code>	Return the mean accuracy on the given test data and labels.
<code>set_params(**params)</code>	Set the parameters of this estimator.
<code>show_sample_explanation(xl, xu, ...[, ...])</code>	Show explanation for predicted results of an input pattern under the form of parallel coordinates or hyperboxes in 2D or 3D planes.
<code>simple_pruning(Xl_val, Xu_val, y_val[, ...])</code>	Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox

### `fit(X, y)`

Fit the model according to the given training data using the original incremental learning algorithm.

#### Parameters

##### **X**

[array-like of shape (n\_samples, n\_features)] Training vector, where *n\_samples* is the number of samples and *n\_features* is the number of features.

##### **y**

[array-like of shape (n\_samples,)] Target vector relative to *X*.

#### Returns

##### **self**

[object] Fitted general fuzzy min-max neural network.

### `get_sample_explanation(xl, xu)`

Get useful information for explaining the reason behind the predicted result for the input pattern

#### Parameters

##### **xl**

[ndarray of shape (n\_feature,)] Minimum point of the input pattern which needs to be explained.

##### **xu**

[ndarray of shape (n\_feature,)] Maximum point of the input pattern which needs to be explained.

#### Returns

**y\_pred**  
[int] The predicted class of the input pattern

**dict\_mem\_val\_classes**  
[dictionary] A dictionary stores all membership values for all classes. The key is class label and the value is the corresponding membership value.

**dict\_min\_point\_classes**  
[dictionary] A dictionary stores all minimal points of hyperboxes having the maximum membership value for each class. The key is the class label and the value is the minimal points of all hyperboxes corresponding to each class

**dict\_max\_point\_classes**  
[dictionary] A dictionary stores all maximal points of hyperboxes having the maximum membership value for each class. The key is the class label and the value is the maximal points of all hyperboxes corresponding to each class

**simple\_pruning**(*Xl\_val, Xu\_val, y\_val, acc\_threshold=0.5, keep\_empty\_boxes=False*)

Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox

#### Parameters

**Xl\_val**  
[array-like of shape (n\_samples, n\_features)] The data matrix contains lower bounds of validation patterns.

**Xu\_val**  
[array-like of shape (n\_samples, n\_features)] The data matrix contains upper bounds of validation patterns.

**y\_val**  
[ndarray of shape (n\_samples,)] A vector contains the true class label corresponding to each validation pattern.

**acc\_threshold**  
[float, optional, default=0.5] The minimum accuracy for each hyperbox to be kept unchanged.

**keep\_empty\_boxes**  
[boolean, optional, default=False] Whether to keep the hyperboxes which do not join the prediction process on the validation set. If True, keep them, else the decision for keeping or removing based on the classification accuracy on the validation dataset

#### Returns

**self**  
A hyperbox-based model with the low-qualified hyperboxes pruned.

## 2.6.2 incremental\_learner.iol\_gfmm

General fuzzy min-max neural network trained by the improved incremental learning algorithm.

```
class hbbrain.numerical_data.incremental_learner.iol_gfmm.ImprovedOnlineGFMM(theta=0.5,  
                                                                           gamma=1,  
                                                                           is_draw=False,  
                                                                           V=None,  
                                                                           W=None,  
                                                                           C=None,  
                                                                           N_samples=None)
```

Bases: *BaseGFMMClassifier*

General fuzzy min-max neural network classifier with an improved online learning algorithm.

This class implements an improved online learning algorithm to train a fuzzy min-max neural network classifier. This learning algorithm does not allow the occurrence of hyperbox overlapping regions when conducting the hyperbox expansion procedure. The details of this algorithm can be found in [1].

---

**Note:** This implementation uses the accelerated mechanism presented in [2] to accelerate the improved online learning algorithm.

---

### Parameters

**theta**

[float, optional, default=0.5] Maximum hyperbox size for numerical features.

**gamma**

[float or ndarray of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each continuous feature.

**is\_draw**

[boolean, optional, default=False] Whether the construction of hyperboxes can be progressively shown during the training process on a canvas window.

**V**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all minimal points for numerical features of all existing hyperboxes, in which each row is a minimal point of a hyperbox.

**W**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all maximal points for numerical features of all existing hyperboxes, in which each row is a maximal point of a hyperbox.

**C**

[array-like of shape (n\_hyperboxes,)] A vector stores all class labels corresponding to existing hyperboxes.

**N\_samples**

[array-like of shape (n\_hyperboxes,)] A vector stores the number of samples fully included in each existing hyperbox.

### References

[1], [2]

## Examples

```
>>> from sklearn.datasets import load_iris
>>> from hbbrain.numerical_data.incremental_learner.iol_gfmm import _
↳ ImprovedOnlineGFMM
>>> X, y = load_iris(return_X_y=True)
>>> from sklearn.preprocessing import MinMaxScaler
>>> scaler = MinMaxScaler()
>>> scaler.fit(X)
MinMaxScaler()
>>> X = scaler.transform(X)
>>> clf = ImprovedOnlineGFMM(theta=0.1).fit(X, y)
>>> clf.predict(X[[10, 50, 100]])
array([0, 1, 2])
```

## Attributes

### **is\_exist\_missing\_value**

[boolean] Is there any missing values in continuous features in the training data.

### **elapsed\_training\_time**

[float] Training time in seconds.

## Methods

<code>delay([delay_constant])</code>	Delay a time period to display hyperboxes
<code>draw_hyperbox_and_boundary([window_name, ...])</code>	Draw the existing hyperboxes and their decision boundaries among classes
<code><i>fit</i>(X, y)</code>	Fit the model according to the given training data using the improved incremental learning algorithm.
<code>get_n_hyperboxes()</code>	Get number of hyperboxes in the trained hyperbox-based model
<code>get_params([deep])</code>	Get parameters for this estimator.
<code><i>get_sample_explanation</i>(xl, xu[, ...])</code>	Get useful information for explaining the reason behind the predicted result for the input pattern
<code>initialise_canvas_graph([n_dims, ...])</code>	Initialise a canvas to draw hyperboxes
<code><i>predict</i>(X[, type_boundary_handling])</code>	Predict class labels for samples in X.
<code>predict_proba(X)</code>	Predict class probabilities of the input samples X.
<code>predict_with_membership(X)</code>	Predict class membership values of the input samples X.
<code>score(X, y[, sample_weight])</code>	Return the mean accuracy on the given test data and labels.
<code>set_params(**params)</code>	Set the parameters of this estimator.
<code>show_sample_explanation(xl, xu, ...[, ...])</code>	Show explanation for predicted results of an input pattern under the form of parallel coordinates or hyperboxes in 2D or 3D planes.
<code><i>simple_pruning</i>(Xl_val, Xu_val, y_val[, ...])</code>	Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox

### **fit(X, y)**

Fit the model according to the given training data using the improved incremental learning algorithm.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] Training vector, where *n\_samples* is the number of samples and *n\_features* is the number of features.

**y**

[array-like of shape (n\_samples,)] Target vector relative to X.

**Returns****self**

[object] Fitted general fuzzy min-max neural network.

**get\_sample\_explanation**(*xl, xu, type\_boundary\_handling=1*)

Get useful information for explaining the reason behind the predicted result for the input pattern

**Parameters****xl**

[ndarray of shape (n\_feature,)] Minimum point of the input pattern which needs to be explained.

**xu**

[ndarray of shape (n\_feature,)] Maximum point of the input pattern which needs to be explained.

**type\_boundary\_handling**

[int, optional, default=PROBABILITY\_MEASURE (aka 1)] The way of handling samples located on the boundary.

**Returns****y\_pred**

[int] The predicted class of the input pattern

**dict\_mem\_val\_classes**

[dictionary] A dictionary stores all membership values for all classes. The key is class label and the value is the corresponding membership value.

**dict\_min\_point\_classes**

[dictionary] A dictionary stores all minimal points of hyperboxes having the maximum membership value for each class. The key is the class label and the value is the minimal points of all hyperboxes corresponding to each class

**dict\_max\_point\_classes**

[dictionary] A dictionary stores all maximal points of hyperboxes having the maximum membership value for each class. The key is the class label and the value is the maximal points of all hyperboxes corresponding to each class

**predict**(*X, type\_boundary\_handling=1*)

Predict class labels for samples in X.

---

**Note:** In the case there are many winner hyperboxes representing different class labels but with the same membership value with respect to the input pattern  $X_i$ , an additional criterion based on the probability generated by number of samples included in winner hyperboxes and membership values or the Manhattan distance between the central point of winner hyperboxes and the input sample is used to find the final winner hyperbox that its class label is used for predicting the class label of the input pattern  $X_i$ .

---

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The data matrix for which we want to predict the targets.

**type\_boundary\_handling**

[int, optional, default=PROBABILITY\_MEASURE (aka 1)] The way of handling many winner hyperboxes, i.e., PROBABILITY\_MEASURE or MANHATTAN\_DIS

**Returns****y\_pred**

[ndarray of shape (n\_samples,)] Vector containing the predictions. In binary and multiclass problems, this is a vector containing *n\_samples*.

**simple\_pruning**(*Xl\_val*, *Xu\_val*, *y\_val*, *acc\_threshold*=0.5, *keep\_empty\_boxes*=False, *type\_boundary\_handling*=1)

Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox

**Parameters****Xl\_val**

[array-like of shape (n\_samples, n\_features)] The data matrix contains lower bounds of validation patterns.

**Xu\_val**

[array-like of shape (n\_samples, n\_features)] The data matrix contains upper bounds of validation patterns.

**y\_val**

[ndarray of shape (n\_samples,)] A vector contains the true class label corresponding to each validation pattern.

**acc\_threshold**

[float, optional, default=0.5] The minimum accuracy for each hyperbox to be kept unchanged.

**keep\_empty\_boxes**

[boolean, optional, default=False] Whether to keep the hyperboxes which do not join the prediction process on the validation set. If True, keep them, else the decision for keeping or removing based on the classification accuracy on the validation dataset

**type\_boundary\_handling**

[int, optional, default=PROBABILITY\_MEASURE (aka 1)] The way of handling samples located on the boundary.

**Returns****self**

A hyperbox-based model with the low-qualified hyperboxes pruned.

### 2.6.3 incremental\_learner.fmnn

Simpson fuzzy min-max neural network classifier trained by the incremental learning algorithm.

```
class hbbrain.numerical_data.incremental_learner.fmnn.FMNNClassifier(theta=0.5, gamma=1,  
                                                                    is_draw=False, V=None,  
                                                                    W=None, C=None)
```

Bases: *BaseFMNNClassifier*

Simpson fuzzy min-max neural network classifier.

This class implements an original incremental learning algorithm to train a fuzzy min-max neural network classifier. The details of this algorithm can be found in [1].

#### Parameters

##### **theta**

[float, optional, default=0.5] Maximum hyperbox size for numerical features.

##### **gamma**

[float or ndarray of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each continuous feature.

##### **is\_draw**

[boolean, optional, default=False] Whether the construction of hyperboxes can be progressively shown during the training process on a canvas window.

##### **V**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all minimal points for numerical features of all existing hyperboxes, in which each row is a minimal point of a hyperbox.

##### **W**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all maximal points for numerical features of all existing hyperboxes, in which each row is a maximal point of a hyperbox.

##### **C**

[array-like of shape (n\_hyperboxes,)] A vector stores all class labels corresponding to existing hyperboxes.

#### References

[1]

#### Examples

```
>>> from sklearn.datasets import load_iris
>>> from hbbrain.numerical_data.incremental_learner.fmnn import FMNNClassifier
>>> X, y = load_iris(return_X_y=True)
>>> from sklearn.preprocessing import MinMaxScaler
>>> scaler = MinMaxScaler()
>>> scaler.fit(X)
MinMaxScaler()
>>> X = scaler.transform(X)
>>> clf = FMNNClassifier(theta=0.1).fit(X, y)
```

(continues on next page)



(continued from previous page)

```
>>> clf.predict(X[[10, 50, 100]])
array([0, 1, 2])
```

**Attributes****elapsed\_training\_time**

[float] Training time in seconds.

**Methods**

<code>delay([delay_constant])</code>	Delay a time period to display hyperboxes
<code>draw_hyperbox_and_boundary([window_name, ...])</code>	Draw the existing hyperboxes and their decision boundaries among classes
<code>fit(X, y)</code>	Fit the fuzzy min-max neural network classifier according to the given training data using the Simpson's original incremental learning algorithm.
<code>get_n_hyperboxes()</code>	Get number of hyperboxes in the trained hyperbox-based model
<code>get_params([deep])</code>	Get parameters for this estimator.
<code>get_sample_explanation(x)</code>	Get useful information for explaining the reason behind the predicted result for the input pattern
<code>initialise_canvas_graph([n_dims, ...])</code>	Initialise a canvas to draw hyperboxes
<code>predict(X)</code>	Predict class labels for samples in <i>X</i> .
<code>predict_proba(X)</code>	Predict class probabilities of the input samples <i>X</i> .
<code>predict_with_membership(X)</code>	Predict class membership values of the input samples <i>X</i> .
<code>score(X, y[, sample_weight])</code>	Return the mean accuracy on the given test data and labels.
<code>set_params(**params)</code>	Set the parameters of this estimator.
<code>show_sample_explanation(xl, xu, ...[, ...])</code>	Show explanation for predicted results of an input pattern under the form of parallel coordinates or hyperboxes in 2D or 3D planes.
<code>simple_pruning(X_val, y_val[, ...])</code>	Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox

**fit(X, y)**

Fit the fuzzy min-max neural network classifier according to the given training data using the Simpson's original incremental learning algorithm.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] Training vector, where *n\_samples* is the number of samples and *n\_features* is the number of features.

**y**

[array-like of shape (n\_samples,)] Target vector relative to *X*.

**Returns****self**

[object] Fitted fuzzy min-max neural network.

## 2.6.4 incremental\_learner.efmnn

Fuzzy min-max neural network classifier trained by the enhanced incremental learning algorithm (EFMNN).

```
class hbbrain.numerical_data.incremental_learner.efmnn.EFMNNClassifier(theta=0.5, gamma=1,  
                                                                    is_draw=False,  
                                                                    V=None, W=None,  
                                                                    C=None)
```

Bases: *BaseFMNNClassifier*

Enhanced fuzzy min-max neural network classifier.

This class implements an enhanced learning algorithm for Simpson's fuzzy min-max neural network. This algorithm use nine test cases for hyperbox overlap test and hyperbox contraction instead of four test cases in the original Simpson's fuzzy min-max neural network (FMNN). Additionally, this algorithm use the same hyperbox expansion condition regarding the maximum hyperbox size as the general fuzzy min-max neural network. The details of this algorithm can be found in [1].

### Parameters

#### **theta**

[float, optional, default=0.5] Maximum hyperbox size for numerical features.

#### **gamma**

[float or ndarray of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each continuous feature.

#### **is\_draw**

[boolean, optional, default=False] Whether the construction of hyperboxes can be progressively shown during the training process on a canvas window.

#### **V**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all minimal points for numerical features of all existing hyperboxes, in which each row is a minimal point of a hyperbox.

#### **W**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all maximal points for numerical features of all existing hyperboxes, in which each row is a maximal point of a hyperbox.

#### **C**

[array-like of shape (n\_hyperboxes,)] A vector stores all class labels corresponding to existing hyperboxes.

### References

[1]

## Examples

```
>>> from sklearn.datasets import load_iris
>>> from hbbrain.numerical_data.incremental_learner.efmnn import EFMNNClassifier
>>> X, y = load_iris(return_X_y=True)
>>> from sklearn.preprocessing import MinMaxScaler
>>> scaler = MinMaxScaler()
>>> scaler.fit(X)
MinMaxScaler()
>>> X = scaler.transform(X)
>>> clf = EFMNNClassifier(theta=0.1).fit(X, y)
>>> clf.predict(X[[10, 50, 100]])
array([0, 1, 2])
```

## Attributes

**elapsed\_training\_time**  
[float] Training time in seconds.

## Methods

<code>delay([delay_constant])</code>	Delay a time period to display hyperboxes
<code>draw_hyperbox_and_boundary([window_name, ...])</code>	Draw the existing hyperboxes and their decision boundaries among classes
<code><i>fit</i>(X, y)</code>	Fit the fuzzy min-max neural network according to the given training data using the enhanced learning algorithm.
<code>get_n_hyperboxes()</code>	Get number of hyperboxes in the trained hyperbox-based model
<code>get_params([deep])</code>	Get parameters for this estimator.
<code>get_sample_explanation(x)</code>	Get useful information for explaining the reason behind the predicted result for the input pattern
<code>initialise_canvas_graph([n_dims, ...])</code>	Initialise a canvas to draw hyperboxes
<code>predict(X)</code>	Predict class labels for samples in X.
<code>predict_proba(X)</code>	Predict class probabilities of the input samples X.
<code>predict_with_membership(X)</code>	Predict class membership values of the input samples X.
<code>score(X, y[, sample_weight])</code>	Return the mean accuracy on the given test data and labels.
<code>set_params(**params)</code>	Set the parameters of this estimator.
<code>show_sample_explanation(xl, xu, ...[, ...])</code>	Show explanation for predicted results of an input pattern under the form of parallel coordinates or hyperboxes in 2D or 3D planes.
<code>simple_pruning(X_val, y_val[, ...])</code>	Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox

## **fit**(X, y)

Fit the fuzzy min-max neural network according to the given training data using the enhanced learning algorithm.

## Parameters

**X**

[array-like of shape (n\_samples, n\_features)] Training vector, where *n\_samples* is the number of samples and *n\_features* is the number of features.

**y**

[array-like of shape (n\_samples,)] Target vector relative to X.

**Returns****self**

[object] Fitted fuzzy min-max neural network.

### 2.6.5 incremental\_learner.knefmnn

Enhanced fuzzy min-max neural network classifier with k-nearest hyperboxes expansion rules trained by the incremental learning algorithm.

```
class hbbrain.numerical_data.incremental_learner.knefmnn.KNEFMNNClassifier(theta=0.5,
                                                                           gamma=1,
                                                                           k_neighbors=5,
                                                                           is_draw=False,
                                                                           V=None,
                                                                           W=None,
                                                                           C=None)
```

Bases: [BaseFMNNClassifier](#)

Enhanced fuzzy min-max neural network classifier with k-nearest hyperboxes expansion rules using the incremental learning algorithm.

This class implements the enhanced online learning algorithm with k-nearest hyperboxes expansion rules to train the Simpson's fuzzy min-max neural network. Rather than creating a new hyperbox when the selected winner hyperbox is not satisfied with the expansion condition as in the enhanced fuzzy min-max neural network, this algorithm considers up to k winner hyperboxes. The creation of a new hyperbox only happens when all k selected winner hyperboxes cannot be extended to cover the input pattern. The details of this algorithm can be found in [1].

**Parameters****theta**

[float, optional, default=0.5] Maximum hyperbox size for numerical features.

**gamma**

[float or ndarray of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each continuous feature.

**k\_neighbors**

[int, optional, default = 5] Number of nearest hyperboxes used for in the consideration of expansion rules.

**is\_draw**

[boolean, optional, default=False] Whether the construction of hyperboxes can be progressively shown during the training process on a canvas window.

**V**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all minimal points for numerical features of all existing hyperboxes, in which each row is a minimal point of a hyperbox.

**W**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all maximal points for numerical features of all existing hyperboxes, in which each row is a minimal point of a hyperbox.

**C**

[array-like of shape (n\_hyperboxes,)] A vector stores all class labels corresponding to existing hyperboxes.

## References

[1]

## Examples

```
>>> from sklearn.datasets import load_iris
>>> from hbbrain.numerical_data.incremental_learner.knefmnn import KNEFMNNClassifier
>>> X, y = load_iris(return_X_y=True)
>>> from sklearn.preprocessing import MinMaxScaler
>>> scaler = MinMaxScaler()
>>> scaler.fit(X)
MinMaxScaler()
>>> X = scaler.transform(X)
>>> clf = KNEFMNNClassifier(theta=0.1, k_neighbors=5).fit(X, y)
>>> clf.predict(X[[10, 50, 100]])
array([0, 1, 2])
```

## Attributes

**elapsed\_training\_time**

[float] Training time in seconds.

## Methods

<code>delay([delay_constant])</code>	Delay a time period to display hyperboxes
<code>draw_hyperbox_and_boundary([window_name, ...])</code>	Draw the existing hyperboxes and their decision boundaries among classes
<code><i>fit</i>(X, y)</code>	Fit the model according to the given training data using the enhanced online learning algorithm with the k-nearest hyperbox selection rule.
<code>get_n_hyperboxes()</code>	Get number of hyperboxes in the trained hyperbox-based model
<code>get_params([deep])</code>	Get parameters for this estimator.
<code>get_sample_explanation(x)</code>	Get useful information for explaining the reason behind the predicted result for the input pattern
<code>initialise_canvas_graph([n_dims, ...])</code>	Initialise a canvas to draw hyperboxes
<code>predict(X)</code>	Predict class labels for samples in X.
<code>predict_proba(X)</code>	Predict class probabilities of the input samples X.
<code>predict_with_membership(X)</code>	Predict class membership values of the input samples X.
<code>score(X, y[, sample_weight])</code>	Return the mean accuracy on the given test data and labels.
<code>set_params(**params)</code>	Set the parameters of this estimator.
<code>show_sample_explanation(xl, xu, ...[, ...])</code>	Show explanation for predicted results of an input pattern under the form of parallel coordinates or hyperboxes in 2D or 3D planes.
<code>simple_pruning(X_val, y_val[, ...])</code>	Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox

### ***fit*(X, y)**

Fit the model according to the given training data using the enhanced online learning algorithm with the k-nearest hyperbox selection rule.

#### **Parameters**

##### **X**

[array-like of shape (n\_samples, n\_features)] Training vector, where *n\_samples* is the number of samples and *n\_features* is the number of features.

##### **y**

[array-like of shape (n\_samples,)] Target vector relative to X.

#### **Returns**

##### **self**

[object] Fitted fuzzy min-max neural network.

## 2.6.6 incremental\_learner.rfmnn

Refined fuzzy min-max neural network classifier trained by the incremental learning algorithm.

```
class hbbrain.numerical_data.incremental_learner.rfmnn.RFMNNClassifier(theta=0.5, gamma=1,  
                                                                    is_draw=False,  
                                                                    V=None, W=None,  
                                                                    C=None)
```

Bases: *BaseFMNNClassifier*

Refined fuzzy min-max neural network classifier.

This class implements essential functions for a refined online learning algorithm to train a fuzzy min-max neural network. This algorithm proposes a new expansion procedure for addressing the problems of overlap leniency and irregularity of hyperbox expansion. It avoids the overlap cases between hyperboxes from different classes, reducing the number of overlap cases to one (containment case) as in the improved online learning algorithm. It also introduces a new formula that simplifies the overlap test procedure. Moreover, it introduces a new contraction procedure for overcoming the data distortion problem and providing more accurate decision boundaries for the contracted hyperboxes is proposed. The details of this algorithm can be found in [1].

### Parameters

#### **theta**

[float, optional, default=0.5] Maximum hyperbox size for numerical features.

#### **gamma**

[float or ndarray of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each continuous feature.

#### **is\_draw**

[boolean, optional, default=False] Whether the construction of hyperboxes can be progressively shown during the training process on a canvas window.

#### **V**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all minimal points for numerical features of all existing hyperboxes, in which each row is a minimal point of a hyperbox.

#### **W**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all maximal points for numerical features of all existing hyperboxes, in which each row is a maximal point of a hyperbox.

#### **C**

[array-like of shape (n\_hyperboxes,)] A vector stores all class labels corresponding to existing hyperboxes.

### References

[1]

## Examples

```
>>> from sklearn.datasets import load_iris
>>> from hbbrain.numerical_data.incremental_learner.rfmnn import RFMNNClassifier
>>> X, y = load_iris(return_X_y=True)
>>> from sklearn.preprocessing import MinMaxScaler
>>> scaler = MinMaxScaler()
>>> scaler.fit(X)
MinMaxScaler()
>>> X = scaler.transform(X)
>>> clf = RFMNNClassifier(theta=0.1).fit(X, y)
>>> clf.predict(X[[10, 50, 100]])
array([0, 1, 2])
```

## Attributes

**elapsed\_training\_time**  
[float] Training time in seconds.

## Methods

<code>delay([delay_constant])</code>	Delay a time period to display hyperboxes
<code>draw_hyperbox_and_boundary([window_name, ...])</code>	Draw the existing hyperboxes and their decision boundaries among classes
<code><i>fit</i>(X, y)</code>	Fit the model according to the given training data using the refined online learning algorithm.
<code>get_n_hyperboxes()</code>	Get number of hyperboxes in the trained hyperbox-based model
<code>get_params([deep])</code>	Get parameters for this estimator.
<code><i>get_sample_explanation</i>(x)</code>	Get useful information for explaining the reason behind the predicted result for the input pattern
<code>initialise_canvas_graph([n_dims, ...])</code>	Initialise a canvas to draw hyperboxes
<code><i>predict</i>(X)</code>	Predict class labels for samples in X.
<code>predict_proba(X)</code>	Predict class probabilities of the input samples X.
<code>predict_with_membership(X)</code>	Predict class membership values of the input samples X.
<code>score(X, y[, sample_weight])</code>	Return the mean accuracy on the given test data and labels.
<code>set_params(**params)</code>	Set the parameters of this estimator.
<code>show_sample_explanation(xl, xu, ...[, ...])</code>	Show explanation for predicted results of an input pattern under the form of parallel coordinates or hyperboxes in 2D or 3D planes.
<code><i>simple_pruning</i>(X_val, y_val[, ...])</code>	Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox

## `fit(X, y)`

Fit the model according to the given training data using the refined online learning algorithm.

### Parameters

**X**



[array-like of shape (n\_samples, n\_features)] Training vector, where *n\_samples* is the number of samples and *n\_features* is the number of features.

**y**

[array-like of shape (n\_samples,)] Target vector relative to X.

### Returns

**self**

[object] Fitted fuzzy min-max neural network.

### get\_sample\_explanation(*x*)

Get useful information for explaining the reason behind the predicted result for the input pattern

### Parameters

**x**

[ndarray of shape (n\_feature,)] The input pattern which needs to be explained.

### Returns

**y\_pred**

[int] The predicted class of the input pattern

**dict\_mem\_val\_classes**

[dictionary] A dictionary stores all membership values for all classes. The key is class label and the value is the corresponding membership value.

**dict\_min\_point\_classes**

[dictionary] A dictionary stores all minimal points of hyperboxes having the maximum membership value for each class. The key is the class label and the value is the minimal points of all hyperboxes corresponding to each class

**dict\_max\_point\_classes**

[dictionary] A dictionary stores all maximal points of hyperboxes having the maximum membership value for each class. The key is the class label and the value is the maximal points of all hyperboxes corresponding to each class

### predict(*X*)

Predict class labels for samples in X.

---

**Note:** In the case there are many winner hyperboxes representing different class labels but with the same membership value with respect to the input pattern  $X_i$ , an additional criterion based on the minimum rfmmn distance between the input pattern  $X_i$  and the winner hyperboxes are used to find the final winner hyperbox that its class label is used for predicting the class label of the input pattern  $X_i$ .

---

### Parameters

**X**

[array-like of shape (n\_samples, n\_features)] The data matrix for which we want to predict the targets.

### Returns

**y\_pred**

[ndarray of shape (n\_samples,)] Vector containing the predictions. In binary and multiclass problems, this is a vector containing *n\_samples*.

**simple\_pruning**(*X\_val*, *y\_val*, *acc\_threshold*=0.5, *keep\_empty\_boxes*=False)

Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox

**Parameters**

**X\_val**

[array-like of shape (n\_samples, n\_features)] The data matrix contains validation patterns.

**y\_val**

[ndarray of shape (n\_samples,)] A vector contains the true class label corresponding to each validation pattern.

**acc\_threshold**

[float, optional, default=0.5] The minimum accuracy for each hyperbox to be kept unchanged.

**keep\_empty\_boxes**

[boolean, optional, default=False] Whether to keep the hyperboxes which do not join the prediction process on the validation set. If True, keep them, else the decision for keeping or removing based on the classification accuracy on the validation dataset

**Returns**

**self**

A hyperbox-based model with the low-qualified hyperboxes pruned.

`hbbrain.numerical_data.incremental_learner.rfmnn.predict_rfmnn(V, W, C, X, g)`

Predict class labels for samples in *X*.

---

**Note:** This is a function to determine the right class labels for *X* with regard to a trained hyperbox-based classifier represented by [*V*, *W*, *C*]. It uses the winner-takes-all principle to predict class labels for each sample in *X* by assigning the class label of the sample to the class label of the hyperbox with the maximum membership value to that sample. It will use a specific distance designed for the refined fuzzy min-max neural networks in the case of many hyperboxes with different classes having the same maximum membership value.

---

**Parameters**

**V**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all minimal points of all hyperboxes of a trained hyperbox-based model, in which each row is a minimal point of a hyperbox.

**W**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all maximal points of all hyperboxes of a trained hyperbox-based model, in which each row is a maximal point of a hyperbox.

**C**

[ndarray of shape (n\_hyperboxes,)] An array contains all class labels for all hyperboxes of a trained hyperbox-based model.

**X**

[array-like of shape (n\_samples, n\_features)] The data matrix contains input patterns for which we want to predict the targets.

**g**

[float or array-like of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each dimension.

**Returns****y\_pred**

[ndarray of shape (n\_samples,)] A vector contains the predictions. In binary and multiclass problems, this is a vector containing *n\_samples*.

## 2.7 multigranular learners

### 2.7.1 multigranular\_learner.multi\_resolution\_gfmm

A multi-resolution hierarchical granular representation based classifier using general fuzzy min-max neural network.

```
class hbbrain.numerical_data.multigranular_learner.multi_resolution_gfmm.MultiGranularGFMM(n_partitions=4,  
                                                                                       gran-  
                                                                                       u-  
                                                                                       lar_theta=[0.1,  
                                                                                       0.2,  
                                                                                       0.3],  
                                                                                       gamma=1,  
                                                                                       min_membership  
                                                                                       ran-  
                                                                                       dom_state=0)
```

Bases: *BaseHyperboxClassifier*

A multi-resolution hierarchical granular representation based classifier using general fuzzy min-max neural network.

This class implements the multi-granular learning algorithm to construct classifiers from multiresolution hierarchical granular representations using hyperbox fuzzy sets. This algorithm forms a series of granular inferences hierarchically through many levels of abstraction. An attractive characteristic of our classifier is that it can maintain a high accuracy in comparison to other fuzzy min-max models at a low degree of granularity based on reusing the knowledge learned from lower levels of abstraction. In addition, our approach can reduce the data size significantly as well as handle the uncertainty and incompleteness associated with data in real-world applications. The construction process of the classifier consists of two phases. The first phase is to formulate the model at the greatest level of granularity, while the later stage aims to reduce the complexity of the constructed model and deduce it from data at higher abstraction levels. The details of this algorithm can be found in [1].

**Parameters****n\_partitions**

[int, default=4] Number of partitions to split the original training set into disjoint training sets to build base learners.

**granular\_theta**

[list of float, optional, default=[0.1, 0.2, 0.3]] Maximum hyperbox sizes at granularity levels.

**gamma**

[float or ndarray of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each continuous feature.

**min\_membership\_aggregation**

[float, optional, default=0.5] Minimum membership value between two hyperboxes aggregated to form a larger sized hyperbox at a higher level of abstraction.

**random\_state**

[int, RandomState instance or None, default=None] Controls the stratified random sampling rate of the original dataset to form disjoint subsets for training base learners.

## References

[1]

## Examples

```
>>> from sklearn.datasets import load_iris
>>> from hbbrain.numerical_data.multigranular_learner.multi_resolution_gfmm import _
↳ MultiGranularGFMM
>>> X, y = load_iris(return_X_y=True)
>>> from sklearn.preprocessing import MinMaxScaler
>>> scaler = MinMaxScaler()
>>> scaler.fit(X)
MinMaxScaler()
>>> X = scaler.transform(X)
>>> clf = MultiGranularGFMM(n_partitions=2, granular_theta=[0.1, 0.2, 0.3, 0.4, 0.
↳ 5], gamma=1, min_membership_aggregation=0.6, random_state=0)
>>> clf.fit(X, y)
>>> clf.predict(X[[10, 50, 100]])
array([0, 1, 2])
>>> clf.predict(X[[10, 50, 100]], level=0)
array([0, 1, 2])
>>> print("Number of hyperboxes at granularity 1 = %d"%clf.get_n_hyperboxes(0))
Number of hyperboxes at granularity 1 = 77
>>> clf.predict(X[[10, 50, 100]], level=4)
array([0, 1, 2])
>>> print("Number of hyperboxes at granularity 5 = %d"%clf.get_n_hyperboxes(4))
Number of hyperboxes at granularity 5 = 11
```

## Attributes

### **granularity\_level**

[dict] A mapping between the maximum hyperbox size and the granular level.

### **smallest\_theta**

[float] Maximum hyperbox size at the highest granularity level.

### **higher\_level\_theta**

[list of float] Maximum hyperbox sizes of higher abstraction levels apart from the highest granularity level.

### **granular\_classifiers\_**

[ndarray of BaseGranular objects with shape (n\_granularity\_levels,)] A list of general fuzzy min-max neural networks at all granularity levels.

### **base\_learners\_**

[list] A list of base learners trained from disjoint subsets of input training patterns.

### **is\_exist\_missing\_value**

[boolean] Is there any missing values in continuous features in the training data.

### **elapsed\_training\_time**

[float] Training time in seconds.

## Methods

<code>delay([delay_constant])</code>	Delay a time period to display hyperboxes
<code>draw_2D_hyperbox_and_boundary_granular_level(window_name, level)</code>	Draw the existing hyperboxes and their decision boundaries among classes at a given granularity level.
<code>draw_2D_hyperbox_and_boundary_partitions([window_name, level, partition])</code>	Draw the existing hyperboxes and their decision boundaries among classes in a given partition.
<code>draw_hyperbox_and_boundary([window_name, ...])</code>	Draw the existing hyperboxes and their decision boundaries among classes
<code>fit(X, y[, learning_type, X_val, y_val, ...])</code>	Fit the model according to the given training data using the multi granularity learning algorithm.
<code>get_n_hyperboxes([level])</code>	Get number of hyperboxes at a given granularity level.
<code>get_n_hyperboxes_at_partition([partition])</code>	Get number of hyperboxes at a given granularity level.
<code>get_params([deep])</code>	Get parameters for this estimator.
<code>get_sample_explanation_granular_level(xl, xu)</code>	Get useful information for explaining the reason behind the predicted result for the input pattern
<code>granular_learning_phase_1(Xl, Xu, y[, ...])</code>	Training a granular general fuzzy min-max neural network using a learning algorithm in phase 1 to distribute disjoint subsets into working processes to build base learners.
<code>granular_learning_phase_2()</code>	Training a granular general fuzzy min-max neural network using a learning algorithm in phase 2 to reduce number of hyperboxes while keeping a good classification performance.
<code>initialise_canvas_graph([n_dims, ...])</code>	Initialise a canvas to draw hyperboxes
<code>predict(X[, level])</code>	Predict class labels for samples in $X$ at a given granularity level.
<code>predict_at_partitions(Xl, Xu[, partition])</code>	Predict class labels for samples in the form of hyperboxes represented by low bounds $Xl$ and upper bounds $Xu$ at a given granularity level.
<code>predict_proba(X[, level])</code>	Predict class probabilities of the input samples $X$ at a given granularity level.
<code>predict_with_membership(X[, level])</code>	Predict class memberships of the input samples $X$ at a given granularity level.
<code>score(X, y[, sample_weight])</code>	Return the mean accuracy on the given test data and labels.
<code>set_params(**params)</code>	Set the parameters of this estimator.
<code>show_sample_explanation(xl, xu, ...[, ...])</code>	Show explanation for predicted results of an input pattern under the form of parallel coordinates or hyperboxes in 2D or 3D planes.
<code>simple_pruning(V, W, C, N_samples, ...[, ...])</code>	Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox

**`draw_2D_hyperbox_and_boundary_granular_level`**(*window\_name*='Hyperbox-based classifier and its decision boundaries', *level*=0)

Draw the existing hyperboxes and their decision boundaries among classes at a given granularity level.

**Note:** This method only works on 2-dimensional datasets.

**Parameters****window\_name**

[str, optional, default="Hyperbox-based classifier and its decision boundaries"] Name of plotting window showing hyperboxes and their decision boundaries.

**level**

[int, optional, default=0] The granularity level needs to draw hyperboxes and its boundaries.

**Returns**

None.

**draw\_2D\_hyperbox\_and\_boundary\_partitions**(*window\_name='Base learners and its decision boundaries', partition=0, fig\_num=100*)

Draw the existing hyperboxes and their decision boundaries among classes in a given partition.

---

**Note:** This method only works on 2-dimensional datasets.

---

**Parameters****window\_name**

[str, optional, default="Hyperbox-based classifier and its decision boundaries"] Name of plotting window showing hyperboxes and their decision boundaries.

**partition**

[int, optional, default=0] The partition needs to draw hyperboxes and its boundary.

**fig\_num**

[int, optional, default=100] Index of the drawing canvas.

**Returns**

None.

**fit**(*X, y, learning\_type=1, X\_val=None, y\_val=None, acc\_threshold=0.5, keep\_empty\_boxes=False*)

Fit the model according to the given training data using the multi granularity learning algorithm.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] Training vector, where *n\_samples* is the number of samples and *n\_features* is the number of features.

**y**

[array-like of shape (n\_samples,)] Target vector relative to X.

**learning\_type**

[enum (int), optional, default=HETEROGENEOUS\_CLASS\_LEARNING] Learning type is used to build base learners from disjoint datasets. It gets two defined enum values being HETEROGENEOUS\_CLASS\_LEARNING and HOMOGENEOUS\_CLASS\_LEARNING. Heterogeneous class learning means that base learners are trained based on the order of input samples. Homogeneous class learning means that input data are sorted and grouped according to class labels before starting the training process.

**X\_val**

[array-like of shape (n\_val\_samples, n\_features), optional, default=None] A matrix contains a validation set, where *n\_val\_samples* is the number of validation samples and *n\_features* is the number of features.

**y\_val**  
[array-like of shape (n\_val\_samples,), optional, default=None] Target vector relative to X\_val.

**acc\_threshold**  
[float, optional, default=0.5] The minimum accuracy for each hyperbox to be kept unchanged.

**keep\_empty\_boxes**  
[boolean, optional, default=False] Whether to keep the hyperboxes which do not join the prediction process on the validation set. If True, keep them, else the decision for keeping or removing based on the classification accuracy on the validation dataset.

#### Returns

**self**  
[object] Fitted multigranular general fuzzy min-max neural network.

#### **get\_n\_hyperboxes**(*level=-1*)

Get number of hyperboxes at a given granularity level.

#### Parameters

**level**  
[int, optional, default=-1] The granularity level needs to get number of hyperboxes. If *level* gets a value of -1, return number of hyperboxes in all granularity levels.

#### Returns

**int**  
Number of hyperboxes at the given granularity level.

#### **get\_n\_hyperboxes\_at\_partition**(*partition=0*)

Get number of hyperboxes at a given granularity level.

#### Parameters

**partition**  
[int, optional, default=0] The partition needs to get number of base learners.

#### Returns

**int**  
Number of hyperboxes at the given partition.

#### **get\_sample\_explanation\_granular\_level**(*xl, xu, level=0*)

Get useful information for explaining the reason behind the predicted result for the input pattern

#### Parameters

**xl**  
[ndarray of shape (n\_feature,)] Minimum point of the input pattern which needs to be explained.

**xu**  
[ndarray of shape (n\_feature,)] Maximum point of the input pattern which needs to be explained.

**level**  
[int, optional, default=0] The granularity level is used to generate prediction.

#### Returns

**y\_pred**

[int] The predicted class of the input pattern

**dict\_mem\_val\_classes**

[dictionary] A dictionary stores all membership values for all classes. The key is class label and the value is the corresponding membership value.

**dict\_min\_point\_classes**

[dictionary] A dictionary stores all minimal points of hyperboxes having the maximum membership value for each class. The key is the class label and the value is the minimal points of all hyperboxes corresponding to each class

**dict\_max\_point\_classes**

[dictionary] A dictionary stores all maximal points of hyperboxes having the maximum membership value for each class. The key is the class label and the value is the maximal points of all hyperboxes corresponding to each class

**granular\_learning\_phase\_1**(*Xl, Xu, y, learning\_type=1, X\_val=None, y\_val=None, acc\_threshold=0.5, keep\_empty\_boxes=False*)

Training a granular general fuzzy min-max neural network using a learning algorithm in phase 1 to distribute disjoint subsets into working processes to build base learners. After that, resulting hyperboxes from all base learners will merged and pruned.

**Parameters****Xl**

[array-like of shape (n\_samples, n\_features)] The data matrix contains lower bounds of input training patterns.

**Xu**

[array-like of shape (n\_samples, n\_features)] The data matrix contains upper bounds of input training patterns.

**y**

[array-like of shape (n\_samples,)] Target vector relative to input training hyperboxes [*Xl, Xu*].

**learning\_type**

[enum (int), optional, default=HETEROGENEOUS\_CLASS\_LEARNING] Learning type is used to build base learners from disjoint datasets. It gets two defined enum values being HETEROGENEOUS\_CLASS\_LEARNING and HOMOGENEOUS\_CLASS\_LEARNING. Heterogeneous class learning means that base learners are trained based on the order of input samples. Homogeneous class learning means that input data are sorted and grouped according to class labels before starting the training process.

**X\_val**

[array-like of shape (n\_samples, n\_features)] The data matrix contains validation patterns.

**y\_val**

[ndarray of shape (n\_samples,)] A vector contains the true class label corresponding to each validation pattern.

**acc\_threshold**

[float, optional, default=0.5] The minimum accuracy for each hyperbox to be kept unchanged.

**keep\_empty\_boxes**

[boolean, optional, default=False] Whether to keep the hyperboxes which do not join the prediction process on the validation set. If True, keep them, else the decision for keeping or removing based on the classification accuracy on the validation dataset



**Returns****self**

[object] A granular general fuzzy min-max neural network trained by a phase-1 learning algorithm.

**granular\_learning\_phase\_2()**

Training a granular general fuzzy min-max neural network using a learning algorithm in phase 2 to reduce number of hyperboxes while keeping a good classification performance.

**Returns****self**

[object] A granular general fuzzy min-max neural network trained by a phase-2 learning algorithm.

**predict(X, level=-1)**

Predict class labels for samples in  $X$  at a given granularity level.

---

**Note:** In the case there are many winner hyperboxes representing different class labels but with the same membership value with respect to the input pattern  $X_i$ , an additional criterion based on the minimum distance between the input samples and the centroids of the winner hyperboxes is used to find the final winner hyperbox that its class label is used for predicting the class label of the input pattern  $X_i$ .

---

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The data matrix for which we want to predict the targets.

**level**

[int, optional, default=-1] The granularity level is used to generate predicted classes for the input testing samples. If this variable gets the values of -1, then the predicted class for each sample is the class getting the most votes from all available granularity levels.

**Returns****y\_pred**

[ndarray of shape (n\_samples,)] Vector containing the predictions. In binary and multiclass problems, this is a vector containing  $n_{samples}$ .

**predict\_at\_partitions(Xl, Xu, partition=0)**

Predict class labels for samples in the form of hyperboxes represented by low bounds  $Xl$  and upper bounds  $Xu$  at a given granularity level.

---

**Note:** In the case there are many winner hyperboxes representing different class labels but with the same membership value with respect to the input pattern  $X_i$  in the form of an hyperbox represented by a lower bound  $Xl_i$  and an upper bound  $Xu_i$ , an additional criterion based on the minimum distance between the centroids of winner hyperboxes and the input sample is used to find the final winner hyperbox that its class label is used for predicting the class label of the input hyperbox  $X_i$ .

---

**Parameters**

**Xl**

[array-like of shape (n\_samples, n\_features)] The data matrix containing the lower bounds of input patterns for which we want to predict the targets.

**Xu**

[array-like of shape (n\_samples, n\_features)] The data matrix containing the upper bounds of input patterns for which we want to predict the targets.

**partition**

[int, optional, default=0] The base learner at a given partition is used to generate predicted classes for the input testing samples.

**Returns****y\_pred**

[ndarray of shape (n\_samples,)] Vector containing the predictions. In binary and multiclass problems, this is a vector containing *n\_samples*.

**predict\_proba(X, level=-1)**

Predict class probabilities of the input samples X at a given granularity level.

The predicted class probability at a given granularity level is the fraction of the membership value of the representative hyperbox of that class at the given granularity level and the sum of all membership values of all representative hyperboxes of all classes joining the prediction procedure.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The input samples.

**level**

[int, optional, default=-1] The granularity level is used to generate predicted class probabilities for the input testing samples. If this variable gets the values of -1, then the predicted class probability value for each sample is the average of probability values at all available granularity levels.

**Returns****proba**

[ndarray of shape (n\_samples, n\_classes)] The class probabilities of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

**predict\_with\_membership(X, level=-1)**

Predict class memberships of the input samples X at a given granularity level.

The predicted class memberships are the membership values of the representative hyperbox of that class at a given granularity level.

**Parameters****X**

[array-like of shape (n\_samples, n\_features)] The input samples.

**level**

[int, optional, default=-1] The granularity level is used to generate predicted classes for the input testing samples. If this variable gets the values of -1, then the predicted class membership value for each sample is the average of all class memberships of all granularity levels.

**Returns**

**mem\_vals**

[ndarray of shape (n\_samples, n\_classes)] The class memberships of the input samples. The order of the classes corresponds to that in ascending integers of class labels.

**simple\_pruning**(*V, W, C, N\_samples, Centroids, Xl\_val, Xu\_val, y\_val, acc\_threshold=0.5, keep\_empty\_boxes=False*)

Simply prune low qualified hyperboxes based on a pre-defined accuracy threshold for each hyperbox

**Parameters****V**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all minimal points for numerical features of all existing hyperboxes, in which each row is a minimal point of a hyperbox.

**W**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all maximal points for numerical features of all existing hyperboxes, in which each row is a maximal point of a hyperbox.

**C**

[array-like of shape (n\_hyperboxes,)] A vector stores all class labels corresponding to existing hyperboxes.

**N\_samples**

[array-like of shape (n\_hyperboxes,)] A vector stores the number of samples fully included in each existing hyperbox.

**Centroids**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all centroid points of all existing hyperboxes, in which each row is a centroid point of a hyperbox.

**Xl\_val**

[array-like of shape (n\_samples, n\_features)] The data matrix contains lower bounds of validation patterns.

**Xu\_val**

[array-like of shape (n\_samples, n\_features)] The data matrix contains upper bounds of validation patterns.

**y\_val**

[ndarray of shape (n\_samples,)] A vector contains the true class label corresponding to each validation pattern.

**acc\_threshold**

[float, optional, default=0.5] The minimum accuracy for each hyperbox to be kept unchanged.

**keep\_empty\_boxes**

[boolean, optional, default=False] Whether to keep the hyperboxes which do not join the prediction process on the validation set. If True, keep them, else the decision for keeping or removing based on the classification accuracy on the validation dataset

**Returns****new\_V**

[array-like of shape (n\_new\_hyperboxes, n\_features)] A matrix stores all minimal points for numerical features of all remaining hyperboxes after pruning, in which each row is a minimal point of a hyperbox.

**new\_W**

[array-like of shape (n\_new\_hyperboxes, n\_features)] A matrix stores all maximal points for numerical features of all remaining hyperboxes after pruning, in which each row is a maximal point of a hyperbox.

**new\_C**

[array-like of shape (n\_new\_hyperboxes,)] A vector stores all class labels corresponding to remaining hyperboxes after pruning.

**new\_N\_samples**

[array-like of shape (n\_new\_hyperboxes,)] A vector stores the number of samples fully included in each remaining hyperbox after pruning.

**new\_Centroids**

[array-like of shape (n\_new\_hyperboxes, n\_features)] A matrix stores all centroid points of all remaining hyperboxes after pruning, in which each row is a centroid point of a hyperbox.

`hbbrain.numerical_data.multigranular_learner.multi_resolution_gfmm.convert_granular_theta_to_level(granu`

Convert a list of maximum hyperbox sizes to the corresponding granular levels.

**Parameters****granular\_thetas**

[list] A list contains all maximum hyperbox sizes for all granularity levels.

**Returns****level\_dic**

[dict] A mapping between the maximum hyperbox size and the granular level.

`hbbrain.numerical_data.multigranular_learner.multi_resolution_gfmm.predict_with_centroids(V,  
W,  
C,  
N_samples,  
Cen-  
troids,  
Xl,  
Xu,  
g=1)`

Predict class labels for samples in  $X$  represented in the form of intervals  $[Xl, Xu]$ . This is a common function to determine the right class labels for  $X$  wrt. a trained hyperbox-based classifier represented by  $[V, W, C]$ . It uses the winner-takes-all principle to predict class labels for each sample in  $X$  by assigning the class label of the sample to the class label of the hyperbox with the maximum membership value to that sample. It will use an Euclidean distance from the input pattern to the centroid point of the hyperbox in the case of many winner hyperboxes with different classes having the same maximum membership value. If two winner hyperboxes show the same Euclidean distance to their centroid points, the winner hyperbox with a higher number of included samples will be selected.

**Parameters****V**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all minimal points for numerical features of all existing hyperboxes, in which each row is a minimal point of a hyperbox.

**W**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all maximal points for numerical features of all existing hyperboxes, in which each row is a minimal point of a hyperbox.

**C**

[array-like of shape (n\_hyperboxes,)] A vector stores all class labels corresponding to existing hyperboxes.

**N\_samples**

[array-like of shape (n\_hyperboxes,)] A vector stores the number of samples fully included in each existing hyperbox.

**Centroids**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all centroid points of all existing hyperboxes, in which each row is a centroid point of a hyperbox.

**Xl**

[array-like of shape (n\_samples, n\_features)] The data matrix contains lower bounds of input patterns for which we want to predict the targets.

**Xu**

[array-like of shape (n\_samples, n\_features)] The data matrix contains upper bounds of input patterns for which we want to predict the targets.

**g**

[float or array-like of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each dimension.

**Returns****y\_pred**

[ndarray of shape (n\_samples,)] A vector contains the predictions. In binary and multiclass problems, this is a vector containing *n\_samples*.

```
hbbrain.numerical_data.multigranular_learner.multi_resolution_gfmm.predict_with_membership(V,
                                                                                          W,
                                                                                          C,
                                                                                          Xl,
                                                                                          Xu,
                                                                                          g=1)
```

Return class membership values for samples in  $X$  represented in the form of intervals  $[Xl, Xu]$ . This is a common function to determine the membership values from an input  $X$  to a trained hyperbox-based classifier represented by  $[V, W, C]$ .

**Parameters****V**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all minimal points for numerical features of all existing hyperboxes, in which each row is a minimal point of a hyperbox.

**W**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all maximal points for numerical features of all existing hyperboxes, in which each row is a minimal point of a hyperbox.

**C**

[array-like of shape (n\_hyperboxes,)] A vector stores all class labels corresponding to existing hyperboxes.

**Xl**

[array-like of shape (n\_samples, n\_features)] The data matrix contains lower bounds of input patterns for which we want to predict the targets.

**Xu**

[array-like of shape (n\_samples, n\_features)] The data matrix contains upper bounds of input patterns for which we want to predict the targets.

**g**

[float or array-like of shape (n\_features,), optional, default=1] A sensitivity parameter describing the speed of decreasing of the membership function in each dimension.

**Returns****mem\_vals**

[ndarray of shape (n\_samples, n\_classes)] A vector contains the membership values for all classes for each input sample which needs to get the membership values.

`hbbrain.numerical_data.multigranular_learner.multi_resolution_gfmm.remove_contained_hyperboxes(V, W, C, N_samples, Centroids)`

Remove all hyperboxes contained in other hyperboxes with the same class label and update the centroids of larger hyperboxes included the removed hyperboxes.

**Parameters****V**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all minimal points for numerical features of all existing hyperboxes, in which each row is a minimal point of a hyperbox.

**W**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all maximum points for numerical features of all existing hyperboxes, in which each row is a maximum point of a hyperbox.

**C**

[array-like of shape (n\_hyperboxes,)] A vector stores all class labels corresponding to existing hyperboxes.

**N\_samples**

[array-like of shape (n\_hyperboxes,)] A vector stores the number of samples fully included in each existing hyperbox.

**Centroids**

[array-like of shape (n\_hyperboxes, n\_features)] A matrix stores all centroid points of all existing hyperboxes, in which each row is a centroid point of a hyperbox.

**Returns****new\_V**

[array-like of shape (n\_new\_hyperboxes, n\_features)] A matrix stores all minimal points for numerical features of all hyperboxes after removal of fully contained hyperboxes, in which each row is a minimal point of a hyperbox.

**new\_W**

[array-like of shape (n\_new\_hyperboxes, n\_features)] A matrix stores all maximal points for numerical features of all hyperboxes after removal of fully contained hyperboxes, in which each row is a maximal point of a hyperbox.

**new\_C**

[array-like of shape (n\_new\_hyperboxes,)] A vector stores all class labels corresponding to remaining hyperboxes after removal of fully contained hyperboxes.

**new\_N\_samples**

[array-like of shape (n\_new\_hyperboxes,)] A vector stores the number of samples fully included in each hyperbox.

**new\_Centroids**

[array-like of shape (n\_new\_hyperboxes, n\_features)] A matrix stores all centroid points of all remaining hyperboxes after removal of fully contained hyperboxes, in which each row is a centroid point of a hyperbox.

**n\_removed\_hyperboxes**

[int] Number of hyperboxes has been removed because they are included in at least one larger hyperbox with the same class label.

## 2.8 Tutorials

### 2.8.1 Batch learners

#### Agglomerative Learning Algorithm for GFMM

This example shows how to use the GFMM classifier using an agglomerative learning algorithm with full similarity matrix (AGGLO-SM)

Note that the numerical features in training and testing datasets must be in the range of [0, 1] because the GFMM classifiers with AGGLO-SM algorithm require features in the unit cube.

#### 1. Execute directly from the python file

```
[1]: %matplotlib notebook
```

```
[2]: import os
import warnings
warnings.filterwarnings('ignore')
```

#### Get the path to the this jupyter notebook file

```
[3]: this_notebook_dir = os.path.dirname(os.path.abspath("__file__"))
this_notebook_dir
```

```
[3]: 'C:\\hyperbox-brain\\docs\\tutorials'
```

## Get the home folder of the Hyperbox-Brain project

```
[4]: from pathlib import Path
project_dir = Path(this_notebook_dir).parent.parent
project_dir

[4]: WindowsPath('C:/hyperbox-brain')
```

## Create the path to the Python file containing the implementation of the GFMM classifier using the agglomerative learning algorithm with full similarity matrix

```
[5]: agglo_sm_file_path = os.path.join(project_dir, Path("hbbrain/numerical_data/batch_
↪ learner/agglo_gfmm.py"))
agglo_sm_file_path

[5]: 'C:\\hyperbox-brain\\hbbrain\\numerical_data\\batch_learner\\agglo_gfmm.py'
```

## Run the found file by showing the execution directions

```
[6]: !python "{agglo_sm_file_path}" -h

usage: agglo_gfmm.py [-h] -training_file TRAINING_FILE -testing_file
                    TESTING_FILE [--theta THETA] [--gamma GAMMA]
                    [--min_simil MIN_SIMIL]
                    [--simil_measure {mid,long,short}]
                    [--asimil_type {min,max}] [--is_draw IS_DRAW]

The description of parameters

required arguments:
  -training_file TRAINING_FILE
                        A required argument for the path to training data file
                        (including file name)
  -testing_file TESTING_FILE
                        A required argument for the path to testing data file
                        (including file name)

optional arguments:
  --theta THETA          Maximum hyperbox size (in the range of (0, 1])
                        (default: 0.5)
  --gamma GAMMA          A sensitivity parameter describing the speed of
                        decreasing of the membership function in each
                        dimension (larger than 0) (default: 1)
  --min_simil MIN_SIMIL  Mimimum similarity value so that two hyperboxes can be
                        merged (in the range of [0, 1])(default: 0.5)
  --simil_measure {mid,long,short}
                        Type of similarity measure (default: mid)
  --asimil_type {min,max}
                        Type of handling asymmetric similarity matrix
```

(continues on next page)



(continued from previous page)

```

--is_draw IS_DRAW      (default: max)
                        Show the existing hyperboxes during the training
                        process on the screen (default: False)

```

### Create the path to training and testing datasets stored in the dataset folder

```
[7]: training_data_file = os.path.join(project_dir, Path("dataset/syn_num_train.csv"))
      training_data_file
```

```
[7]: 'C:\\hyperbox-brain\\dataset\\syn_num_train.csv'
```

```
[8]: testing_data_file = os.path.join(project_dir, Path("dataset/syn_num_test.csv"))
      testing_data_file
```

```
[8]: 'C:\\hyperbox-brain\\dataset\\syn_num_test.csv'
```

### Run a demo program

```
[9]: !python "{agglo_sm_file_path}" -training_file "{training_data_file}" -testing_file "
      ↪{testing_data_file}" --theta 0.1 --min_simil 0.5 --simil_measure "short" --asimil_type
      ↪"max" --gamma 1
```

```
Number of hyperboxes = 63
```

```
Testing accuracy (using a probability measure for samples on the boundary) = 85.10%
```

```
Testing accuracy (using a Manhattan distance for samples on the boundary) = 85.10%
```

## 2. Using the GFMM classifier with the agglomerative learning algorithm with full similarity matrix through its init, fit, and predict functions

```
[10]: from hbbrain.numerical_data.batch_learner.agglo_gfmm import AgglomerativeLearningGFMM
      import pandas as pd
```

### Create training and testing data sets

```
[11]: df_train = pd.read_csv(training_data_file, header=None)
      df_test = pd.read_csv(testing_data_file, header=None)
```

```
Xy_train = df_train.to_numpy()
```

```
Xy_test = df_test.to_numpy()
```

```
Xtr = Xy_train[:, :-1]
```

```
ytr = Xy_train[:, -1]
```

```
Xtest = Xy_test[:, :-1]
```

```
ytest = Xy_test[:, -1]
```

## Initializing parameters

```
[12]: theta = 0.1
      min_simil = 0.5
      simil_measure = 'short'
      asimil_type = 'max'
      gamma = 1
      is_draw = True
```

## Training

```
[13]: aggro_gfmm_clf = AgglomerativeLearningGFMM(theta=theta, min_simil=min_simil, simil_
      ↪measure=simil_measure, asimil_type=asimil_type, gamma=gamma, is_draw=is_draw)
      aggro_gfmm_clf.fit(Xtr, ytr)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[13]: AgglomerativeLearningGFMM(is_draw=True, simil_measure='short', theta=0.1)
```

The code below shows how to display decision boundaries among classes if input data are 2-dimensional

```
[14]: aggro_gfmm_clf.draw_hyperbox_and_boundary("The trained GFMM classifier and its decision_
      ↪boundaries")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[15]: print("Number of existing hyperboxes = %d"%(aggro_gfmm_clf.get_n_hyperboxes()))
```

```
Number of existing hyperboxes = 63
```

```
[16]: print("Training time = %f (s)"%aggro_gfmm_clf.elapsed_training_time)
```

```
Training time = 39.946154 (s)
```

## Prediction

```
[17]: from sklearn.metrics import accuracy_score
      from hbbrain.constants import MANHATTAN_DIS
```

```
[18]: y_pred = aggro_gfmm_clf.predict(Xtest)
      acc = accuracy_score(ytest, y_pred)
      print(f'Accuracy (using a probability measure for samples on the boundary) = {acc * 100:
      ↪.2f}%')
```

```
Accuracy (using a probability measure for samples on the boundary) = 85.10%
```

```
[19]: y_pred = agglo_gfmm_clf.predict(Xtest, MANHATTAN_DIS)
acc = accuracy_score(ytest, y_pred)
print(f'Accuracy (using a Manhattan distance for samples on the boundary) = {acc * 100: .
↪2f}%')
```

Accuracy (using a Manhattan distance for samples on the boundary) = 85.10%

**Explaining the predicted result for the input sample by showing membership values and hyperboxes for each class**

```
[20]: sample_need_explain = 10
y_pred_input_0, mem_val_classes, min_points_classes, max_points_classes = agglo_gfmm_clf.
↪get_sample_explanation(Xtest[sample_need_explain], Xtest[sample_need_explain])
```

```
[21]: print("Predicted class for sample X = [%f, %f] is %d and real class is %d" %_
↪(Xtest[sample_need_explain, 0], Xtest[sample_need_explain, 1], y_pred_input_0, _
↪ytest[sample_need_explain]))
```

Predicted class for sample X = [0.571640, 0.233700] is 2 and real class is 2

```
[22]: print("Membership values:")
for key, val in mem_val_classes.items():
    print("Class %d has the maximum membership value = %f" % (key, val))

for key in min_points_classes:
    print("Class %d has the representative hyperbox: V = %s and W = %s" % (key, min_
↪points_classes[key], max_points_classes[key]))
```

Membership values:  
Class 1 has the maximum membership value = 0.870180  
Class 2 has the maximum membership value = 0.961410  
Class 1 has the representative hyperbox: V = [0.66562 0.36352] and W = [0.66562 0.36352]  
Class 2 has the representative hyperbox: V = [0.57285 0.27229] and W = [0.57285 0.27229]

**Show input sample and hyperboxes belonging to each class. In 2D, we can show rectangles or use parallel coordinates**

**Using rectangles to show explanations**

```
[23]: agglo_gfmm_clf.show_sample_explanation(Xtest[sample_need_explain], Xtest[sample_need_
↪explain], min_points_classes, max_points_classes, y_pred_input_0, "2D")
```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

**Using parallel coordinates. This mode best fits for any dimensions**

```
[24]: # Create a parallel coordinates graph
agglo_gfmm_clf.show_sample_explanation(Xtest[sample_need_explain], Xtest[sample_need_
↪explain], min_points_classes, max_points_classes, y_pred_input_0, file_path="par_cord/
↪agglo_gfmm_par_cord.html")

[25]: # Load parallel coordinates to display on the notebook
from IPython.display import IFrame
# We load the parallel coordinates from GitHub here for demonstration in readthedocs
# On the local notebook, we only need to load from the graph storing at 'par_cord/agglo_
↪gfmm_par_cord.html'
IFrame('https://uts-caslab.github.io/hyperbox-brain/docs/tutorials/par_cord/agglo_gfmm_
↪par_cord.html', width=820, height=520)

[25]: <IPython.lib.display.IFrame at 0x2a509965898>
```

**Accelerated Agglomerative Learning Algorithm for GFMM**

This example shows how to use the GFMM classifier using an accelerated agglomerative learning algorithm (AGGLO-2)

Note that the numerical features in training and testing datasets must be in the range of [0, 1] because the AGGLO-2-GFMM classifiers require features in the unit cube.

**1. Execute directly from the python file**

```
[1]: %matplotlib notebook

[2]: import os
import warnings
warnings.filterwarnings('ignore')
```

**Get the path to the this jupyter notebook file**

```
[3]: this_notebook_dir = os.path.dirname(os.path.abspath("__file__"))
this_notebook_dir

[3]: 'C:\\hyperbox-brain\\docs\\tutorials'
```

## Get the home folder of the Hyperbox-Brain project

```
[4]: from pathlib import Path
project_dir = Path(this_notebook_dir).parent.parent
project_dir

[4]: WindowsPath('C:/hyperbox-brain')
```

## Create the path to the Python file containing the implementation of the GFMM classifier using the accelerated agglomerative learning algorithm

```
[5]: accel_agglo_file_path = os.path.join(project_dir, Path("hbbrain/numerical_data/batch_
↳ learner/accel_agglo_gfmm.py"))
accel_agglo_file_path

[5]: 'C:\\hyperbox-brain\\hbbrain\\numerical_data\\batch_learner\\accel_agglo_gfmm.py'
```

## Run the found file by showing the execution directions

```
[6]: !python "{accel_agglo_file_path}" -h

usage: accel_agglo_gfmm.py [-h] -training_file TRAINING_FILE -testing_file
                        TESTING_FILE [--theta THETA] [--gamma GAMMA]
                        [--min_simil MIN_SIMIL]
                        [--simil_measure {mid,long,short}]
                        [--asimil_type {min,max}] [--is_draw IS_DRAW]

The description of parameters

required arguments:
  -training_file TRAINING_FILE
                        A required argument for the path to training data file
                        (including file name)
  -testing_file TESTING_FILE
                        A required argument for the path to testing data file
                        (including file name)

optional arguments:
  --theta THETA         Maximum hyperbox size (in the range of (0, 1])
                        (default: 0.5)
  --gamma GAMMA         A sensitivity parameter describing the speed of
                        decreasing of the membership function in each
                        dimension (larger than 0) (default: 1)
  --min_simil MIN_SIMIL
                        Mimimum similarity value so that two hyperboxes can be
                        merged (in the range of [0, 1])(default: 0.5)
  --simil_measure {mid,long,short}
                        Type of similarity measure (default: mid)
  --asimil_type {min,max}
                        Type of handling asymmetric similarity matrix
```

(continues on next page)

(continued from previous page)

```

--is_draw IS_DRAW      (default: max)
                        Show the existing hyperboxes during the training
                        process on the screen (default: False)

```

### Create the path to training and testing datasets stored in the dataset folder

```
[7]: training_data_file = os.path.join(project_dir, Path("dataset/syn_num_train.csv"))
      training_data_file
```

```
[7]: 'C:\\hyperbox-brain\\dataset\\syn_num_train.csv'
```

```
[8]: testing_data_file = os.path.join(project_dir, Path("dataset/syn_num_test.csv"))
      testing_data_file
```

```
[8]: 'C:\\hyperbox-brain\\dataset\\syn_num_test.csv'
```

### Run a demo program

```
[9]: !python "{accel_agglo_file_path}" -training_file "{training_data_file}" -testing_file "
      ↪ "{testing_data_file}" --theta 0.1 --min_simil 0.5 --simil_measure "short" --asimil_type
      ↪ "max" --gamma 1
```

```
Number of hyperboxes = 61
```

```
Testing accuracy (using a probability measure for samples on the boundary) = 85.00%
```

```
Testing accuracy (using a Manhattan distance for samples on the boundary) = 85.00%
```

## 2. Using the GFMM classifier with the accelerated agglomerative learning algorithm through its init, fit, and predict functions

```
[10]: from hbbrain.numerical_data.batch_learner.accel_agglo_gfmm import _
      ↪ AccelAgglomerativeLearningGFMM
      import pandas as pd
```

### Create training and testing data sets

```
[11]: df_train = pd.read_csv(training_data_file, header=None)
      df_test = pd.read_csv(testing_data_file, header=None)
```

```
Xy_train = df_train.to_numpy()
```

```
Xy_test = df_test.to_numpy()
```

```
Xtr = Xy_train[:, :-1]
```

```
ytr = Xy_train[:, -1]
```

```
Xtest = Xy_test[:, :-1]
```

```
ytest = Xy_test[:, -1]
```

## Initializing parameters

```
[12]: theta = 0.1
      min_simil = 0.5
      simil_measure = 'short'
      asimil_type = 'max'
      gamma = 1
      is_draw = True
```

## Training

```
[13]: accel_agglo_gfmm_clf = AccelAgglomerativeLearningGFMM(theta=theta, min_simil=min_simil,
      ↪simil_measure=simil_measure, asimil_type=asimil_type, gamma=gamma, is_draw=is_draw)
      accel_agglo_gfmm_clf.fit(Xtr, ytr)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[13]: AccelAgglomerativeLearningGFMM(is_draw=True, simil_measure='short', theta=0.1)
```

The code below shows how to display decision boundaries among classes if input data are 2-dimensional

```
[14]: accel_agglo_gfmm_clf.draw_hyperbox_and_boundary("The trained GFMM classifier and its
      ↪decision boundaries")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[15]: print("Number of existing hyperboxes = %d"%(accel_agglo_gfmm_clf.get_n_hyperboxes()))
```

```
Number of existing hyperboxes = 61
```

```
[16]: print("Training time = %f (s)"%accel_agglo_gfmm_clf.elapsed_training_time)
```

```
Training time = 3.239400 (s)
```

## Prediction

```
[17]: from sklearn.metrics import accuracy_score
      from hbbrain.constants import MANHATTAN_DIS
```

```
[18]: y_pred = accel_agglo_gfmm_clf.predict(Xtest)
      acc = accuracy_score(ytest, y_pred)
      print(f'Accuracy (using a probability measure for samples on the boundary) = {acc * 100:
      ↪.2f}%')
```

```
Accuracy (using a probability measure for samples on the boundary) = 85.00%
```

```
[19]: y_pred = accel_agglo_gfmm_clf.predict(Xtest, MANHATTAN_DIS)
acc = accuracy_score(ytest, y_pred)
print(f'Accuracy (using a Manhattan distance for samples on the boundary) = {acc * 100: .
↪2f}%')
```

Accuracy (using a Manhattan distance for samples on the boundary) = 85.00%

**Explaining the predicted result for the input sample by showing membership values and hyperboxes for each class**

```
[20]: sample_need_explain = 10
y_pred_input_0, mem_val_classes, min_points_classes, max_points_classes = accel_agglo_
↪gfmm_clf.get_sample_explanation(Xtest[sample_need_explain], Xtest[sample_need_explain])
```

```
[21]: print("Predicted class for sample X = [%f, %f] is %d and real class is %d" %_
↪(Xtest[sample_need_explain, 0], Xtest[sample_need_explain, 1], y_pred_input_0, _
↪ytest[sample_need_explain]))
```

Predicted class for sample X = [0.571640, 0.233700] is 2 and real class is 2

```
[22]: print("Membership values:")
for key, val in mem_val_classes.items():
    print("Class %d has the maximum membership value = %f" % (key, val))

for key in min_points_classes:
    print("Class %d has the representative hyperbox: V = %s and W = %s" % (key, min_
↪points_classes[key], max_points_classes[key]))
```

Membership values:  
Class 1 has the maximum membership value = 0.870180  
Class 2 has the maximum membership value = 0.961410  
Class 1 has the representative hyperbox: V = [0.66562 0.36352] and W = [0.66562 0.36352]  
Class 2 has the representative hyperbox: V = [0.57285 0.27229] and W = [0.61106 0.28476]

**Show input sample and hyperboxes belonging to each class. In 2D, we can show rectangles or use parallel coordinates**

**Using rectangles to show explanations**

```
[23]: accel_agglo_gfmm_clf.show_sample_explanation(Xtest[sample_need_explain], Xtest[sample_
↪need_explain], min_points_classes, max_points_classes, y_pred_input_0, "2D")
```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>



### Using parallel coordinates. This mode best fits for any dimensions

```
[24]: # Create a parallel coordinates graph
accel_agglo_gfmm_clf.show_sample_explanation(Xtest[sample_need_explain], Xtest[sample_
↪need_explain], min_points_classes, max_points_classes, y_pred_input_0, file_path="par_
↪cord/accel_agglo_gfmm_par_cord.html")

[25]: # Load parallel coordinates to display on the notebook
from IPython.display import IFrame
# We load the parallel coordinates from GitHub here for demonstration in readthedocs
# On the local notebook, we only need to load from the graph storing at 'par_cord/accel_
↪agglo_gfmm_par_cord.html'
IFrame('https://uts-caslab.github.io/hyperbox-brain/docs/tutorials/par_cord/accel_agglo_
↪gfmm_par_cord.html', width=820, height=520)

[25]: <IPython.lib.display.IFrame at 0x2b2f6665a90>
```

## 2.8.2 Incremental learners

### Original Online Learning Algorithm for GFMM

This example shows how to use the GFMM classifier using an original online learning algorithm.

Note that the numerical features in training and testing datasets must be in the range of [0, 1] because the GFMM classifiers require features in the unit cube.

#### 1. Execute directly from the python file

```
[1]: %matplotlib notebook
```

```
[2]: import os
import warnings
warnings.filterwarnings('ignore')
```

#### Get the path to the this jupyter notebook file

```
[3]: this_notebook_dir = os.path.dirname(os.path.abspath("__file__"))
this_notebook_dir
```

```
[3]: 'C:\\hyperbox-brain\\docs\\tutorials'
```

### Get the home folder of the Hyperbox-Brain project

```
[4]: from pathlib import Path
project_dir = Path(this_notebook_dir).parent.parent
project_dir

[4]: WindowsPath('C:/hyperbox-brain')
```

### Create the path to the Python file containing the implementation of the GFMM classifier using the original online learning algorithm

```
[5]: onln_gfmm_file_path = os.path.join(project_dir, Path("hbbrain/numerical_data/incremental_
↳ learner/onln_gfmm.py"))
onln_gfmm_file_path

[5]: 'C:\\hyperbox-brain\\hbbrain\\numerical_data\\incremental_learner\\onln_gfmm.py'
```

### Run the found file by showing the execution directions

```
[6]: !python "{onln_gfmm_file_path}" -h

usage: onln_gfmm.py [-h] -training_file TRAINING_FILE -testing_file
                    TESTING_FILE [--theta THETA] [--theta_min THETA_MIN]
                    [--gamma GAMMA] [--alpha ALPHA] [--is_draw IS_DRAW]

The description of parameters

required arguments:
  -training_file TRAINING_FILE
                        A required argument for the path to training data file
                        (including file name)
  -testing_file TESTING_FILE
                        A required argument for the path to testing data file
                        (including file name)

optional arguments:
  --theta THETA          Maximum hyperbox size (in the range of (0, 1])
                        (default: 0.5)
  --theta_min THETA_MIN  Minimum value of the maximum hyperbox size to escape
                        the training loop (in the range of (0, 1]) (default:
                        0.5)
  --gamma GAMMA          A sensitivity parameter describing the speed of
                        decreasing of the membership function in each
                        dimension (larger than 0) (default: 1)
  --alpha ALPHA          Multiplier showing the decrease of theta in each step
                        (default: 0.9)
  --is_draw IS_DRAW      Show the existing hyperboxes during the training
                        process on the screen (default: False)
```

**Create the path to training and testing datasets stored in the dataset folder**

```
[7]: training_data_file = os.path.join(project_dir, Path("dataset/syn_num_train.csv"))
      training_data_file
```

```
[7]: 'C:\\hyperbox-brain\\dataset\\syn_num_train.csv'
```

```
[8]: testing_data_file = os.path.join(project_dir, Path("dataset/syn_num_test.csv"))
      testing_data_file
```

```
[8]: 'C:\\hyperbox-brain\\dataset\\syn_num_test.csv'
```

**Run a demo program**

```
[9]: !python "{onln_gfmm_file_path}" -training_file "{training_data_file}" -testing_file "
      ↪{testing_data_file}" --theta 0.1 --theta_min 0.1 --gamma 1
```

```
Number of hyperboxes = 53
Testing accuracy = 84.50%
```

**2. Using the GFMM classifier with original online learning algorithm through its init, fit, and predict functions**

```
[10]: from hbbrain.numerical_data.incremental_learner.onln_gfmm import OnlineGFMM
```

```
[11]: import pandas as pd
```

**Create training and testing data sets**

```
[12]: df_train = pd.read_csv(training_data_file, header=None)
      df_test = pd.read_csv(testing_data_file, header=None)
```

```
Xy_train = df_train.to_numpy()
Xy_test = df_test.to_numpy()
```

```
Xtr = Xy_train[:, :-1]
ytr = Xy_train[:, -1]
```

```
Xtest = Xy_test[:, :-1]
ytest = Xy_test[:, -1]
```

## Initializing parameters

```
[13]: theta = 0.1
      theta_min = 0.1
      gamma = 1
      is_draw = True
```

## Training

```
[14]: onln_gfmm_clf = OnlineGFMM(theta=theta, theta_min=theta_min, gamma=gamma, is_draw=is_
      ↪draw)
      onln_gfmm_clf.fit(Xtr, ytr)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[14]: OnlineGFMM(C=array([1, 2, 1, 1, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 2, 1,
      1, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 2, 2, 1, 1, 2, 1, 2, 1, 2, 1, 1,
      2, 2, 1, 2, 2, 2, 2, 2, 1])),
      V=array([[0.42413    , 0.53516    ],
      [0.70577    , 0.397105   ],
      [0.82785    , 0.78025    ],
      [0.66038    , 0.51128    ],
      [0.48794    , 0.672     ],
      [0.26651    , 0.18424    ],
      [0.32289    , 0.60194    ],
      [0.19944    , 0.03      ],
      [0.29343    , 0.28975    ],
      [0.63683    , 0.6936     ],
      [0.32906    , 0.55512    ],
      [0.03       , 0.47757    ],
      [0.54...
      [0.815     , 0.397095   ],
      [0.67906    , 0.83605    ],
      [0.37033    , 0.26124    ],
      [0.52197    , 0.91371    ],
      [0.66037    , 0.57837    ],
      [0.52621    , 0.66846    ],
      [0.80583    , 0.43242    ],
      [0.79935    , 0.7757     ],
      [0.35813    , 0.58772    ],
      [0.79516    , 0.32629    ],
      [0.70743    , 0.50325    ],
      [0.36057    , 0.71561    ],
      [0.72496    , 0.38674    ],
      [0.28822    , 0.62174    ],
      [0.14737    , 0.28498    ],
      [0.56487    , 0.17003    ],
      [0.68469    , 0.2221     ],
      [0.55763    , 0.43813    ]]),
      is_draw=True, theta=0.1, theta_min=0.1)
```

The code below shows how to display decision boundaries among classes if input data are 2-dimensional

```
[15]: onln_gfmm_clf.draw_hyperbox_and_boundary("The trained GFMM classifier and its decision_
↳ boundaries")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[16]: print("Number of existing hyperboxes = %d"%(onln_gfmm_clf.get_n_hyperboxes()))
```

```
Number of existing hyperboxes = 53
```

### Prediction

```
[17]: from sklearn.metrics import accuracy_score
```

```
[18]: y_pred = onln_gfmm_clf.predict(Xtest)
acc = accuracy_score(ytest, y_pred)
print(f'Accuracy = {acc * 100: .2f}%')
```

```
Accuracy = 84.50%
```

Explaining the predicted result for the input sample by showing membership values and hyperboxes for each class

```
[19]: sample_need_explain = 10
y_pred_input_0, mem_val_classes, min_points_classes, max_points_classes = onln_gfmm_clf.
↳ get_sample_explanation(Xtest[sample_need_explain], Xtest[sample_need_explain])
```

```
[20]: print("Predicted class for sample X = [%f, %f] is %d and real class is %d" %_
↳ (Xtest[sample_need_explain, 0], Xtest[sample_need_explain, 1], y_pred_input_0, _
↳ ytest[sample_need_explain]))
```

```
Predicted class for sample X = [0.571640, 0.233700] is 2 and real class is 2
```

```
[21]: print("Membership values:")
for key, val in mem_val_classes.items():
    print("Class %d has the maximum membership value = %f" % (key, val))

for key in min_points_classes:
    print("Class %d has the representative hyperbox: V = %s and W = %s" % (key, min_
↳ points_classes[key], max_points_classes[key]))
```

```
Membership values:
```

```
Class 1 has the maximum membership value = 0.870180
```

```
Class 2 has the maximum membership value = 0.984660
```

```
Class 1 has the representative hyperbox: V = [0.58339 0.36352] and W = [0.66562 0.
↳ 38375437]
```

```
Class 2 has the representative hyperbox: V = [0.57285 0.24904] and W = [0.65695 0.31638]
```

Show input sample and hyperboxes belonging to each class. In 2D, we can show rectangles or use parallel coordinates

### Using rectangles to show explanations

```
[22]: onln_gfmm_clf.show_sample_explanation(Xtest[sample_need_explain], Xtest[sample_need_
      ↪explain], min_points_classes, max_points_classes, y_pred_input_0, "2D")

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>
```

### Using parallel coordinates. This mode best fits for any dimensions

```
[23]: # Create a parallel coordinates graph
      onln_gfmm_clf.show_sample_explanation(Xtest[sample_need_explain], Xtest[sample_need_
      ↪explain], min_points_classes, max_points_classes, y_pred_input_0, file_path="par_cord/
      ↪onln_gfmm_par_cord_1.html")

[24]: # Load parallel coordinates to display on the notebook
      from IPython.display import IFrame
      # We load the parallel coordinates from GitHub here for demonstration in readthedocs
      # On the local notebook, we only need to load from the graph storing at 'par_cord/onln_
      ↪gfmm_par_cord_1.html'
      IFrame('https://uts-caslab.github.io/hyperbox-brain/docs/tutorials/par_cord/onln_gfmm_
      ↪par_cord_1.html', width=820, height=520)

[24]: <IPython.lib.display.IFrame at 0x1e6eeeab9b0>
```

### An example for the wrong prediction case with the demonstration

```
[25]: # Try another sample
      sample_need_explain = 1
      y_pred_input_0, mem_val_classes, min_points_classes, max_points_classes = onln_gfmm_clf.
      ↪get_sample_explanation(Xtest[sample_need_explain], Xtest[sample_need_explain])

[26]: print("Predicted class for sample X = [%f, %f] is %d and real class is %d" %
      ↪(Xtest[sample_need_explain, 0], Xtest[sample_need_explain, 1], y_pred_input_0,
      ↪ytest[sample_need_explain]))

      Predicted class for sample X = [0.485900, 0.476500] is 2 and real class is 1

[27]: print("Membership values:")
      for key, val in mem_val_classes.items():
          print("Class %d has the maximum membership value = %f" % (key, val))
      print("Hyperboxes:")
      for key in min_points_classes:
          print("Class %d has the representative hyperbox: V = %s and W = %s" % (key, min_
          ↪points_classes[key], max_points_classes[key]))
```

```
Membership values:
Class 1 has the maximum membership value = 0.941340
Class 2 has the maximum membership value = 0.955140
Hyperboxes:
Class 1 has the representative hyperbox: V = [0.42413 0.53516] and W = [0.52084 0.60972]
Class 2 has the representative hyperbox: V = [0.49255 0.39125] and W = [0.52124 0.43164]
```

```
[28]: onln_gfmm_clf.show_sample_explanation(Xtest[sample_need_explain], Xtest[sample_need_
      ↪explain], min_points_classes, max_points_classes, y_pred_input_0, "2D")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

### Using parallel coordinates. This mode best fits for any dimensions

```
[29]: # Create a parallel coordinates graph
onln_gfmm_clf.show_sample_explanation(Xtest[sample_need_explain], Xtest[sample_need_
      ↪explain], min_points_classes, max_points_classes, y_pred_input_0, file_path="par_cord/
      ↪onln_gfmm_par_cord_2.html")
```

```
[30]: # Load parallel coordinates to display on the notebook
from IPython.display import IFrame
# We load the parallel coordinates from GitHub here for demonstration in readthedocs
# On the local notebook, we only need to load from the graph storing at 'par_cord/onln_
      ↪gfmm_par_cord_2.html'
IFrame('https://uts-caslab.github.io/hyperbox-brain/docs/tutorials/par_cord/onln_gfmm_
      ↪par_cord_2.html', width=820, height=520)
```

```
[30]: <IPython.lib.display.IFrame at 0x1e6ee6da4e0>
```

## Improved Online Learning Algorithm for GFMM

This example shows how to use the GFMM classifier using an improved online learning algorithm (IOL-GFMM)

Note that the numerical features in training and testing datasets must be in the range of [0, 1] because the GFMM classifiers require features in the unit cube.

### 1. Execute directly from the python file

```
[1]: %matplotlib notebook
```

```
[2]: import os
import warnings
warnings.filterwarnings('ignore')
```

### Get the path to the this jupyter notebook file

```
[3]: this_notebook_dir = os.path.dirname(os.path.abspath("__file__"))
    this_notebook_dir
```

```
[3]: 'C:\\hyperbox-brain\\docs\\tutorials'
```

### Get the home folder of the Hyperbox-Brain project

```
[4]: from pathlib import Path
    project_dir = Path(this_notebook_dir).parent.parent
    project_dir
```

```
[4]: WindowsPath('C:/hyperbox-brain')
```

### Create the path to the Python file containing the implementation of the GFMM classifier using the improved online learning algorithm

```
[5]: iol_gfmm_file_path = os.path.join(project_dir, Path("hbbrain/numerical_data/incremental_
    ↪ learner/iol_gfmm.py"))
    iol_gfmm_file_path
```

```
[5]: 'C:\\hyperbox-brain\\hbbrain\\numerical_data\\incremental_learner\\iol_gfmm.py'
```

### Run the found file by showing the execution directions

```
[6]: !python "{iol_gfmm_file_path}" -h
```

```
usage: iol_gfmm.py [-h] -training_file TRAINING_FILE -testing_file
                  TESTING_FILE [--theta THETA] [--gamma GAMMA]
                  [--is_draw IS_DRAW]
```

The description of parameters

required arguments:

- training\_file TRAINING\_FILE  
A required argument for the path to training data file (including file name)
- testing\_file TESTING\_FILE  
A required argument for the path to testing data file (including file name)

optional arguments:

- theta THETA  
Maximum hyperbox size (in the range of (0, 1]) (default: 0.5)
- gamma GAMMA  
A sensitivity parameter describing the speed of decreasing of the membership function in each dimension (larger than 0) (default: 1)
- is\_draw IS\_DRAW  
Show the existing hyperboxes during the training process on the screen (default: False)



**Create the path to training and testing datasets stored in the dataset folder**

```
[7]: training_data_file = os.path.join(project_dir, Path("dataset/syn_num_train.csv"))
      training_data_file
```

```
[7]: 'C:\\hyperbox-brain\\dataset\\syn_num_train.csv'
```

```
[8]: testing_data_file = os.path.join(project_dir, Path("dataset/syn_num_test.csv"))
      testing_data_file
```

```
[8]: 'C:\\hyperbox-brain\\dataset\\syn_num_test.csv'
```

**Run a demo program**

```
[9]: !python "{iol_gfmm_file_path}" -training_file "{training_data_file}" -testing_file "
      ↪{testing_data_file}" --theta 0.1 --gamma 1
```

Number of hyperboxes = 68

Testing accuracy (using a probability measure for samples on the boundary) = 87.20%

**2. Using the GFMM classifier with IOL-GFMM algorithm through its init, fit, and predict functions**

```
[10]: from hbbrain.numerical_data.incremental_learner.iol_gfmm import ImprovedOnlineGFMM
      import pandas as pd
```

**Create training and testing data sets**

```
[11]: df_train = pd.read_csv(training_data_file, header=None)
      df_test = pd.read_csv(testing_data_file, header=None)
```

```
Xy_train = df_train.to_numpy()
Xy_test = df_test.to_numpy()
```

```
Xtr = Xy_train[:, :-1]
ytr = Xy_train[:, -1]
```

```
Xtest = Xy_test[:, :-1]
ytest = Xy_test[:, -1]
```

## Initializing parameters

```
[12]: theta = 0.1
      gamma = 1
      is_draw = True
```

## Training

```
[13]: iol_gfmm_clf = ImprovedOnlineGFMM(theta=theta, gamma=gamma, is_draw=is_draw)
      iol_gfmm_clf.fit(Xtr, ytr)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[13]: ImprovedOnlineGFMM(C=array([1, 2, 1, 1, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 2, 2, 2, 1,
↪ 2, 1,
      2, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 2, 2, 1, 1, 1, 2, 1, 2,
      1, 2, 2, 1, 2, 1, 1, 2, 2, 1, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2,
      2, 2]),
      N_samples=array([11, 3, 2, 10, 5, 6, 6, 2, 7, 6, 3, 1, 7,
↪ 6, 2, 11, 1,
      5, 7, 2, 3, 9, 3, 4, 6, 9, 10, 5, 8, 13, 4, 4, 3, 3,
      6, 4, 2, 1, 2, 1, 1, 1, 2, 1, 2, 1, 1, 2, 1, 1, 1,
      5, 2, 1, 1, 6, 1, 5, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1]),
      V=array([[0.42413, 0.5351...
      [0.52621 , 0.59493 ],
      [0.79935 , 0.7757  ],
      [0.6248  , 0.39922 ],
      [0.79516 , 0.32629 ],
      [0.66562 , 0.36352 ],
      [0.36057 , 0.71561 ],
      [0.72496 , 0.38674 ],
      [0.70743 , 0.50325 ],
      [0.40233 , 0.67232 ],
      [0.28822 , 0.62174 ],
      [0.14737 , 0.28498 ],
      [0.75421 , 0.40498 ],
      [0.59655 , 0.56029 ],
      [0.91185 , 0.48697 ],
      [0.6504  , 0.51624 ],
      [0.68853 , 0.41466 ],
      [0.56487 , 0.17003 ],
      [0.59235 , 0.54123 ],
      [0.68469 , 0.2221  ]]),
      is_draw=True, theta=0.1)
```

The code below shows how to display decision boundaries among classes if input data are 2-dimensional

```
[14]: iol_gfmm_clf.draw_hyperbox_and_boundary("The trained GFMM classifier using IOL-GFMM and
↳ its decision boundaries")

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>
```

```
[15]: print("Number of existing hyperboxes = %d"%(iol_gfmm_clf.get_n_hyperboxes()))

Number of existing hyperboxes = 68
```

### Prediction

```
[16]: from sklearn.metrics import accuracy_score
```

Predict the class label for input samples using a probability measure based on the number of samples included inside the winner hyperboxes for the samples located on the decision boundaries

```
[17]: y_pred = iol_gfmm_clf.predict(Xtest)
acc = accuracy_score(ytest, y_pred)
print(f'Accuracy = {acc * 100: .2f}%')

Accuracy = 87.20%
```

Predict the class label for input samples using Manhattan distance measure for the samples located on the decision boundaries

```
[18]: from hbbrain.constants import MANHATTAN_DIS
y_pred = iol_gfmm_clf.predict(Xtest, MANHATTAN_DIS)
acc = accuracy_score(ytest, y_pred)
print(f'Accuracy (Manhattan distance for samples on the decision boundaries) = {acc *
↳ 100: .2f}%')

Accuracy (Manhattan distance for samples on the decision boundaries) = 87.20%
```

Explaining the predicted result for the input sample by showing membership values and hyperboxes for each class

```
[19]: sample_need_explain = 10
y_pred_input_0, mem_val_classes, min_points_classes, max_points_classes = iol_gfmm_clf.
↳ get_sample_explanation(Xtest[sample_need_explain], Xtest[sample_need_explain])
```

```
[20]: print("Predicted class for sample X = [%f, %f] is %d and real class is %d" %
↳ (Xtest[sample_need_explain, 0], Xtest[sample_need_explain, 1], y_pred_input_0,
↳ ytest[sample_need_explain]))
```

Predicted class for sample X = [0.571640, 0.233700] is 2 and real class is 2

```
[21]: print("Membership values:")
      for key, val in mem_val_classes.items():
          print("Class %d has the maximum membership value = %f" % (key, val))

      for key in min_points_classes:
          print("Class %d has the representative hyperbox: V = %s and W = %s" % (key, min_
            ↪points_classes[key], max_points_classes[key]))
```

Membership values:  
 Class 1 has the maximum membership value = 0.870180  
 Class 2 has the maximum membership value = 0.984660  
 Class 1 has the representative hyperbox: V = [0.66562 0.36352] and W = [0.66562 0.36352]  
 Class 2 has the representative hyperbox: V = [0.57285 0.24904] and W = [0.65695 0.31638]

**Show input sample and hyperboxes belonging to each class. In 2D, we can show rectangles or use parallel coordinates. For 4 or more dimensions, parallel coordinates should be used**

**Using rectangles to show explanations**

```
[22]: iol_gfmm_clf.show_sample_explanation(Xtest[sample_need_explain], Xtest[sample_need_
      ↪explain], min_points_classes, max_points_classes, y_pred_input_0, "2D")
```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

**Using parallel coordinates. This mode best fits for any dimensions**

```
[23]: # Create a parallel coordinates graph
      iol_gfmm_clf.show_sample_explanation(Xtest[sample_need_explain], Xtest[sample_need_
      ↪explain], min_points_classes, max_points_classes, y_pred_input_0, file_path="par_cord/
      ↪iol_gfmm_par_cord.html")
```

```
[24]: # Load parallel coordinates to display on the notebook
      from IPython.display import IFrame
      # We load the parallel coordinates from GitHub here for demonstration in readthedocs
      # On the local notebook, we only need to load from the graph storing at 'par_cord/iol_
      ↪gfmm_par_cord.html'
      IFram('https://uts-caslab.github.io/hyperbox-brain/docs/tutorials/par_cord/iol_gfmm_par_
      ↪cord.html', width=820, height=520)
```

```
[24]: <IPython.lib.display.IFrame at 0x23beb125e10>
```

## Fuzzy Min-Max Neural Network with Original Online Learning Algorithm

This example shows how to use Simpson's fuzzy min-max neural network with the original online learning algorithm (FMNN)

Note that the numerical features in training and testing datasets must be in the range of [0, 1] because the FMNN classifiers require features in the unit cube.

### 1. Execute directly from the python file

```
[1]: %matplotlib notebook
```

```
[2]: import os
import warnings
warnings.filterwarnings('ignore')
```

### Get the path to the this jupyter notebook file

```
[3]: this_notebook_dir = os.path.dirname(os.path.abspath("__file__"))
this_notebook_dir
```

```
[3]: 'C:\\hyperbox-brain\\docs\\tutorials'
```

### Get the home folder of the Hyperbox-Brain project

```
[4]: from pathlib import Path
project_dir = Path(this_notebook_dir).parent.parent
project_dir
```

```
[4]: WindowsPath('C:/hyperbox-brain')
```

### Create the path to the Python file containing the implementation of the FMNN classifier using the original online learning algorithm

```
[5]: fmnn_file_path = os.path.join(project_dir, Path("hbbrain/numerical_data/incremental_
↳ learner/fmnn.py"))
fmnn_file_path
```

```
[5]: 'C:\\hyperbox-brain\\hbbrain\\numerical_data\\incremental_learner\\fmnn.py'
```

### Run the found file by showing the execution directions

```
[6]: !python "{fmnn_file_path}" -h
```

usage: fmnn.py [-h] -training\_file TRAINING\_FILE -testing\_file TESTING\_FILE  
                  [--theta THETA] [--gamma GAMMA] [--is\_draw IS\_DRAW]

The description of parameters

required arguments:

- training\_file TRAINING\_FILE  
                  A required argument for the path to training data file  
                  (including file name)
- testing\_file TESTING\_FILE  
                  A required argument for the path to testing data file  
                  (including file name)

optional arguments:

- theta THETA       Maximum hyperbox size (in the range of (0, 1])  
                  (default: 0.5)
- gamma GAMMA       A sensitivity parameter describing the speed of  
                  decreasing of the membership function in each  
                  dimension (larger than 0) (default: 1)
- is\_draw IS\_DRAW   Show the existing hyperboxes during the training  
                  process on the screen (default: False)

### Create the path to training and testing datasets stored in the dataset folder

```
[7]: training_data_file = os.path.join(project_dir, Path("dataset/syn_num_train.csv"))  
training_data_file
```

```
[7]: 'C:\\hyperbox-brain\\dataset\\syn_num_train.csv'
```

```
[8]: testing_data_file = os.path.join(project_dir, Path("dataset/syn_num_test.csv"))  
testing_data_file
```

```
[8]: 'C:\\hyperbox-brain\\dataset\\syn_num_test.csv'
```

### Run a demo program

```
[9]: !python "{fmnn_file_path}" -training_file "{training_data_file}" -testing_file "{testing_  
↪data_file}" --theta 0.1 --gamma 1
```

Number of hyperboxes = 91  
Testing accuracy = 85.30%

## 2. Using the Simpson's FMNN classifier through init, fit, and predict functions

```
[10]: from hbbrain.numerical_data.incremental_learner.fmnn import FMNNClassifier
import pandas as pd
```

### Create training and testing data sets

```
[11]: df_train = pd.read_csv(training_data_file, header=None)
df_test = pd.read_csv(testing_data_file, header=None)

Xy_train = df_train.to_numpy()
Xy_test = df_test.to_numpy()

Xtr = Xy_train[:, :-1]
ytr = Xy_train[:, -1]

Xtest = Xy_test[:, :-1]
ytest = Xy_test[:, -1]
```

### Initializing parameters

```
[12]: theta = 0.1
gamma = 1
is_draw = True
```

### Training

```
[13]: fmnn_clf = FMNNClassifier(theta=theta, gamma=gamma, is_draw=is_draw)
fmnn_clf.fit(Xtr, ytr)

<IPython.core.display.Javascript object>
<IPython.core.display.HTML object>

[13]: FMNNClassifier(C=array([1, 2, 1, 1, 1, 2, 1, 2, 1, 2, 2, 2, 1, 1, 2, 1, 2, 1, 2, 2, 2, 2,
2, 2, 1, 2, 2, 1, 1, 2, 2, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 1,
2, 1, 1, 1, 2, 2, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 2,
2, 1, 1, 2, 2, 1, 2, 1, 1, 2, 1, 1, 2, 1, 2, 2, 2, 1, 1, 1, 1, 2,
1, 1, 1]),
V=array([[0.36239 , 0.55942 ],
[0.64082 , 0.43016875],
[0.91059 , 0.82085 ],
[0.65328 , 0.50326 ],
[0.46107 , 0.68306 ],
[0.29812 , 0.18424 ],
[0.33593 , 0.68775 ],
[0.1...,
[0.25621 , 0.62174 ],
[0.42403 , 0.7592 ]],
```

(continues on next page)

(continued from previous page)

```

[0.79157 , 0.59996 ],
[0.72496 , 0.34978 ],
[0.36842 , 0.76576 ],
[0.73681 , 0.71261 ],
[0.66773 , 0.31155 ],
[0.32289 , 0.6747  ],
[0.28077 , 0.27116 ],
[0.61106 , 0.28476 ],
[0.75421 , 0.40498 ],
[0.38038 , 0.67232 ],
[0.36745 , 0.52006 ],
[0.91185 , 0.48697 ],
[0.35813 , 0.58584 ],
[0.25924 , 0.42696 ],
[0.70685 , 0.64383 ],
[0.75047 , 0.6092  ],
[0.72842 , 0.61048 ]]),
    is_draw=True, theta=0.1)

```

The code below shows how to display decision boundaries among classes if input data are 2-dimensional

```
[14]: fmnn_clf.draw_hyperbox_and_boundary("The trained Simpson's FMNN classifier and its_
      ↪decision boundaries")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[15]: print("Number of existing hyperboxes = %d"%(fmnn_clf.get_n_hyperboxes()))
```

```
Number of existing hyperboxes = 91
```

## Predicting

```
[16]: from sklearn.metrics import accuracy_score
```

```
[17]: y_pred = fmnn_clf.predict(Xtest)
      acc = accuracy_score(ytest, y_pred)
      print(f'Accuracy = {acc * 100: .2f}%')
```

```
Accuracy = 85.30%
```



## Explaining the predicted result for the input sample by showing membership values and hyperboxes for each class

```
[18]: sample_need_explain = 10
y_pred_input_0, mem_val_classes, min_points_classes, max_points_classes = fmn_clf.get_
↳ sample_explanation(Xtest[sample_need_explain])
```

```
[19]: print("Predicted class for sample X = [%f, %f] is %d and real class is %d" %
↳ (Xtest[sample_need_explain, 0], Xtest[sample_need_explain, 1], y_pred_input_0,
↳ ytest[sample_need_explain]))
```

Predicted class for sample X = [0.571640, 0.233700] is 2 and real class is 2

```
[20]: print("Membership values:")
for key, val in mem_val_classes.items():
    print("Class %d has the maximum membership value = %f" % (key, val))

for key in min_points_classes:
    print("Class %d has the representative hyperbox: V = %s and W = %s" % (key, min_
↳ points_classes[key], max_points_classes[key]))
```

Membership values:

Class 1 has the maximum membership value = 0.960506

Class 2 has the maximum membership value = 1.000000

Class 1 has the representative hyperbox: V = [0.55763 0.3916775] and W = [0.67326 0.
↳ 43015875]

Class 2 has the representative hyperbox: V = [0.56487 0.17003] and W = [0.57285 0.27229]

## Show input sample and hyperboxes belonging to each class. In 2D, we can show rectangles or use parallel coordinates

### Using rectangles to show explanations

```
[21]: fmn_clf.show_sample_explanation(Xtest[sample_need_explain], Xtest[sample_need_explain],
↳ min_points_classes, max_points_classes, y_pred_input_0, "2D")
```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

### Using parallel coordinates. This mode best fits for any dimensions

```
[22]: # Create a parallel coordinates graph
fmn_clf.show_sample_explanation(Xtest[sample_need_explain], Xtest[sample_need_explain],
↳ min_points_classes, max_points_classes, y_pred_input_0, file_path="par_cord/fmn_par_
↳ cord.html")
```

```
[23]: # Load parallel coordinates to display on the notebook
from IPython.display import IFrame
# We load the parallel coordinates from GitHub here for demonstration in readthedocs
```

(continues on next page)

(continued from previous page)

```
# On the local notebook, we only need to load from the graph storing at 'par_cord/fmnn_
↪par_cord.html'
IFrame('https://uts-caslab.github.io/hyperbox-brain/docs/tutorials/par_cord/fmnn_par_
↪cord.html', width=820, height=520)
```

```
[23]: <IPython.lib.display.IFrame at 0x276832ea828>
```

## Enhanced Online Learning Algorithm for FMNN

This example shows how to use the Simpson's Fuzzy Min-Max Neural Network trained by an enhanced online learning algorithm (EFMNN)

Note that the numerical features in training and testing datasets must be in the range of [0, 1] because the EFMNN classifiers require features in the unit cube.

### 1. Execute directly from the python file

```
[1]: %matplotlib notebook
```

```
[2]: import os
import warnings
warnings.filterwarnings('ignore')
```

### Get the path to the this jupyter notebook file

```
[3]: this_notebook_dir = os.path.dirname(os.path.abspath("__file__"))
this_notebook_dir
```

```
[3]: 'C:\\hyperbox-brain\\docs\\tutorials'
```

### Get the home folder of the Hyperbox-Brain project

```
[4]: from pathlib import Path
project_dir = Path(this_notebook_dir).parent.parent
project_dir
```

```
[4]: WindowsPath('C:/hyperbox-brain')
```

Create the path to the Python file containing the implementation of the FMNN using the enhanced online learning algorithm

```
[5]: efmnn_file_path = os.path.join(project_dir, Path("hbbrain/numerical_data/incremental_
↳ learner/efmnn.py"))
efmnn_file_path

[5]: 'C:\\hyperbox-brain\\hbbrain\\numerical_data\\incremental_learner\\efmnn.py'
```

Run the found file by showing the execution directions

```
[6]: !python "{efmnn_file_path}" -h

usage: efmnn.py [-h] -training_file TRAINING_FILE -testing_file TESTING_FILE
               [--theta THETA] [--gamma GAMMA] [--is_draw IS_DRAW]

The description of parameters

required arguments:
  -training_file TRAINING_FILE
                        A required argument for the path to training data file
                        (including file name)
  -testing_file TESTING_FILE
                        A required argument for the path to testing data file
                        (including file name)

optional arguments:
  --theta THETA        Maximum hyperbox size (in the range of (0, 1])
                        (default: 0.5)
  --gamma GAMMA        A sensitivity parameter describing the speed of
                        decreasing of the membership function in each
                        dimension (larger than 0) (default: 1)
  --is_draw IS_DRAW    Show the existing hyperboxes during the training
                        process on the screen (default: False)
```

Create the path to training and testing datasets stored in the dataset folder

```
[7]: training_data_file = os.path.join(project_dir, Path("dataset/syn_num_train.csv"))
training_data_file

[7]: 'C:\\hyperbox-brain\\dataset\\syn_num_train.csv'

[8]: testing_data_file = os.path.join(project_dir, Path("dataset/syn_num_test.csv"))
testing_data_file

[8]: 'C:\\hyperbox-brain\\dataset\\syn_num_test.csv'
```

## Run a demo program

```
[9]: !python "{efmnn_file_path}" -training_file "{training_data_file}" -testing_file "  
    ↪{testing_data_file}" --theta 0.1 --gamma 1
```

```
Number of hyperboxes = 77  
Testing accuracy = 85.80%
```

## 2. Using the EFMNN through its init, fit, and predict functions

```
[10]: from hbbbrain.numerical_data.incremental_learner.efmnn import EFMNNClassifier  
import pandas as pd
```

### Create training and testing data sets

```
[11]: df_train = pd.read_csv(training_data_file, header=None)  
df_test = pd.read_csv(testing_data_file, header=None)  
  
Xy_train = df_train.to_numpy()  
Xy_test = df_test.to_numpy()  
  
Xtr = Xy_train[:, :-1]  
ytr = Xy_train[:, -1]  
  
Xtest = Xy_test[:, :-1]  
ytest = Xy_test[:, -1]
```

### Initializing parameters

```
[12]: theta = 0.1  
gamma = 1  
is_draw = True
```

### Training

```
[13]: efmnn_clf = EFMNNClassifier(theta=theta, gamma=gamma, is_draw=is_draw)  
efmnn_clf.fit(Xtr, ytr)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[13]: EFMNNClassifier(C=array([1, 2, 1, 1, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 2, ↵  
    ↪1,  
        1, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2,  
        1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 2, 2, 2, 1, 2,  
        2, 1, 1, 2, 1, 2, 2, 1, 2, 1, 2])),
```

(continues on next page)

(continued from previous page)

```

V=array([[0.42413    , 0.53516    ],
 [0.74088    , 0.43310562],
 [0.91059    , 0.82085    ],
 [0.64586    , 0.50818437],
 [0.4713     , 0.672     ],
 [0.33484    , 0.18424    ],
 [0.32289    , 0.59994    ],
 [0.19944    , 0.03      ],
 [0.28077    , 0.26124...
 [0.36842    , 0.77238    ],
 [0.36057    , 0.71561    ],
 [0.6733     , 0.29485    ],
 [0.25621    , 0.62174    ],
 [0.14737    , 0.28498    ],
 [0.72496    , 0.34978    ],
 [0.73681    , 0.71261    ],
 [0.76138    , 0.32629    ],
 [0.52124    , 0.41288    ],
 [0.42647    , 0.66846    ],
 [0.91185    , 0.48697    ],
 [0.29163    , 0.086547   ],
 [0.35813    , 0.58584    ],
 [0.56487    , 0.17003    ],
 [0.287      , 0.46814    ],
 [0.44575    , 0.64683    ],
 [0.25924    , 0.42696    ],
 [0.77398    , 0.5357     ],
 [0.68469    , 0.2221     ]]),
is_draw=True, theta=0.1)

```

The code below shows how to display decision boundaries among classes if input data are 2-dimensional

```
[14]: efmmn_clf.draw_hyperbox_and_boundary("The trained FMNN and its decision boundaries")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[15]: print("Number of existing hyperboxes = %d"%(efmmn_clf.get_n_hyperboxes()))
```

```
Number of existing hyperboxes = 77
```

## Prediction

```
[16]: from sklearn.metrics import accuracy_score
```

```
[17]: y_pred = efmmn_clf.predict(Xtest)
      acc = accuracy_score(ytest, y_pred)
      print(f'Accuracy = {acc * 100: .2f}%')
```

```
Accuracy = 85.80%
```

## Explaining the predicted result for the input sample by showing membership values and hyperboxes for each class

```
[18]: sample_need_explain = 10
      y_pred_input_0, mem_val_classes, min_points_classes, max_points_classes = efmmn_clf.get_
      ↪sample_explanation(Xtest[sample_need_explain])
```

```
[19]: print("Predicted class for sample X = [%f, %f] is %d and real class is %d" %
      ↪(Xtest[sample_need_explain, 0], Xtest[sample_need_explain, 1], y_pred_input_0,
      ↪ytest[sample_need_explain]))
```

```
Predicted class for sample X = [0.571640, 0.233700] is 2 and real class is 2
```

```
[20]: print("Membership values:")
      for key, val in mem_val_classes.items():
          print("Class %d has the maximum membership value = %f" % (key, val))

      for key in min_points_classes:
          print("Class %d has the representative hyperbox: V = %s and W = %s" % (key, min_
          ↪points_classes[key], max_points_classes[key]))
```

```
Membership values:
```

```
Class 1 has the maximum membership value = 0.957157
```

```
Class 2 has the maximum membership value = 0.995862
```

```
Class 1 has the representative hyperbox: V = [0.55763 0.40507] and W = [0.58339 0.43813]
```

```
Class 2 has the representative hyperbox: V = [0.57285 0.24904] and W = [0.66773 0.31638]
```

## Show input sample and hyperboxes belonging to each class. In 2D, we can show rectangles or use parallel coordinates

### Using rectangles to show explanations

```
[21]: efmmn_clf.show_sample_explanation(Xtest[sample_need_explain], Xtest[sample_need_explain],
      ↪ min_points_classes, max_points_classes, y_pred_input_0, "2D")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

### Using parallel coordinates. This mode best fits for any dimensions

```
[22]: # Create a parallel coordinates graph
efmnn_clf.show_sample_explanation(Xtest[sample_need_explain], Xtest[sample_need_explain],
    ↪ min_points_classes, max_points_classes, y_pred_input_0, file_path="par_cord/efmnn_par_
    ↪ cord.html")

[23]: # Load parallel coordinates to display on the notebook
from IPython.display import IFrame
# We load the parallel coordinates from GitHub here for demonstration in readthedocs
# On the local notebook, we only need to load from the graph storing at 'par_cord/efmnn_
    ↪ par_cord.html'
IFrame('https://uts-caslab.github.io/hyperbox-brain/docs/tutorials/par_cord/efmnn_par_
    ↪ cord.html', width=820, height=520)

[23]: <IPython.lib.display.IFrame at 0x210cf663c18>
```

### Enhanced Online Learning Algorithm with K-nearest Hyperboxes Selection for FMNN

This example shows how to use the Simpson's Fuzzy Min-Max Neural Network classifier using an enhanced online learning algorithm with k-nearest hyperboxes selection (KNEFMNN)

Note that the numerical features in training and testing datasets must be in the range of [0, 1] because the KNEFMNN classifier require features in the unit cube.

#### 1. Execute directly from the python file

```
[1]: %matplotlib notebook

[2]: import os
import warnings
warnings.filterwarnings('ignore')
```

#### Get the path to the this jupyter notebook file

```
[3]: this_notebook_dir = os.path.dirname(os.path.abspath("__file__"))
this_notebook_dir

[3]: 'C:\\hyperbox-brain\\docs\\tutorials'
```

### Get the home folder of the Hyperbox-Brain project

```
[4]: from pathlib import Path
project_dir = Path(this_notebook_dir).parent.parent
project_dir

[4]: WindowsPath('C:/hyperbox-brain')
```

### Create the path to the Python file containing the implementation of the Simpson's FMNN classifier using the enhanced online learning algorithm with k-nearest hyperboxes selection

```
[5]: knefmnn_file_path = os.path.join(project_dir, Path("hbbrain/numerical_data/incremental_
↳ learner/knefmnn.py"))
knefmnn_file_path

[5]: 'C:\\hyperbox-brain\\hbbrain\\numerical_data\\incremental_learner\\knefmnn.py'
```

### Run the found file by showing the execution directions

```
[6]: !python "{knefmnn_file_path}" -h

usage: knefmnn.py [-h] -training_file TRAINING_FILE -testing_file TESTING_FILE
                  [--theta THETA] [--gamma GAMMA] [--k_neighbors K_NEIGHBORS]
                  [--is_draw IS_DRAW]

The description of parameters

required arguments:
  -training_file TRAINING_FILE
                        A required argument for the path to training data file
                        (including file name)
  -testing_file TESTING_FILE
                        A required argument for the path to testing data file
                        (including file name)

optional arguments:
  --theta THETA         Maximum hyperbox size (in the range of (0, 1])
                        (default: 0.5)
  --gamma GAMMA         A sensitivity parameter describing the speed of
                        decreasing of the membership function in each
                        dimension (larger than 0) (default: 1)
  --k_neighbors K_NEIGHBORS
                        The number of nearest hyperboxes is considered for the
                        hyperbox expansion process
  --is_draw IS_DRAW     Show the existing hyperboxes during the training
                        process on the screen (default: False)
```



**Create the path to training and testing datasets stored in the dataset folder**

```
[7]: training_data_file = os.path.join(project_dir, Path("dataset/syn_num_train.csv"))
      training_data_file
```

```
[7]: 'C:\\hyperbox-brain\\dataset\\syn_num_train.csv'
```

```
[8]: testing_data_file = os.path.join(project_dir, Path("dataset/syn_num_test.csv"))
      testing_data_file
```

```
[8]: 'C:\\hyperbox-brain\\dataset\\syn_num_test.csv'
```

**Run a demo program**

```
[9]: !python "{knefmnn_file_path}" -training_file "{training_data_file}" -testing_file "
      ↪{testing_data_file}" --theta 0.1 --k_neighbors 5 --gamma 1
```

```
Number of hyperboxes = 51
Testing accuracy = 86.60%
```

**2. Using the KNEFMNN classifier through its init, fit, and predict functions**

```
[10]: from hbbrain.numerical_data.incremental_learner.knefmnn import KNEFMNNClassifier
      import pandas as pd
```

**Create training and testing data sets**

```
[11]: df_train = pd.read_csv(training_data_file, header=None)
      df_test = pd.read_csv(testing_data_file, header=None)
```

```
Xy_train = df_train.to_numpy()
Xy_test = df_test.to_numpy()
```

```
Xtr = Xy_train[:, :-1]
ytr = Xy_train[:, -1]
```

```
Xtest = Xy_test[:, :-1]
ytest = Xy_test[:, -1]
```

## Initializing parameters

```
[12]: theta = 0.1
      k_neighbors = 5
      gamma = 1
      is_draw = True
```

## Training

```
[13]: knefmnn_clf = KNEFMNNClassifier(theta=theta, k_neighbors=k_neighbors, gamma=gamma, is_
      ↪ draw=is_draw)
      knefmnn_clf.fit(Xtr, ytr)

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

[13]: KNEFMNNClassifier(C=array([[1, 2, 1, 1, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 2,
      ↪ 1,
      1, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 2, 2, 1, 1, 1, 2, 1, 2, 1, 1, 2,
      1, 2, 2, 2, 2, 2, 1]]),
      V=array([[0.42413 , 0.53516 ],
      [0.70577 , 0.397105 ],
      [0.82785 , 0.78025 ],
      [0.66038 , 0.51128 ],
      [0.48794 , 0.672 ],
      [0.26651 , 0.18424 ],
      [0.32289 , 0.59994 ],
      [0.19944 , 0.03 ],
      [0.28077 , 0.26124 ],
      [0.63683 , 0.6936 ],
      [0.28822 , 0.55512 ],
      [0.03 , 0.47757 ],
      [0...
      [0.91185 , 0.5761 ],
      [0.2246 , 0.13567 ],
      [0.25929 , 0.81558 ],
      [0.815 , 0.397095 ],
      [0.67906 , 0.83605 ],
      [0.52197 , 0.91371 ],
      [0.66037 , 0.57837 ],
      [0.49408 , 0.66846 ],
      [0.80583 , 0.43242 ],
      [0.79935 , 0.7757 ],
      [0.35813 , 0.58772 ],
      [0.79516 , 0.32629 ],
      [0.36057 , 0.71561 ],
      [0.68469 , 0.29485 ],
      [0.70743 , 0.50325 ],
      [0.25621 , 0.62174 ],
      [0.14737 , 0.28498 ],
      [0.56487 , 0.17003 ]],
```

(continues on next page)

(continued from previous page)

```
[0.55763 , 0.43813 ])),
is_draw=True, theta=0.1)
```

The code below shows how to display decision boundaries among classes if input data are 2-dimensional

```
[14]: knefmnn_clf.draw_hyperbox_and_boundary("The trained KNEFMNN classifier and its decision_
↳ boundaries")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[15]: print("Number of existing hyperboxes = %d"%(knefmnn_clf.get_n_hyperboxes()))
```

```
Number of existing hyperboxes = 51
```

## Prediction

```
[16]: from sklearn.metrics import accuracy_score
```

```
[17]: y_pred = knefmnn_clf.predict(Xtest)
acc = accuracy_score(ytest, y_pred)
print(f'Accuracy = {acc * 100: .2f}%')
```

```
Accuracy = 86.60%
```

Explaining the predicted result for the input sample by showing membership values and hyperboxes for each class

```
[18]: sample_need_explain = 10
y_pred_input_0, mem_val_classes, min_points_classes, max_points_classes = knefmnn_clf.
↳ get_sample_explanation(Xtest[sample_need_explain])
```

```
[19]: print("Predicted class for sample X = [%f, %f] is %d and real class is %d" %_
↳ (Xtest[sample_need_explain, 0], Xtest[sample_need_explain, 1], y_pred_input_0,
↳ ytest[sample_need_explain]))
```

```
Predicted class for sample X = [0.571640, 0.233700] is 2 and real class is 2
```

```
[20]: print("Membership values:")
for key, val in mem_val_classes.items():
    print("Class %d has the maximum membership value = %f" % (key, val))

for key in min_points_classes:
    print("Class %d has the representative hyperbox: V = %s and W = %s" % (key, min_
↳ points_classes[key], max_points_classes[key]))
```

Membership values:

Class 1 has the maximum membership value = 0.964263

Class 2 has the maximum membership value = 0.990050

Class 1 has the representative hyperbox:  $V = [0.58339 \ 0.3649]$  and  $W = [0.66091 \ 0.38616125]$

Class 2 has the representative hyperbox:  $V = [0.57285 \ 0.27229]$  and  $W = [0.66773 \ 0.36489]$

**Show input sample and hyperboxes belonging to each class. In 2D, we can show rectangles or use parallel coordinates**

### Using rectangles to show explanations

```
[21]: knefmn_clf.show_sample_explanation(Xtest[sample_need_explain], Xtest[sample_need_
      ↪explain], min_points_classes, max_points_classes, y_pred_input_0, "2D")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

### Using parallel coordinates. This mode best fits for any dimensions

```
[22]: # Create a parallel coordinates graph
      ↪ knefmn_clf.show_sample_explanation(Xtest[sample_need_explain], Xtest[sample_need_
      ↪explain], min_points_classes, max_points_classes, y_pred_input_0, file_path="par_cord/
      ↪knefmn_par_cord.html")
```

```
[23]: # Load parallel coordinates to display on the notebook
      ↪ from IPython.display import IFrame
      ↪ # We load the parallel coordinates from GitHub here for demonstration in readthedocs
      ↪ # On the local notebook, we only need to load from the graph storing at 'par_cord/knefmnn_
      ↪par_cord.html'
      ↪ IFrame('https://uts-caslab.github.io/hyperbox-brain/docs/tutorials/par_cord/knefmnn_par_
      ↪cord.html', width=820, height=520)
```

```
[23]: <IPython.lib.display.IFrame at 0x2da0fabd518>
```

### Refined Online Learning Algorithm for FMNN

This example shows how to use the fuzzy min-max neural network classifier using a refined online learning algorithm (RFMNN)

Note that the numerical features in training and testing datasets must be in the range of [0, 1] because the RFMNN classifiers require features in the unit cube.

## 1. Execute directly from the python file

```
[1]: %matplotlib notebook
```

```
[2]: import os
import warnings
warnings.filterwarnings('ignore')
```

### Get the path to the this jupyter notebook file

```
[3]: this_notebook_dir = os.path.dirname(os.path.abspath("__file__"))
this_notebook_dir
```

```
[3]: 'C:\\hyperbox-brain\\docs\\tutorials'
```

### Get the home folder of the Hyperbox-Brain project

```
[4]: from pathlib import Path
project_dir = Path(this_notebook_dir).parent.parent
project_dir
```

```
[4]: WindowsPath('C:/hyperbox-brain')
```

### Create the path to the Python file containing the implementation of the RFMNN classifier

```
[5]: rfmmn_file_path = os.path.join(project_dir, Path("hbbrain/numerical_data/incremental_
↪ learner/rfmmn.py"))
rfmmn_file_path
```

```
[5]: 'C:\\hyperbox-brain\\hbbrain\\numerical_data\\incremental_learner\\rfmmn.py'
```

### Run the found file by showing the execution directions

```
[6]: !python "{rfmmn_file_path}" -h
```

```
usage: rfmmn.py [-h] -training_file TRAINING_FILE -testing_file TESTING_FILE
               [--theta THETA] [--gamma GAMMA] [--is_draw IS_DRAW]
```

The description of parameters

required arguments:

```
-training_file TRAINING_FILE
                        A required argument for the path to training data file
                        (including file name)
-testing_file TESTING_FILE
                        A required argument for the path to testing data file
                        (including file name)
```

(continues on next page)

(continued from previous page)

```
optional arguments:
  --theta THETA      Maximum hyperbox size (in the range of (0, 1])
                     (default: 0.5)
  --gamma GAMMA      A sensitivity parameter describing the speed of
                     decreasing of the membership function in each
                     dimension (larger than 0) (default: 1)
  --is_draw IS_DRAW  Show the existing hyperboxes during the training
                     process on the screen (default: False)
```

### Create the path to training and testing datasets stored in the dataset folder

```
[7]: training_data_file = os.path.join(project_dir, Path("dataset/syn_num_train.csv"))
     training_data_file
```

```
[7]: 'C:\\hyperbox-brain\\dataset\\syn_num_train.csv'
```

```
[8]: testing_data_file = os.path.join(project_dir, Path("dataset/syn_num_test.csv"))
     testing_data_file
```

```
[8]: 'C:\\hyperbox-brain\\dataset\\syn_num_test.csv'
```

### Run a demo program

```
[9]: !python "{rfmnn_file_path}" -training_file "{training_data_file}" -testing_file "
     ↪{testing_data_file}" --theta 0.1 --gamma 1
```

```
Number of hyperboxes = 95
Testing accuracy = 86.20%
```

## 2. Using the RFMNN classifier through its init, fit, and predict functions

```
[10]: from hbbrain.numerical_data.incremental_learner.rfmnn import RFMNNClassifier
     import pandas as pd
```

### Create training and testing data sets

```
[11]: df_train = pd.read_csv(training_data_file, header=None)
     df_test = pd.read_csv(testing_data_file, header=None)

     Xy_train = df_train.to_numpy()
     Xy_test = df_test.to_numpy()

     Xtr = Xy_train[:, :-1]
     ytr = Xy_train[:, -1]
```

(continues on next page)

(continued from previous page)

```
Xtest = Xy_test[:, :-1]
ytest = Xy_test[:, -1]
```

## Initializing parameters

```
[12]: theta = 0.1
      gamma = 1
      is_draw = True
```

## Training

```
[13]: rfmmnn_clf = RFMMNNClassifier(theta=theta, gamma=gamma, is_draw=is_draw)
      rfmmnn_clf.fit(Xtr, ytr)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[13]: RFMMNNClassifier(C=array([1, 2, 1, 1, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 2, 2, 2, 1, 2,
↪ 1,
      2, 1, 1, 2, 2, 2, 2, 2, 1, 2, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1,
      2, 1, 2, 2, 1, 2, 1, 1, 2, 1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1,
      2, 1, 1, 1, 2, 2, 1, 1, 2, 2, 2, 1, 1, 1, 2, 2, 2, 1, 1, 2, 1, 2,
      1, 2, 2, 1, 2, 1, 2]),
      V=array([[0.42413 , 0.53516 ],
      [0.77074 , 0.48234 ],
      [0.91059 , 0.82085 ],
      [0.64586 , 0.46818 ],
      [0.4713 , 0.672 ],
      [0.33484 , 0.18424 ],
      [0.32289 , 0.60093 ],
      [...],
      [0.72496 , 0.34978 ],
      [0.73681 , 0.71261 ],
      [0.38312 , 0.65216 ],
      [0.37646 , 0.62974 ],
      [0.76138 , 0.32629 ],
      [0.52124 , 0.41288 ],
      [0.59655 , 0.56029 ],
      [0.38038 , 0.67232 ],
      [0.91185 , 0.48697 ],
      [0.6504 , 0.51624 ],
      [0.65227 , 0.56081 ],
      [0.29163 , 0.086547],
      [0.35813 , 0.58584 ],
      [0.56487 , 0.17003 ],
      [0.287 , 0.46814 ],
      [0.44575 , 0.64683 ],
      [0.25924 , 0.42696 ],
      [0.77398 , 0.5357 ]],
```

(continues on next page)

(continued from previous page)

```
[0.68469 , 0.2221  ]],  
        is_draw=True, theta=0.1)
```

The code below shows how to display decision boundaries among classes if input data are 2-dimensional

```
[14]: rfmmn_clf.draw_hyperbox_and_boundary("The trained RFMNN classifier and its decision_  
      ↪ boundaries")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[15]: print("Number of existing hyperboxes = %d"%(rfmmn_clf.get_n_hyperboxes()))
```

```
Number of existing hyperboxes = 95
```

```
[16]: print("Traning time = %f (s)"%rfmmn_clf.elapsed_training_time)
```

```
Traning time = 48.427096 (s)
```

## Prediction

```
[17]: from sklearn.metrics import accuracy_score
```

```
[18]: y_pred = rfmmn_clf.predict(Xtest)  
acc = accuracy_score(ytest, y_pred)  
print(f'Accuracy = {acc * 100: .2f}%')
```

```
Accuracy = 86.20%
```

Explaining the predicted result for the input sample by showing membership values and hyperboxes for each class

```
[19]: sample_need_explain = 10  
y_pred_input_0, mem_val_classes, min_points_classes, max_points_classes = rfmmn_clf.get_  
      ↪ sample_explanation(Xtest[sample_need_explain])
```

```
[20]: print("Predicted class for sample X = [%f, %f] is %d and real class is %d" %_  
      ↪ (Xtest[sample_need_explain, 0], Xtest[sample_need_explain, 1], y_pred_input_0,_  
      ↪ ytest[sample_need_explain]))
```

```
Predicted class for sample X = [0.571640, 0.233700] is 2 and real class is 2
```

```
[21]: print("Membership values:")  
for key, val in mem_val_classes.items():  
    print("Class %d has the maximum membership value = %f" % (key, val))
```

(continues on next page)



(continued from previous page)

```
for key in min_points_classes:
    print("Class %d has the representative hyperbox: V = %s and W = %s" % (key, min_
    ↪ points_classes[key], max_points_classes[key]))
```

Membership values:  
 Class 1 has the maximum membership value = 0.957157  
 Class 2 has the maximum membership value = 0.995862  
 Class 1 has the representative hyperbox: V = [0.55763 0.40507] and W = [0.58339 0.43813]  
 Class 2 has the representative hyperbox: V = [0.57285 0.24904] and W = [0.66773 0.31638]

**Show input sample and hyperboxes belonging to each class. In 2D, we can show rectangles or use parallel coordinates**

**Using rectangles to show explanations**

```
[22]: rfmmn_clf.show_sample_explanation(Xtest[sample_need_explain], Xtest[sample_need_explain],
    ↪ min_points_classes, max_points_classes, y_pred_input_0, "2D")
```

<IPython.core.display.Javascript object>  
 <IPython.core.display.HTML object>

**Using parallel coordinates. This mode best fits for any dimensions**

```
[23]: # Create a parallel coordinates graph
rfmmn_clf.show_sample_explanation(Xtest[sample_need_explain], Xtest[sample_need_explain],
    ↪ min_points_classes, max_points_classes, y_pred_input_0, file_path="par_cord/rfmmn_par_
    ↪ cord.html")
```

```
[24]: # Load parallel coordinates to display on the notebook
from IPython.display import IFrame
# We load the parallel coordinates from GitHub here for demonstration in readthedocs
# On the local notebook, we only need to load from the graph storing at 'par_cord/rfmmn_
    ↪ par_cord.html'
IFrame('https://uts-caslab.github.io/hyperbox-brain/docs/tutorials/par_cord/rfmmn_par_
    ↪ cord.html', width=820, height=520)
```

```
[24]: <IPython.lib.display.IFrame at 0x1de537f45f8>
```

## 2.8.3 Multigranular learners

### Multi-resolution Hierarchical Granular Representation based Classifier using GFMM

This example shows how to use the multi-resolution hierarchical granular representation based classifier using general fuzzy min-max neural network.

## 1. Execute directly from the python file

```
[1]: %matplotlib notebook
```

```
[2]: import os
import warnings
warnings.filterwarnings('ignore')
```

### Get the path to the this jupyter notebook file

```
[3]: this_notebook_dir = os.path.dirname(os.path.abspath("__file__"))
this_notebook_dir
```

```
[3]: 'C:\\hyperbox-brain\\docs\\tutorials'
```

### Get the home folder of the Hyperbox-Brain project

```
[4]: from pathlib import Path
project_dir = Path(this_notebook_dir).parent.parent
project_dir
```

```
[4]: WindowsPath('C:/hyperbox-brain')
```

### Create the path to the Python file containing the implementation of the multi-resolution hierarchical granular representation based classifier using the general fuzzy min-max neural network

```
[5]: multi_resolution_gfmm_file_path = os.path.join(project_dir, Path("hbbrain/numerical_data/
↳ multigranular_learner/multi_resolution_gfmm.py"))
multi_resolution_gfmm_file_path
```

```
[5]: 'C:\\hyperbox-brain\\hbbrain\\numerical_data\\multigranular_learner\\multi_resolution_
↳ gfmm.py'
```

### Run the found file by showing the execution directions

```
[6]: !python "{multi_resolution_gfmm_file_path}" -h
```

```
usage: multi_resolution_gfmm.py [-h] -training_file TRAINING_FILE
                                -testing_file TESTING_FILE
                                [--val_file VAL_FILE]
                                [--n_partitions N_PARTITIONS]
                                [--granular_theta GRANULAR_THETA]
                                [--gamma GAMMA]
                                [--min_membership_aggregation MIN_MEMBERSHIP_AGGREGATION]
```

The description of parameters

(continues on next page)

(continued from previous page)

required arguments:

- training\_file TRAINING\_FILE  
A required argument for the path to training data file (including file name)
- testing\_file TESTING\_FILE  
A required argument for the path to testing data file (including file name)

optional arguments:

- val\_file VAL\_FILE The path to validation data file (including file name)
- n\_partitions N\_PARTITIONS  
Number of disjoint partitions to train base learners (default: 4)
- granular\_theta GRANULAR\_THETA  
Granular maximum hyperbox sizes (default: [0.1, 0.2, 0.3, 0.4, 0.5])
- gamma GAMMA  
A sensitivity parameter describing the speed of decreasing of the membership function in each dimension (larger than 0) (default: 1)
- min\_membership\_aggregation MIN\_MEMBERSHIP\_AGGREGATION  
Minimum membership value for hyperbox aggregation at higher granular levels (in the range of [0, 1]) (default: 0)

### Create the path to training and testing datasets stored in the dataset folder

```
[7]: training_data_file = os.path.join(project_dir, Path("dataset/syn_num_train.csv"))
training_data_file
```

```
[7]: 'C:\\hyperbox-brain\\dataset\\syn_num_train.csv'
```

```
[8]: testing_data_file = os.path.join(project_dir, Path("dataset/syn_num_test.csv"))
testing_data_file
```

```
[8]: 'C:\\hyperbox-brain\\dataset\\syn_num_test.csv'
```

### Run a demo program

If the argument 'validation\_file' gets the value of validation file path, the pruning procedure will be used after merging all hyperboxes from base learners. Otherwise, the pruning procedure will not be used.

```
[9]: !python "{multi_resolution_gfmm_file_path}" -training_file "{training_data_file}" -
↪testing_file "{testing_data_file}" --n_partitions 4 --granular_theta "[0.1, 0.2, 0.3,
↪0.4, 0.5, 0.6]" --gamma 1 --min_membership_aggregation 0.1
```

Training time: 3.847 (s)

Testing accuracy (using voting from all granularity levels) = 86.70%

Prediction of each base learner at a given partition:

```
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.  
[Parallel(n_jobs=4)]: Done 2 out of 4 | elapsed: 3.6s remaining: 3.6s  
[Parallel(n_jobs=4)]: Done 4 out of 4 | elapsed: 3.7s finished
```

```
Partition 0 - Testing accuracy = 84.00% - No boxes = 27  
Partition 1 - Testing accuracy = 87.80% - No boxes = 29  
Partition 2 - Testing accuracy = 85.40% - No boxes = 27  
Partition 3 - Testing accuracy = 87.10% - No boxes = 26  
Prediction for each granularity level:  
Level 1 - Testing accuracy = 85.10% - No boxes = 101  
Level 2 - Testing accuracy = 88.20% - No boxes = 38  
Level 3 - Testing accuracy = 87.10% - No boxes = 27  
Level 4 - Testing accuracy = 86.10% - No boxes = 20  
Level 5 - Testing accuracy = 86.20% - No boxes = 14  
Level 6 - Testing accuracy = 82.60% - No boxes = 10
```

## 2. Using the multi-resolution hierarchical granular representation based classifier using general fuzzy min-max neural network through init, fit, and predict functions

```
[10]: from hbbbrain.numerical_data.multigranular_learner.multi_resolution_gfmm import   
      ↪ MultiGranularGFMM  
      import pandas as pd
```

### Create training and testing data sets

```
[11]: df_train = pd.read_csv(training_data_file, header=None)  
      df_test = pd.read_csv(testing_data_file, header=None)  
  
      Xy_train = df_train.to_numpy()  
      Xy_test = df_test.to_numpy()  
  
      Xtr = Xy_train[:, :-1]  
      ytr = Xy_train[:, -1]  
  
      Xtest = Xy_test[:, :-1]  
      ytest = Xy_test[:, -1]
```

### Initializing parameters

```
[12]: # number of disjoint partitions to build base learners  
      n_partitions = 4  
      # a list of maximum hyperbox sizes for granularity levels  
      granular_theta = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6]  
      # minimum membership values between two hyperboxes aggregated at higher abstraction   
      ↪ levels  
      min_membership_aggregation = 0.1  
      # the speed of decreasing of membership values  
      gamma = 1
```

## Training

```
[13]: from hbbbrain.constants import HETEROGENEOUS_CLASS_LEARNING
multi_granular_gfmm_clf = MultiGranularGFMM(n_partitions=n_partitions, granular_
↳ theta=granular_theta, gamma=gamma, min_membership_aggregation=min_membership_
↳ aggregation)
# Training using the heterogeneous model for class labels.
multi_granular_gfmm_clf.fit(Xtr, ytr, learning_type=HETEROGENEOUS_CLASS_LEARNING)

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done 2 out of 4 | elapsed: 2.9s remaining: 2.9s
[Parallel(n_jobs=4)]: Done 4 out of 4 | elapsed: 2.9s finished

[13]: MultiGranularGFMM(granular_theta=[0.1, 0.2, 0.3, 0.4, 0.5, 0.6],
min_membership_aggregation=0.1)
```

The code below shows how to display decision boundaries among classes at a given granularity level if input data are 2-dimensional

```
[14]: # showing hyperboxes and boundaries at the first granularity level (level 1). Note that,
↳ the order of the level starts from 0.
multi_granular_gfmm_clf.draw_2D_hyperbox_and_boundary_granular_level(window_name=
↳ "Hyperbox-based classifier and its decision boundaries at level 1", level = 0)
```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

```
[15]: # showing hyperboxes and boundaries at the last granularity level (level 6)
multi_granular_gfmm_clf.draw_2D_hyperbox_and_boundary_granular_level(window_name=
↳ "Hyperbox-based classifier and its decision boundaries at level 6", level = 5)
```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

```
[16]: # Get total number of hyperboxes at all granularity levels
print("Total number of hyperboxes at all granularity levels = %d"%multi_granular_gfmm_
↳ clf.get_n_hyperboxes(level=-1))
```

Total number of hyperboxes at all granularity levels = 210

```
[17]: # Get number of hyperboxes at a given granularity level
print("Total number of hyperboxes at the first granularity levels = %d"%multi_granular_
↳ gfmm_clf.get_n_hyperboxes(level=0))
print("Total number of hyperboxes at the last granularity levels = %d"%multi_granular_
↳ gfmm_clf.get_n_hyperboxes(level=5))
```

Total number of hyperboxes at the first granularity levels = 101

Total number of hyperboxes at the last granularity levels = 10

## Prediction

Using all GFMM models from all granularity level to make the final prediction using majority voting, in which each granularity level contributes one predicted result for each input pattern and the final predicted result is the class getting most of votes from the models at all granularity levels.

```
[18]: from sklearn.metrics import accuracy_score

[19]: y_pred = multi_granular_gfmm_clf.predict(Xtest, level=-1)
      acc = accuracy_score(ytest, y_pred)
      print(f'Accuracy (majority voting) = {acc * 100: .2f}%')

Accuracy (majority voting) = 86.70%
```

## Use a certain granularity level to make prediction

```
[20]: print("Prediction for each granularity level:")
      for i in range(len(granular_theta)):
          y_pred_lv = multi_granular_gfmm_clf.predict(Xtest, level=i)
          acc_lv = accuracy_score(ytest, y_pred_lv)
          n_boxes = multi_granular_gfmm_clf.get_n_hyperboxes(i)
          print(f'Level {i + 1} - Testing accuracy = {acc_lv * 100: .2f}% - No hyperboxes = {n_
→boxes}')

Prediction for each granularity level:
Level 1 - Testing accuracy = 85.10% - No hyperboxes = 101
Level 2 - Testing accuracy = 88.20% - No hyperboxes = 38
Level 3 - Testing accuracy = 87.10% - No hyperboxes = 27
Level 4 - Testing accuracy = 86.10% - No hyperboxes = 20
Level 5 - Testing accuracy = 86.20% - No hyperboxes = 14
Level 6 - Testing accuracy = 82.60% - No hyperboxes = 10
```

Explaining the predicted result for the input sample by showing membership values and hyperboxes for each class using the model at a given granularity level

```
[21]: sample_need_explain = 0
      # Using the trained model at the sixth granularity level to make prediction and
      →explanation. Note that the value for the level parameter starts from 0.
      level_explain = 5
      y_pred_input_0, mem_val_classes, min_points_classes, max_points_classes = multi_granular_
      →gfmm_clf.get_sample_explanation_granular_level(Xtest[sample_need_explain],
      →Xtest[sample_need_explain], level_explain)

[22]: print("Predicted class for sample X = [%f, %f] is %d and real class is %d" %
      →(Xtest[sample_need_explain, 0], Xtest[sample_need_explain, 1], y_pred_input_0,
      →ytest[sample_need_explain]))

Predicted class for sample X = [0.752930, 0.385920] is 1 and real class is 2
```

```
[23]: print("Membership values:")
      for key, val in mem_val_classes.items():
          print("Class %d has the maximum membership value = %f" % (key, val))

      for key in min_points_classes:
          print("Class %d has the representative hyperbox: V = %s and W = %s" % (key, min_
            ↪ points_classes[key], max_points_classes[key]))
```

Membership values:  
 Class 1 has the maximum membership value = 0.989125  
 Class 2 has the maximum membership value = 0.915132  
 Class 1 has the representative hyperbox: V = [0.763805 0.369765] and W = [0.91185 0.  
 ↪ 48598]  
 Class 2 has the representative hyperbox: V = [0.563065 0.17003 ] and W = [0.6680625 0.  
 ↪ 65662 ]

Show input sample and hyperboxes belonging to each class. In 2D, we can show rectangles or use parallel coordinates

### Using rectangles to show explanations

```
[24]: multi_granular_gfmm_clf.show_sample_explanation(Xtest[sample_need_explain], Xtest[sample_
      ↪ need_explain], min_points_classes, max_points_classes, y_pred_input_0, "2D")
```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

Using parallel coordinates to show explanations. This method can be used for any number of dimensions

```
[25]: # Create a parallel coordinates graph
      multi_granular_gfmm_clf.show_sample_explanation(Xtest[sample_need_explain], Xtest[sample_
        ↪ need_explain], min_points_classes, max_points_classes, y_pred_input_0, file_path="par_
        ↪ cord/multi_resolution_gfmm_lv6_par_cord.html")
```

```
[26]: # Load parallel coordinates to display on the notebook
      from IPython.display import IFrame
      # We load the parallel coordinates from GitHub here for demonstration in readthedocs
      # On the local notebook, we only need to load from the graph storing at 'par_cord/multi_
        ↪ resolution_gfmm_lv6_par_cord.html'
      IFram('https://uts-caslab.github.io/hyperbox-brain/docs/tutorials/par_cord/multi_
        ↪ resolution_gfmm_lv6_par_cord.html', width=820, height=520)
```

```
[26]: <IPython.lib.display.IFrame at 0x1e8371cf588>
```

## 2.8.4 Ensemble learners

### Decision-level Bagging of Hyperbox-based Models

This example shows how to use a Bagging classifier of base hyperbox-based models trained on a full set of features and a subset of samples.

```
[1]: import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from hbbbrain.numerical_data.ensemble_learner.decision_comb_bagging import _
↳ DecisionCombinationBagging
from hbbbrain.numerical_data.incremental_learner.onln_gfmm import OnlineGFMM
```

#### Load dataset.

This example will use the breast cancer dataset available in sklearn to demonstrate how to use this ensemble classifier.

```
[2]: from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import MinMaxScaler
```

```
[3]: df = load_breast_cancer()
X = df.data
y = df.target
```

```
[4]: # Normailise data into the range of [0, 1] as hyperbox-based models only work in the _
↳ unit cube
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
```

```
[5]: # Split data into training, validation and testing sets
Xtr_val, X_test, ytr_val, y_test = train_test_split(X, y, train_size=0.8, random_state=0)
Xtr, X_val, ytr, y_val = train_test_split(X, y, train_size=0.75, random_state=0)
```

This example will use the GFMM classifier with the original online learning algorithm as base learners. However, any type of hyperbox-based learning algorithms in this library can also be used to train base learners.

### 1. Using random subsampling to generate training sets for various base learners

#### Training

```
[6]: # Initialise parameters
n_estimators = 20 # number of base learners
max_samples = 0.5 # sampling rate for samples
bootstrap = False # random subsampling without replacement
class_balanced = False # do not use the class-balanced sampling mode
n_jobs = 4 # number of processes is used to build base learners
```



```
[7]: # Init a hyperbox-based model used to train base learners
# Using the GFMM classifier with the original online learning algorithm with the maximum_
↳ hyperbox size 0.1
base_estimator = OnlineGFMM(theta=0.1)
```

```
[8]: dc_bagging_subsampling = DecisionCombinationBagging(base_estimator=base_estimator, n_
↳ estimators=n_estimators, max_samples=max_samples, bootstrap=bootstrap, class_
↳ balanced=class_balanced, n_jobs=n_jobs, random_state=0)
```

```
[9]: dc_bagging_subsampling.fit(Xtr, ytr)
```

```
[9]: DecisionCombinationBagging(base_estimator=OnlineGFMM(C=array([], dtype=float64),
V=array([], dtype=float64),
W=array([], dtype=float64),
theta=0.1),
n_estimators=20, n_jobs=4, random_state=0)
```

```
[10]: print("Training time: %.3f (s)"%(dc_bagging_subsampling.elapsed_training_time))
```

```
Training time: 4.355 (s)
```

```
[11]: print('Total number of hyperboxes from all base learners = %d'%dc_bagging_subsampling.
↳ get_n_hyperboxes())
```

```
Total number of hyperboxes from all base learners = 3948
```

## Prediction

```
[12]: y_pred = dc_bagging_subsampling.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print(f'Testing accuracy = {acc * 100: .2f}%')
```

```
Testing accuracy = 93.86%
```

## Apply pruning for base learners

```
[13]: acc_threshold=0.5 # minimum accuracy score of the unpruned hyperboxes
keep_empty_boxes=False # False means hyperboxes that do not join the prediction process_
↳ within the pruning procedure are also eliminated
dc_bagging_subsampling.simple_pruning_base_estimators(X_val, y_val, acc_threshold, keep_
↳ empty_boxes)
```

```
[13]: DecisionCombinationBagging(base_estimator=OnlineGFMM(C=array([], dtype=float64),
V=array([], dtype=float64),
W=array([], dtype=float64),
theta=0.1),
n_estimators=20, n_jobs=4, random_state=0)
```

```
[14]: print('Total number of hyperboxes from all base learners after pruning = %d'%dc_bagging_
↳ subsampling.get_n_hyperboxes())
```

```
Total number of hyperboxes from all base learners after pruning = 2195
```

### Prediction after doing a pruning procedure

```
[15]: y_pred_2 = dc_bagging_subsampling.predict(X_test)
      acc_pruned = accuracy_score(y_test, y_pred_2)
      print(f'Testing accuracy (after pruning) = {acc_pruned * 100: .2f}%')

Testing accuracy (after pruning) = 95.61%
```

## 2. Using random undersampling to generate class-balanced training sets for various base learners

### Training

```
[16]: # Initialise parameters
      n_estimators = 20 # number of base learners
      max_samples = 0.5 # sampling rate for samples
      bootstrap = False # random subsampling without replacement
      class_balanced = True # use the class-balanced sampling mode
      n_jobs = 4 # number of processes is used to build base learners

[17]: # Init a hyperbox-based model used to train base learners
      # Using the GFMM classifier with the original online learning algorithm with the maximum_
      ↪ hyperbox size 0.1
      base_estimator = OnlineGFMM(theta=0.1)

[18]: dc_bagging_class_balanced = DecisionCombinationBagging(base_estimator=base_estimator, n_
      ↪ estimators=n_estimators, max_samples=max_samples, bootstrap=bootstrap, class_
      ↪ balanced=class_balanced, n_jobs=n_jobs, random_state=0)

[19]: dc_bagging_class_balanced.fit(Xtr, ytr)

[19]: DecisionCombinationBagging(base_estimator=OnlineGFMM(C=array([], dtype=float64),
      V=array([], dtype=float64),
      W=array([], dtype=float64),
      theta=0.1),
      class_balanced=True, n_estimators=20, n_jobs=4,
      random_state=0)

[20]: print("Training time: %.3f (s)"%(dc_bagging_class_balanced.elapsed_training_time))

Training time: 0.271 (s)

[21]: print('Total number of hyperboxes from all base learners = %d'%dc_bagging_class_balanced.
      ↪ get_n_hyperboxes())

Total number of hyperboxes from all base learners = 4010
```

## Prediction

```
[22]: y_pred = dc_bagging_class_balanced.predict(X_test)
      acc = accuracy_score(y_test, y_pred)
      print(f'Testing accuracy = {acc * 100: .2f}%')
```

```
Testing accuracy = 92.11%
```

## Apply pruning for base learners

```
[23]: acc_threshold=0.5 # minimum accuracy score of the unpruned hyperboxes
      keep_empty_boxes=False # False means hyperboxes that do not join the prediction process,
      ↪ within the pruning procedure are also eliminated
      dc_bagging_class_balanced.simple_pruning_base_estimators(X_val, y_val, acc_threshold,
      ↪ keep_empty_boxes)
```

```
[23]: DecisionCombinationBagging(base_estimator=OnlineGFMM(C=array([], dtype=float64),
      V=array([], dtype=float64),
      W=array([], dtype=float64),
      theta=0.1),
      class_balanced=True, n_estimators=20, n_jobs=4,
      random_state=0)
```

```
[24]: print('Total number of hyperboxes from all base learners after pruning = %d'%dc_bagging_
      ↪ class_balanced.get_n_hyperboxes())
```

```
Total number of hyperboxes from all base learners after pruning = 2738
```

## Prediction after doing a pruning procedure

```
[25]: y_pred_2 = dc_bagging_class_balanced.predict(X_test)
      acc_pruned = accuracy_score(y_test, y_pred_2)
      print(f'Testing accuracy (after pruning) = {acc_pruned * 100: .2f}%')
```

```
Testing accuracy (after pruning) = 94.74%
```

## Decision-level Bagging of Hyperbox-based Models with Hyper-parameter Optimisation

This example shows how to use a Bagging classifier of base hyperbox-based models trained on a full set of features and a subset of samples, in which each base learner is trained by random search-based hyper-parameter tuning and k-fold cross-validation.

While the original bagging model in the class `DecisionCombinationBagging` uses the same base learners with the same hyperparameters, the cross-validation bagging model in the class `DecisionCombinationCrossValBagging` allows each base learner to use specific hyperparameters depending on the training data by performing random research to find the best combination of hyperparameters for each base learner.

```
[1]: import warnings
      warnings.filterwarnings('ignore')
      import numpy as np
```

(continues on next page)

(continued from previous page)

```

from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from hbbbrain.numerical_data.ensemble_learner.decision_comb_cross_val_bagging import 
    ↪ DecisionCombinationCrossValBagging
from hbbbrain.numerical_data.incremental_learner.onln_gfmm import OnlineGFMM

```

## Load dataset.

This example will use the breast cancer dataset available in sklearn to demonstrate how to use this ensemble classifier.

```

[2]: from sklearn.datasets import load_breast_cancer
     from sklearn.preprocessing import MinMaxScaler

```

```

[3]: df = load_breast_cancer()
     X = df.data
     y = df.target

```

```

[4]: # Normailise data into the range of [0, 1] as hyperbox-based models only work in the 
     ↪ unit cube
     scaler = MinMaxScaler()
     X = scaler.fit_transform(X)

```

```

[5]: # Split data into training, validation and testing sets
     Xtr_val, X_test, ytr_val, y_test = train_test_split(X, y, train_size=0.8, random_state=0)
     Xtr, X_val, ytr, y_val = train_test_split(X, y, train_size=0.75, random_state=0)

```

This example will use the GFMM classifier with the original online learning algorithm as base learners. However, any type of hyperbox-based learning algorithms in this library can also be used to train base learners.

## 1. Using random subsampling to generate training sets for various base learners

### Training

```

[6]: # Initialise parameters
     n_estimators = 20 # number of base learners
     max_samples = 0.5 # sampling rate for samples
     bootstrap = False # random subsampling without replacement
     class_balanced = False # do not use the class-balanced sampling mode
     n_jobs = 4 # number of processes is used to build base learners
     n_iter = 20 # Number of parameter settings that are randomly sampled to choose the best 
     ↪ combination of hyperparameters
     k_fold = 5 # Number of folds to conduct Stratified K-Fold cross-validation for 
     ↪ hyperparameter tuning

```

```

[7]: # Init a hyperbox-based model used to train base learners
     # Using the GFMM classifier with the original online learning algorithm
     base_estimator = OnlineGFMM()

```

```
[8]: # Init ranges for hyperparameters of base learners to perform a random search process_
      ↪ for hyperparameter tuning
      base_estimator_params = {'theta': np.arange(0.05, 1.01, 0.05), 'theta_min': [1], 'gamma':
      ↪ [0.5, 1, 2, 4, 8, 16]}

[9]: dc_cv_bagging_subsampling = DecisionCombinationCrossValBagging(base_estimator=base_
      ↪ estimator, base_estimator_params=base_estimator_params, n_estimators=n_estimators, max_
      ↪ samples=max_samples, bootstrap=bootstrap, class_balanced=class_balanced, n_iter=n_iter,
      ↪ k_fold=k_fold, n_jobs=n_jobs, random_state=0)
      dc_cv_bagging_subsampling.fit(Xtr, ytr)

[9]: DecisionCombinationCrossValBagging(base_estimator=OnlineGFMM(C=array([], dtype=float64),
      V=array([], dtype=float64),
      W=array([], dtype=float64)),
      base_estimator_params={'gamma': [0.5, 1, 2,
      4, 8, 16],
      'theta': array([0.05, 0.1 , 0.
      ↪ 15, 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 , 0.55,
      0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95, 1. ]),
      'theta_min': [1]},
      n_estimators=20, n_iter=20, n_jobs=4,
      random_state=0)

[10]: print("Training time: %.3f (s)"%(dc_cv_bagging_subsampling.elapsed_training_time))
      Training time: 46.519 (s)

[11]: print('Total number of hyperboxes from all base learners = %d'%dc_cv_bagging_subsampling.
      ↪ get_n_hyperboxes())
      Total number of hyperboxes from all base learners = 1168
```

## Prediction

```
[12]: y_pred = dc_cv_bagging_subsampling.predict(X_test)
      acc = accuracy_score(y_test, y_pred)
      print(f'Testing accuracy = {acc * 100: .2f}%')
      Testing accuracy = 94.74%
```

## Apply pruning for base learners

```
[13]: acc_threshold=0.5 # minimum accuracy score of the unpruned hyperboxes
      keep_empty_boxes=False # False means hyperboxes that do not join the prediction process_
      ↪ within the pruning procedure are also eliminated
      dc_cv_bagging_subsampling.simple_pruning_base_estimators(X_val, y_val, acc_threshold,
      ↪ keep_empty_boxes)

[13]: DecisionCombinationCrossValBagging(base_estimator=OnlineGFMM(C=array([], dtype=float64),
      V=array([], dtype=float64),
      W=array([], dtype=float64)),
```

(continues on next page)

(continued from previous page)

```

base_estimator_params={'gamma': [0.5, 1, 2,
                                4, 8, 16],
                      'theta': array([0.05, 0.1, 0.
→15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55,
                                0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 1. ]),
                      'theta_min': [1]},
n_estimators=20, n_iter=20, n_jobs=4,
random_state=0)

```

```

[14]: print('Total number of hyperboxes from all base learners after pruning = %d'%dc_cv_
→bagging_subsampling.get_n_hyperboxes())

```

```
Total number of hyperboxes from all base learners after pruning = 756
```

### Prediction after doing a pruning procedure

```

[15]: y_pred_2 = dc_cv_bagging_subsampling.predict(X_test)
acc_pruned = accuracy_score(y_test, y_pred_2)
print(f'Testing accuracy (after pruning) = {acc_pruned * 100: .2f}%')

```

```
Testing accuracy (after pruning) = 96.49%
```

## 2. Using random undersampling to generate class-balanced training sets for various base learners

### Training

```

[16]: # Initialise parameters
n_estimators = 20 # number of base learners
max_samples = 0.5 # sampling rate for samples
bootstrap = False # random subsampling without replacement
class_balanced = True # use the class-balanced sampling mode
n_jobs = 4 # number of processes is used to build base learners
n_iter = 20 # Number of parameter settings that are randomly sampled to choose the best_
→combination of hyperparameters
k_fold = 5 # Number of folds to conduct Stratified K-Fold cross-validation for_
→hyperparameter tuning

```

```

[17]: # Init a hyperbox-based model used to train base learners
# Using the GFMM classifier with the original online learning algorithm with the maximum_
→hyperbox size 0.1
base_estimator = OnlineGFMM()

```

```

[18]: # Init ranges for hyperparameters of base learners to perform a random search process_
→for hyperparameter tuning
base_estimator_params = {'theta': np.arange(0.05, 1.01, 0.05), 'theta_min': [1], 'gamma':
→[0.5, 1, 2, 4, 8, 16]}

```

```
[19]: dc_cv_bagging_class_balanced = DecisionCombinationCrossValBagging(base_estimator=base_
↳ estimator, base_estimator_params=base_estimator_params, n_estimators=n_estimators, max_
↳ samples=max_samples, bootstrap=bootstrap, class_balanced=class_balanced, n_iter=n_iter,
↳ k_fold=k_fold, n_jobs=n_jobs, random_state=0)
dc_cv_bagging_class_balanced.fit(Xtr, ytr)
```

```
[19]: DecisionCombinationCrossValBagging(base_estimator=OnlineGFMM(C=array([], dtype=float64),
V=array([], dtype=float64),
W=array([], dtype=float64)),
base_estimator_params={'gamma': [0.5, 1, 2,
4, 8, 16],
'theta': array([0.05, 0.1 , 0.
↳ 15, 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 , 0.55,
0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95, 1. ]),
'theta_min': [1]},
class_balanced=True, n_estimators=20,
n_iter=20, n_jobs=4, random_state=0)
```

```
[20]: print("Training time: %.3f (s)"%(dc_cv_bagging_class_balanced.elapsed_training_time))
```

```
Training time: 32.595 (s)
```

```
[21]: print('Total number of hyperboxes from all base learners = %d'%dc_cv_bagging_class_
↳ balanced.get_n_hyperboxes())
```

```
Total number of hyperboxes from all base learners = 1407
```

## Prediction

```
[22]: y_pred = dc_cv_bagging_class_balanced.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print(f'Testing accuracy = {acc * 100: .2f}%')
```

```
Testing accuracy = 94.74%
```

## Apply pruning for base learners

```
[23]: acc_threshold=0.5 # minimum accuracy score of the unpruned hyperboxes
keep_empty_boxes=False # False means hyperboxes that do not join the prediction process_
↳ within the pruning procedure are also eliminated
dc_cv_bagging_class_balanced.simple_pruning_base_estimators(X_val, y_val, acc_threshold,
↳ keep_empty_boxes)
```

```
[23]: DecisionCombinationCrossValBagging(base_estimator=OnlineGFMM(C=array([], dtype=float64),
V=array([], dtype=float64),
W=array([], dtype=float64)),
base_estimator_params={'gamma': [0.5, 1, 2,
4, 8, 16],
'theta': array([0.05, 0.1 , 0.
↳ 15, 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 , 0.55,
0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95, 1. ]),
```

(continues on next page)

(continued from previous page)

```
'theta_min': [1]],
class_balanced=True, n_estimators=20,
n_iter=20, n_jobs=4, random_state=0)
```

```
[24]: print('Total number of hyperboxes from all base learners after pruning = %d'%dc_cv_
↳ bagging_class_balanced.get_n_hyperboxes())
```

```
Total number of hyperboxes from all base learners after pruning = 719
```

### Prediction after doing a pruning procedure

```
[25]: y_pred_2 = dc_cv_bagging_class_balanced.predict(X_test)
acc_pruned = accuracy_score(y_test, y_pred_2)
print(f'Testing accuracy (after pruning) = {acc_pruned * 100: .2f}%')
```

```
Testing accuracy (after pruning) = 95.61%
```

### Model-level Bagging of Hyperbox-based Models

This example shows how to use a Bagging classifier with a combination at the model level to generate a single model from many base learners, in which each base hyperbox-based model is trained on a full set of features and a subset of samples.

```
[1]: import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from hbbrain.numerical_data.ensemble_learner.model_comb_bagging import _
↳ ModelCombinationBagging
from hbbrain.numerical_data.incremental_learner.onln_gfmm import OnlineGFMM
from hbbrain.numerical_data.batch_learner.accel_agglo_gfmm import _
↳ AccelAgglomerativeLearningGFMM
```

### Load dataset.

This example will use the breast cancer dataset available in sklearn to demonstrate how to use this ensemble classifier.

```
[2]: from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import MinMaxScaler
```

```
[3]: df = load_breast_cancer()
X = df.data
y = df.target
```

```
[4]: # Normalise data into the range of [0, 1] as hyperbox-based models only work in the _
↳ unit cube
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
```



```
[5]: # Split data into training, validation and testing sets
Xtr_val, X_test, ytr_val, y_test = train_test_split(X, y, train_size=0.8, random_state=0)
Xtr, X_val, ytr, y_val = train_test_split(X, y, train_size=0.75, random_state=0)
```

This example will use the GFMM classifier with the original online learning algorithm as base learners. However, any type of hyperbox-based learning algorithms in this library can also be used to train base learners.

## 1. Using random subsampling to generate training sets for various base learners

### a. Training without pruning for base learners

```
[6]: # Initialise parameters
n_estimators = 20 # number of base learners
max_samples = 0.5 # sampling rate for samples
bootstrap = False # random subsampling without replacement
class_balanced = False # do not use the class-balanced sampling mode
n_jobs = 4 # number of processes is used to build base learners

[7]: # Init a hyperbox-based model used to train base learners
# Using the GFMM classifier with the original online learning algorithm with the maximum_
↳ hyperbox size 0.1
base_estimator = OnlineGFMM(theta=0.1)

[8]: # Init a hyperbox-based model used to aggregate the resulting hyperboxes from all base_
↳ learners
# Using the accelerated agglomerative learning algorithm for the GFMM model to do this_
↳ task
model_level_estimator = AccelAgglomerativeLearningGFMM(theta=0.1, min_simil=0, simil_
↳ measure='long')

[9]: model_comb_bagging_subsampling = ModelCombinationBagging(base_estimator=base_estimator,
↳ model_level_estimator=model_level_estimator, n_estimators=n_estimators, max_
↳ samples=max_samples, bootstrap=bootstrap, class_balanced=class_balanced, n_jobs=n_jobs,
↳ random_state=0)
model_comb_bagging_subsampling.fit(Xtr, ytr)

[9]: ModelCombinationBagging(base_estimator=OnlineGFMM(C=array([], dtype=float64),
V=array([], dtype=float64),
W=array([], dtype=float64),
theta=0.1),
model_level_estimator=AccelAgglomerativeLearningGFMM(min_simil=0,
simil_
↳ measure='long',
theta=0.1),
n_estimators=20, n_jobs=4, random_state=0)

[10]: print("Training time: %.3f (s)"%(model_comb_bagging_subsampling.elapsed_training_time))
Training time: 16.647 (s)
```

```
[11]: print('Total number of hyperboxes in all base learners = %d'%model_comb_bagging_
↳subsampling.get_n_hyperboxes())
```

```
Total number of hyperboxes in all base learners = 3948
```

```
[12]: print('Number of hyperboxes in the combined model = %d'%model_comb_bagging_subsampling.
↳get_n_hyperboxes_comb_model())
```

```
Number of hyperboxes in the combined model = 401
```

## Prediction

### Using majority voting from predicted results of all base learners

```
[13]: y_pred_voting = model_comb_bagging_subsampling.predict_voting(X_test)
```

```
[14]: acc_voting = accuracy_score(y_test, y_pred_voting)
print(f'Testing accuracy using voting of decisions from base learners = {acc_voting *
↳100 : .2f}%')
```

```
Testing accuracy using voting of decisions from base learners = 93.86%
```

### Using the final combined single model to make prediction

```
[16]: y_pred = model_comb_bagging_subsampling.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print(f'Testing accuracy of the combined model = {acc * 100: .2f}%')
```

```
Testing accuracy of the combined model = 92.98%
```

### Apply pruning for the final combined model

```
[17]: acc_threshold=0.5 # minimum accuracy score of the unpruned hyperboxes
keep_empty_boxes=False # False means hyperboxes that do not join the prediction process,
↳within the pruning procedure are also eliminated
model_comb_bagging_subsampling.simple_pruning(X_val, y_val, acc_threshold, keep_empty_
↳boxes)
```

```
[17]: ModelCombinationBagging(base_estimator=OnlineGFMM(C=array([], dtype=float64),
V=array([], dtype=float64),
W=array([], dtype=float64),
theta=0.1),
model_level_estimator=AccelAgglomerativeLearningGFMM(min_simil=0,
simil_
↳measure='long',
theta=0.1),
n_estimators=20, n_jobs=4, random_state=0)
```

```
[18]: print('Number of hyperboxes of the combined single model after pruning = %d'%model_comb_
      ↪ bagging_subsampling.get_n_hyperboxes_comb_model())
```

Number of hyperboxes of the combined single model after pruning = 393

### Prediction after doing a pruning procedure for the combined single model

```
[20]: y_pred_2 = model_comb_bagging_subsampling.predict(X_test)
      acc_pruned = accuracy_score(y_test, y_pred_2)
      print(f'Testing accuracy after pruning the final model = {acc_pruned * 100: .2f}%')
```

Testing accuracy after pruning the final model = 94.74%

### b. Training with pruning for base learners

```
[21]: model_comb_bagging_subsampling_base_learner_pruning = ModelCombinationBagging(base_
      ↪ estimator=base_estimator, model_level_estimator=model_level_estimator, n_estimators=n_
      ↪ estimators, max_samples=max_samples, bootstrap=bootstrap, class_balanced=class_
      ↪ balanced, n_jobs=n_jobs, random_state=0)
      model_comb_bagging_subsampling_base_learner_pruning.fit(Xtr, ytr, is_pruning_base_
      ↪ learners=True, X_val=X_val, y_val=y_val, acc_threshold=acc_threshold, keep_empty_
      ↪ boxes=keep_empty_boxes)
```

```
[21]: ModelCombinationBagging(base_estimator=OnlineGFMM(C=array([], dtype=float64),
      V=array([], dtype=float64),
      W=array([], dtype=float64),
      theta=0.1),
      model_level_estimator=AccelAgglomerativeLearningGFMM(min_simil=0,
      simil_
      ↪ measure='long',
      theta=0.1),
      n_estimators=20, n_jobs=4, random_state=0)
```

```
[22]: print("Training time: %.3f (s)"%(model_comb_bagging_subsampling_base_learner_pruning.
      ↪ elapsed_training_time))
```

Training time: 8.254 (s)

```
[23]: print('Total number of hyperboxes in all base learners = %d'%model_comb_bagging_
      ↪ subsampling_base_learner_pruning.get_n_hyperboxes())
```

Total number of hyperboxes in all base learners = 2195

```
[24]: print('Number of hyperboxes in the combined model = %d'%model_comb_bagging_subsampling_
      ↪ base_learner_pruning.get_n_hyperboxes_comb_model())
```

Number of hyperboxes in the combined model = 388

## Prediction

### Using majority voting from predicted results of all base learners

```
[25]: y_pred_voting = model_comb_bagging_subsampling_base_learner_pruning.predict_voting(X_
      ↪ test)
```

```
[26]: acc_voting = accuracy_score(y_test, y_pred_voting)
      print(f'Testing accuracy using voting of decisions from base learners = {acc_voting * 100 : .2f}%')
```

```
Testing accuracy using voting of decisions from base learners = 95.61%
```

### Using the final combined single model to make prediction

```
[27]: y_pred = model_comb_bagging_subsampling_base_learner_pruning.predict(X_test)
      acc = accuracy_score(y_test, y_pred)
      print(f'Testing accuracy of the combined model = {acc * 100 : .2f}%')
```

```
Testing accuracy of the combined model = 94.74%
```

### Apply pruning for the final combined model

```
[28]: acc_threshold=0.5 # minimum accuracy score of the unpruned hyperboxes
      keep_empty_boxes=False # False means hyperboxes that do not join the prediction process_
      ↪ within the pruning procedure are also eliminated
      model_comb_bagging_subsampling_base_learner_pruning.simple_pruning(X_val, y_val, acc_
      ↪ threshold, keep_empty_boxes)
```

```
[28]: ModelCombinationBagging(base_estimator=OnlineGFMM(C=array([], dtype=float64),
      V=array([], dtype=float64),
      W=array([], dtype=float64),
      theta=0.1),
      model_level_estimator=AccelAgglomerativeLearningGFMM(min_simil=0,
      simil_
      ↪ measure='long',
      theta=0.1),
      n_estimators=20, n_jobs=4, random_state=0)
```

```
[29]: print('Number of hyperboxes of the combined single model after pruning = %d'%model_comb_
      ↪ bagging_subsampling_base_learner_pruning.get_n_hyperboxes_comb_model())
```

```
Number of hyperboxes of the combined single model after pruning = 383
```

## Prediction after doing a pruning procedure for the combined single model

```
[30]: y_pred_2 = model_comb_bagging_subsampling_base_learner_pruning.predict(X_test)
acc_pruned = accuracy_score(y_test, y_pred_2)
print(f'Testing accuracy after pruning the final model = {acc_pruned * 100: .2f}%')
```

```
Testing accuracy after pruning the final model = 94.74%
```

## 2. Using random undersampling to generate class-balanced training sets for various base learners

### a. Training without pruning for base learners

```
[31]: # Initialise parameters
n_estimators = 20 # number of base learners
max_samples = 0.5 # sampling rate for samples
bootstrap = False # random subsampling without replacement
class_balanced = True # use the class-balanced sampling mode
n_jobs = 4 # number of processes is used to build base learners
```

```
[32]: # Init a hyperbox-based model used to train base learners
# Using the GFMM classifier with the original online learning algorithm with the maximum_
↳ hyperbox size 0.1
base_estimator = OnlineGFMM(theta=0.1)
```

```
[33]: # Init a hyperbox-based model used to aggregate the resulting hyperboxes from all base_
↳ learners
# Using the accelerated agglomerative learning algorithm for the GFMM model to do this_
↳ task
model_level_estimator = AccelAgglomerativeLearningGFMM(theta=0.1, min_simil=0, simil_
↳ measure='long')
```

```
[34]: model_comb_bagging_class_balanced = ModelCombinationBagging(base_estimator=base_
↳ estimator, model_level_estimator=model_level_estimator, n_estimators=n_estimators, max_
↳ samples=max_samples, bootstrap=bootstrap, class_balanced=class_balanced, n_jobs=n_jobs,
↳ random_state=0)
model_comb_bagging_class_balanced.fit(Xtr, ytr)
```

```
[34]: ModelCombinationBagging(base_estimator=OnlineGFMM(C=array([], dtype=float64),
V=array([], dtype=float64),
W=array([], dtype=float64),
theta=0.1),
class_balanced=True,
model_level_estimator=AccelAgglomerativeLearningGFMM(min_simil=0,
simil_
↳ measure='long',
theta=0.1),
n_estimators=20, n_jobs=4, random_state=0)
```

```
[35]: print("Training time: %.3f (s)"%(model_comb_bagging_class_balanced.elapsed_training_
↳ time))
```

```
Training time: 16.955 (s)
```

```
[36]: print('Total number of hyperboxes in all base learners = %d'%model_comb_bagging_class_
↳ balanced.get_n_hyperboxes())
```

```
Total number of hyperboxes in all base learners = 4010
```

```
[37]: print('Number of hyperboxes in the combined model = %d'%model_comb_bagging_class_
↳ balanced.get_n_hyperboxes_comb_model())
```

```
Number of hyperboxes in the combined model = 400
```

## Prediction

### Using majority voting from predicted results of all base learners

```
[38]: y_pred_voting = model_comb_bagging_class_balanced.predict_voting(X_test)
```

```
[39]: acc_voting = accuracy_score(y_test, y_pred_voting)
print(f'Testing accuracy using voting of decisions from base learners = {acc_voting *
↳ 100 : .2f}%')
```

```
Testing accuracy using voting of decisions from base learners = 92.11%
```

### Using the final combined single model to make prediction

```
[40]: y_pred = model_comb_bagging_class_balanced.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print(f'Testing accuracy of the combined model = {acc * 100: .2f}%')
```

```
Testing accuracy of the combined model = 92.98%
```

### Apply pruning for the final combined model

```
[41]: acc_threshold=0.5 # minimum accuracy score of the unpruned hyperboxes
keep_empty_boxes=False # False means hyperboxes that do not join the prediction process.
↳ within the pruning procedure are also eliminated
model_comb_bagging_class_balanced.simple_pruning(X_val, y_val, acc_threshold, keep_empty_
↳ boxes)
```

```
[41]: ModelCombinationBagging(base_estimator=OnlineGFMM(C=array([], dtype=float64),
V=array([], dtype=float64),
W=array([], dtype=float64),
theta=0.1),
class_balanced=True,
model_level_estimator=AccelAgglomerativeLearningGFMM(min_simil=0,
simil_
↳ measure='long',
```

(continues on next page)

(continued from previous page)

```

n_estimators=20, n_jobs=4, random_state=0)
theta=0.1),

```

```

[42]: print('Number of hyperboxes of the combined single model after pruning = %d'%model_comb_
      ↪ bagging_class_balanced.get_n_hyperboxes_comb_model())

```

```

Number of hyperboxes of the combined single model after pruning = 392

```

### Prediction after doing a pruning procedure for the combined single model

```

[43]: y_pred_2 = model_comb_bagging_class_balanced.predict(X_test)
      ↪ acc_pruned = accuracy_score(y_test, y_pred_2)
      ↪ print(f'Testing accuracy after pruning the final model = {acc_pruned * 100: .2f}%')

```

```

Testing accuracy after pruning the final model = 94.74%

```

### b. Training with pruning for base learners

```

[44]: model_comb_bagging_class_balanced_base_learner_pruning = ModelCombinationBagging(base_
      ↪ estimator=base_estimator, model_level_estimator=model_level_estimator, n_estimators=n_
      ↪ estimators, max_samples=max_samples, bootstrap=bootstrap, class_balanced=class_
      ↪ balanced, n_jobs=n_jobs, random_state=0)
      ↪ model_comb_bagging_class_balanced_base_learner_pruning.fit(Xtr, ytr, is_pruning_base_
      ↪ learners=True, X_val=X_val, y_val=y_val, acc_threshold=acc_threshold, keep_empty_
      ↪ boxes=keep_empty_boxes)

```

```

[44]: ModelCombinationBagging(base_estimator=OnlineGFMM(C=array([], dtype=float64),
      ↪ V=array([], dtype=float64),
      ↪ W=array([], dtype=float64),
      ↪ theta=0.1),
      ↪ class_balanced=True,
      ↪ model_level_estimator=AccelAgglomerativeLearningGFMM(min_simil=0,
      ↪ simil_
      ↪ measure='long',
      ↪ theta=0.1),
      ↪ n_estimators=20, n_jobs=4, random_state=0)

```

```

[45]: print("Training time: %.3f (s)"%(model_comb_bagging_class_balanced_base_learner_pruning.
      ↪ elapsed_training_time))

```

```

Training time: 7.264 (s)

```

```

[46]: print('Total number of hyperboxes in all base learners = %d'%model_comb_bagging_class_
      ↪ balanced_base_learner_pruning.get_n_hyperboxes())

```

```

Total number of hyperboxes in all base learners = 2738

```

```

[47]: print('Number of hyperboxes in the combined model = %d'%model_comb_bagging_class_
      ↪ balanced_base_learner_pruning.get_n_hyperboxes_comb_model())

```

```
Number of hyperboxes in the combined model = 395
```

## Prediction

### Using majority voting from predicted results of all base learners

```
[48]: y_pred_voting = model_comb_bagging_class_balanced_base_learner_pruning.predict_voting(X_
↳ test)
```

```
[49]: acc_voting = accuracy_score(y_test, y_pred_voting)
print(f'Testing accuracy using voting of decisions from base learners = {acc_voting *
↳ 100 : .2f}%')
```

```
Testing accuracy using voting of decisions from base learners = 94.74%
```

### Using the final combined single model to make prediction

```
[50]: y_pred = model_comb_bagging_class_balanced_base_learner_pruning.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print(f'Testing accuracy of the combined model = {acc * 100: .2f}%')
```

```
Testing accuracy of the combined model = 94.74%
```

### Apply pruning for the final combined model

```
[51]: acc_threshold=0.5 # minimum accuracy score of the unpruned hyperboxes
keep_empty_boxes=False # False means hyperboxes that do not join the prediction process,
↳ within the pruning procedure are also eliminated
model_comb_bagging_class_balanced_base_learner_pruning.simple_pruning(X_val, y_val, acc_
↳ threshold, keep_empty_boxes)
```

```
[51]: ModelCombinationBagging(base_estimator=OnlineGFMM(C=array([], dtype=float64),
V=array([], dtype=float64),
W=array([], dtype=float64),
theta=0.1),
class_balanced=True,
model_level_estimator=AccelAgglomerativeLearningGFMM(min_simil=0,
simil_
↳ measure='long',
theta=0.1),
n_estimators=20, n_jobs=4, random_state=0)
```

```
[52]: print('Number of hyperboxes of the combined single model after pruning = %d'%model_comb_
↳ bagging_class_balanced_base_learner_pruning.get_n_hyperboxes_comb_model())
```

```
Number of hyperboxes of the combined single model after pruning = 100
```



### Prediction after doing a pruning procedure for the combined single model

```
[53]: y_pred_2 = model_comb_bagging_class_balanced_base_learner_pruning.predict(X_test)
acc_pruned = accuracy_score(y_test, y_pred_2)
print(f'Testing accuracy after pruning the final model = {acc_pruned * 100: .2f}%')

Testing accuracy after pruning the final model = 94.74%
```

### Model-level Bagging of Hyperbox-based Learners with Hyper-parameter Optimisation

This example shows how to use a Bagging classifier with a combination at the model level to generate a single model from many base learners, in which each base learner is trained by random search-based hyper-parameter tuning and k-fold cross-validation.

While the original model-level combination bagging classifier in the class `ModelCombinationBagging` uses the same base learners with the same hyperparameters, the cross-validation model-level combination bagging classifier in the class `ModelCombinationCrossValBagging` allows each base learner to use specific hyperparameters depending on the training data by performing random research to find the best combination of hyperparameters for each base learner.

```
[1]: import warnings
warnings.filterwarnings('ignore')
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from hbbbrain.numerical_data.ensemble_learner.model_comb_cross_val_bagging import _
↳ ModelCombinationCrossValBagging
from hbbbrain.numerical_data.incremental_learner.onln_gfmm import OnlineGFMM
from hbbbrain.numerical_data.batch_learner.accel_agglo_gfmm import _
↳ AccelAgglomerativeLearningGFMM
```

**Load dataset.** This example will use the breast cancer dataset available in sklearn to demonstrate how to use this ensemble classifier.

```
[2]: from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import MinMaxScaler
```

```
[3]: df = load_breast_cancer()
X = df.data
y = df.target
```

```
[4]: # Normailise data into the range of [0, 1] as hyperbox-based models only work in the _
↳ unit cube
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
```

```
[5]: # Split data into training, validation and testing sets
Xtr_val, X_test, ytr_val, y_test = train_test_split(X, y, train_size=0.8, random_state=0)
Xtr, X_val, ytr, y_val = train_test_split(X, y, train_size=0.75, random_state=0)
```

This example will use the GFMM classifier with the original online learning algorithm as base learners. However, any type of hyperbox-based learning algorithms in this library can also be used to train base learners.

## 1. Using random subsampling to generate training sets for various base learners

### a. Training without pruning for base learners

```
[6]: # Initialise parameters
n_estimators = 20 # number of base learners
max_samples = 0.5 # sampling rate for samples
bootstrap = False # random subsampling without replacement
class_balanced = False # do not use the class-balanced sampling mode
n_jobs = 4 # number of processes is used to build base learners
n_iter = 20 # Number of parameter settings that are randomly sampled to choose the best
↳ combination of hyperparameters
k_fold = 5 # Number of folds to conduct Stratified K-Fold cross-validation for
↳ hyperparameter tuning

[7]: # Init a hyperbox-based model used to train base learners
# Using the GFMM classifier with the original online learning algorithm
base_estimator = OnlineGFMM()

[8]: # Init ranges for hyperparameters of base learners to perform a random search process
↳ for hyperparameter tuning
base_estimator_params = {'theta': np.arange(0.05, 1.01, 0.05), 'theta_min': [1], 'gamma':
↳ [0.5, 1, 2, 4, 8, 16]}

[9]: # Init a hyperbox-based model used to aggregate the resulting hyperboxes from all base
↳ learners
# Using the accelerated agglomerative learning algorithm for the GFMM model to do this
↳ task
model_level_estimator = AccelAgglomerativeLearningGFMM(theta=0.1, min_simil=0, simil_
↳ measure='long')

[10]: model_comb_cv_bagging_subsampling = ModelCombinationCrossValBagging(base_estimator=base_
↳ estimator, base_estimator_params=base_estimator_params, model_level_estimator=model_
↳ level_estimator, n_estimators=n_estimators, max_samples=max_samples,
↳ bootstrap=bootstrap, class_balanced=class_balanced, n_iter=n_iter, k_fold=k_fold, n_
↳ jobs=n_jobs, random_state=0)
model_comb_cv_bagging_subsampling.fit(Xtr, ytr)

[10]: ModelCombinationCrossValBagging(base_estimator=OnlineGFMM(C=array([], dtype=float64),
V=array([], dtype=float64),
W=array([], dtype=float64)),
base_estimator_params={'gamma': [0.5, 1, 2, 4,
8, 16],
'theta': array([0.05, 0.1 , 0.15,
↳ 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 , 0.55,
0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95, 1. ]),
'theta_min': [1]},
model_level_estimator=AccelAgglomerativeLearningGFMM(min_
↳ simil=0,
↳ simil_measure='long',
```

(continues on next page)

(continued from previous page)

```

↪ theta=0.1),
                                n_estimators=20, n_iter=20, n_jobs=4,
                                random_state=0)

```

```

[11]: print("Training time: %.3f (s)"%(model_comb_cv_bagging_subsampling.elapsed_training_
↪ time))

```

```

Training time: 44.155 (s)

```

```

[12]: print('Total number of hyperboxes in all base learners = %d'%model_comb_cv_bagging_
↪ subsampling.get_n_hyperboxes())

```

```

Total number of hyperboxes in all base learners = 1168

```

```

[13]: print('Number of hyperboxes in the combined model = %d'%model_comb_cv_bagging_
↪ subsampling.get_n_hyperboxes_comb_model())

```

```

Number of hyperboxes in the combined model = 779

```

## Prediction

### Using majority voting from predicted results of all base learners

```

[14]: y_pred_voting = model_comb_cv_bagging_subsampling.predict_voting(X_test)

```

```

[15]: acc_voting = accuracy_score(y_test, y_pred_voting)
print(f'Testing accuracy using voting of decisions from base learners = {acc_voting * 100 : .2f}%')
↪

```

```

Testing accuracy using voting of decisions from base learners = 94.74%

```

### Using the final combined single model to make prediction

```

[16]: y_pred = model_comb_cv_bagging_subsampling.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print(f'Testing accuracy of the combined model = {acc * 100 : .2f}%')

```

```

Testing accuracy of the combined model = 88.60%

```

### Apply pruning for the final combined model

```

[17]: acc_threshold=0.5 # minimum accuracy score of the unpruned hyperboxes
keep_empty_boxes=False # False means hyperboxes that do not join the prediction process_
↪ within the pruning procedure are also eliminated
model_comb_cv_bagging_subsampling.simple_pruning(X_val, y_val, acc_threshold, keep_empty_
↪ boxes)

```

```
[17]: ModelCombinationCrossValBagging(base_estimator=OnlineGFMM(C=array([], dtype=float64),
                                V=array([], dtype=float64),
                                W=array([], dtype=float64)),
                                base_estimator_params={'gamma': [0.5, 1, 2, 4,
                                                                8, 16],
                                'theta': array([0.05, 0.1 , 0.15,
↪0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 , 0.55,
                                0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95, 1. ]),
                                'theta_min': [1]}],
                                model_level_estimator=AccelAgglomerativeLearningGFMM(min_
↪simil=0,
↪simil_measure='long',
↪theta=0.1),
                                n_estimators=20, n_iter=20, n_jobs=4,
                                random_state=0)
```

```
[18]: print('Number of hyperboxes of the combined single model after pruning = %d'%model_comb_
↪cv_bagging_subsampling.get_n_hyperboxes_comb_model())
```

Number of hyperboxes of the combined single model after pruning = 36

### Prediction after doing a pruning procedure for the combined single model

```
[19]: y_pred_2 = model_comb_cv_bagging_subsampling.predict(X_test)
acc_pruned = accuracy_score(y_test, y_pred_2)
print(f'Testing accuracy after pruning the final model = {acc_pruned * 100: .2f}%')
```

Testing accuracy after pruning the final model = 88.60%

### b. Training with pruning for base learners

```
[20]: model_comb_cv_bagging_subsampling_base_learner_pruning =
↪ModelCombinationCrossValBagging(base_estimator=base_estimator, base_estimator_
↪params=base_estimator_params, model_level_estimator=model_level_estimator, n_
↪estimators=n_estimators, max_samples=max_samples, bootstrap=bootstrap, class_
↪balanced=class_balanced, n_iter=n_iter, k_fold=k_fold, n_jobs=n_jobs, random_state=0)
model_comb_cv_bagging_subsampling_base_learner_pruning.fit(Xtr, ytr, is_pruning_base_
↪learners=True, X_val=X_val, y_val=y_val, acc_threshold=acc_threshold, keep_empty_
↪boxes=keep_empty_boxes)
```

```
[20]: ModelCombinationCrossValBagging(base_estimator=OnlineGFMM(C=array([], dtype=float64),
                                V=array([], dtype=float64),
                                W=array([], dtype=float64)),
                                base_estimator_params={'gamma': [0.5, 1, 2, 4,
                                                                8, 16],
                                'theta': array([0.05, 0.1 , 0.15,
↪0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 , 0.55,
                                0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95, 1. ]),
```

(continues on next page)

(continued from previous page)

```

        'theta_min': [1]},
        model_level_estimator=AccelAgglomerativeLearningGFMM(min_
↪simil=0,
        ↪simil_measure='long',
        ↪theta=0.1),
        n_estimators=20, n_iter=20, n_jobs=4,
        random_state=0)

```

```
[21]: print("Training time: %.3f (s)"%(model_comb_cv_bagging_subsampling_base_learner_pruning.
↪elapsed_training_time))
```

```
Training time: 44.437 (s)
```

```
[22]: print('Total number of hyperboxes in all base learners = %d'%model_comb_cv_bagging_
↪subsampling_base_learner_pruning.get_n_hyperboxes())
```

```
Total number of hyperboxes in all base learners = 756
```

```
[23]: print('Number of hyperboxes in the combined model = %d'%model_comb_cv_bagging_
↪subsampling_base_learner_pruning.get_n_hyperboxes_comb_model())
```

```
Number of hyperboxes in the combined model = 613
```

## Prediction

### Using majority voting from predicted results of all base learners

```
[24]: y_pred_voting = model_comb_cv_bagging_subsampling_base_learner_pruning.predict_voting(X_
↪test)
```

```
[25]: acc_voting = accuracy_score(y_test, y_pred_voting)
print(f'Testing accuracy using voting of decisions from base learners = {acc_voting *
↪100 : .2f}%')
```

```
Testing accuracy using voting of decisions from base learners = 96.49%
```

### Using the final combined single model to make prediction

```
[26]: y_pred = model_comb_cv_bagging_subsampling_base_learner_pruning.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print(f'Testing accuracy of the combined model = {acc * 100: .2f}%')
```

```
Testing accuracy of the combined model = 88.60%
```

## Apply pruning for the final combined model

```
[27]: acc_threshold=0.5 # minimum accuracy score of the unpruned hyperboxes
keep_empty_boxes=False # False means hyperboxes that do not join the prediction process,
↳ within the pruning procedure are also eliminated
model_comb_cv_bagging_subsampling_base_learner_pruning.simple_pruning(X_val, y_val, acc_
↳ threshold, keep_empty_boxes)

[27]: ModelCombinationCrossValBagging(base_estimator=OnlineGFMM(C=array([], dtype=float64),
                                V=array([], dtype=float64),
                                W=array([], dtype=float64)),
                                base_estimator_params={'gamma': [0.5, 1, 2, 4,
                                                                8, 16],
                                                        'theta': array([0.05, 0.1 , 0.15,
↳ 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 , 0.55,
                                0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95, 1. ]),
                                                        'theta_min': [1]}),
                                model_level_estimator=AccelAgglomerativeLearningGFMM(min_
↳ simil=0,
                                simil_measure='long',
                                theta=0.1),
                                n_estimators=20, n_iter=20, n_jobs=4,
                                random_state=0)

[28]: print('Number of hyperboxes of the combined single model after pruning = %d'%model_comb_
↳ cv_bagging_subsampling_base_learner_pruning.get_n_hyperboxes_comb_model())
Number of hyperboxes of the combined single model after pruning = 36
```

## Prediction after doing a pruning procedure for the combined single model

```
[29]: y_pred_2 = model_comb_cv_bagging_subsampling_base_learner_pruning.predict(X_test)
acc_pruned = accuracy_score(y_test, y_pred_2)
print(f'Testing accuracy after pruning the final model = {acc_pruned * 100: .2f}%')
Testing accuracy after pruning the final model = 88.60%
```

## 2. Using random undersampling to generate class-balanced training sets for various base learners

### a. Training without pruning for base learners

```
[30]: # Initialise parameters
n_estimators = 20 # number of base learners
max_samples = 0.5 # sampling rate for samples
bootstrap = False # random subsampling without replacement
class_balanced = True # use the class-balanced sampling mode
n_jobs = 4 # number of processes is used to build base learners
n_iter = 20 # Number of parameter settings that are randomly sampled to choose the best_
```

(continues on next page)

(continued from previous page)

```

↪ combination of hyperparameters
k_fold = 5 # Number of folds to conduct Stratified K-Fold cross-validation for
↪ hyperparameter tuning

```

```

[31]: # Init a hyperbox-based model used to train base learners
      # Using the GFMM classifier with the original online learning algorithm
      base_estimator = OnlineGFMM()

```

```

[32]: # Init ranges for hyperparameters of base learners to perform a random search process
      ↪ for hyperparameter tuning
      base_estimator_params = {'theta': np.arange(0.05, 1.01, 0.05), 'theta_min': [1], 'gamma':
      ↪ [0.5, 1, 2, 4, 8, 16]}

```

```

[33]: # Init a hyperbox-based model used to aggregate the resulting hyperboxes from all base
      ↪ learners
      # Using the accelerated agglomerative learning algorithm for the GFMM model to do this
      ↪ task
      model_level_estimator = AccelAgglomerativeLearningGFMM(theta=0.1, min_simil=0, simil_
      ↪ measure='long')

```

```

[34]: model_comb_cv_bagging_class_balanced = ModelCombinationCrossValBagging(base_
      ↪ estimator=base_estimator, base_estimator_params=base_estimator_params, model_level_
      ↪ estimator=model_level_estimator, n_estimators=n_estimators, max_samples=max_samples,
      ↪ bootstrap=bootstrap, class_balanced=class_balanced, n_iter=n_iter, k_fold=k_fold, n_
      ↪ jobs=n_jobs, random_state=0)
      model_comb_cv_bagging_class_balanced.fit(Xtr, ytr)

```

```

[34]: ModelCombinationCrossValBagging(base_estimator=OnlineGFMM(C=array([], dtype=float64),
                                                                V=array([], dtype=float64),
                                                                W=array([], dtype=float64)),
                                     base_estimator_params={'gamma': [0.5, 1, 2, 4,
                                                                8, 16],
                                                                'theta': array([0.05, 0.1 , 0.15,
      ↪ 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 , 0.55,
                                                                0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95, 1. ]),
                                                                'theta_min': [1]},
                                     class_balanced=True,
                                     model_level_estimator=AccelAgglomerativeLearningGFMM(min_
      ↪ simil=0,
                                     simil_measure='long',
      ↪ theta=0.1),
                                     n_estimators=20, n_iter=20, n_jobs=4,
                                     random_state=0)

```

```

[35]: print("Training time: %.3f (s)"%(model_comb_cv_bagging_class_balanced.elapsed_training_
      ↪ time))

```

```

Training time: 36.885 (s)

```

```
[36]: print('Total number of hyperboxes in all base learners = %d'%model_comb_cv_bagging_class_
↳balanced.get_n_hyperboxes())
```

```
Total number of hyperboxes in all base learners = 1407
```

```
[37]: print('Number of hyperboxes in the combined model = %d'%model_comb_cv_bagging_class_
↳balanced.get_n_hyperboxes_comb_model())
```

```
Number of hyperboxes in the combined model = 812
```

## Prediction

### Using majority voting from predicted results of all base learners

```
[38]: y_pred_voting = model_comb_cv_bagging_class_balanced.predict_voting(X_test)
```

```
[39]: acc_voting = accuracy_score(y_test, y_pred_voting)
print(f'Testing accuracy using voting of decisions from base learners = {acc_voting * 100 : .2f}%')
```

```
Testing accuracy using voting of decisions from base learners = 94.74%
```

### Using the final combined single model to make prediction

```
[40]: y_pred = model_comb_cv_bagging_class_balanced.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print(f'Testing accuracy of the combined model = {acc * 100: .2f}%')
```

```
Testing accuracy of the combined model = 89.47%
```

## Apply pruning for the final combined model

```
[41]: acc_threshold=0.5 # minimum accuracy score of the unpruned hyperboxes
keep_empty_boxes=False # False means hyperboxes that do not join the prediction process_
↳within the pruning procedure are also eliminated
model_comb_cv_bagging_class_balanced.simple_pruning(X_val, y_val, acc_threshold, keep_
↳empty_boxes)
```

```
[41]: ModelCombinationCrossValBagging(base_estimator=OnlineGFMM(C=array([], dtype=float64),
V=array([], dtype=float64),
W=array([], dtype=float64)),
base_estimator_params={'gamma': [0.5, 1, 2, 4,
8, 16],
'theta': array([0.05, 0.1 , 0.15, 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 , 0.55,
0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95, 1. ]),
'theta_min': [1]}),
class_balanced=True,
model_level_estimator=AccelAgglomerativeLearningGFMM(min_
```

(continues on next page)



(continued from previous page)

```

↪simil=0,

↪simil_measure='long',

↪theta=0.1),

                                n_estimators=20, n_iter=20, n_jobs=4,
                                random_state=0)

```

```

[42]: print('Number of hyperboxes of the combined single model after pruning = %d'%model_comb_
↪cv_bagging_class_balanced.get_n_hyperboxes_comb_model())

```

```

Number of hyperboxes of the combined single model after pruning = 42

```

### Prediction after doing a pruning procedure for the combined single model

```

[43]: y_pred_2 = model_comb_cv_bagging_class_balanced.predict(X_test)
acc_pruned = accuracy_score(y_test, y_pred_2)
print(f'Testing accuracy after pruning the final model = {acc_pruned * 100: .2f}%')

```

```

Testing accuracy after pruning the final model = 90.35%

```

### b. Training with pruning for base learners

```

[45]: model_comb_cv_bagging_class_balanced_base_learner_pruning =
↪ModelCombinationCrossValBagging(base_estimator=base_estimator, base_estimator_
↪params=base_estimator_params, model_level_estimator=model_level_estimator, n_
↪estimators=n_estimators, max_samples=max_samples, bootstrap=bootstrap, class_
↪balanced=class_balanced, n_iter=n_iter, k_fold=k_fold, n_jobs=n_jobs, random_state=0)
model_comb_cv_bagging_class_balanced_base_learner_pruning.fit(Xtr, ytr, is_pruning_base_
↪learners=True, X_val=X_val, y_val=y_val, acc_threshold=acc_threshold, keep_empty_
↪boxes=keep_empty_boxes)

[45]: ModelCombinationCrossValBagging(base_estimator=OnlineGFMM(C=array([], dtype=float64),
                                V=array([], dtype=float64),
                                W=array([], dtype=float64)),
                                base_estimator_params={'gamma': [0.5, 1, 2, 4,
                                8, 16],
                                'theta': array([0.05, 0.1 , 0.15,
↪0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 , 0.55,
                                0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95, 1. ]),
                                'theta_min': [1]}},
                                class_balanced=True,
                                model_level_estimator=AccelAgglomerativeLearningGFMM(min_
↪simil=0,

↪simil_measure='long',

↪theta=0.1),

                                n_estimators=20, n_iter=20, n_jobs=4,
                                random_state=0)

```

```
[46]: print("Training time: %.3f (s)"%(model_comb_cv_bagging_class_balanced_base_learner_
      ↪ pruning.elapsed_training_time))
```

```
Training time: 31.609 (s)
```

```
[47]: print('Total number of hyperboxes in all base learners = %d'%model_comb_cv_bagging_class_
      ↪ balanced_base_learner_pruning.get_n_hyperboxes())
```

```
Total number of hyperboxes in all base learners = 719
```

```
[48]: print('Number of hyperboxes in the combined model = %d'%model_comb_cv_bagging_class_
      ↪ balanced_base_learner_pruning.get_n_hyperboxes_comb_model())
```

```
Number of hyperboxes in the combined model = 538
```

## Prediction

### Using majority voting from predicted results of all base learners

```
[50]: y_pred_voting = model_comb_cv_bagging_class_balanced_base_learner_pruning.predict_
      ↪ voting(X_test)
```

```
[51]: acc_voting = accuracy_score(y_test, y_pred_voting)
      print(f'Testing accuracy using voting of decisions from base learners = {acc_voting * 100 : .2f}%')
```

```
Testing accuracy using voting of decisions from base learners = 95.61%
```

### Using the final combined single model to make prediction

```
[52]: y_pred = model_comb_cv_bagging_class_balanced_base_learner_pruning.predict(X_test)
      acc = accuracy_score(y_test, y_pred)
      print(f'Testing accuracy of the combined model = {acc * 100: .2f}%')
```

```
Testing accuracy of the combined model = 89.47%
```

### Apply pruning for the final combined model

```
[53]: acc_threshold=0.5 # minimum accuracy score of the unpruned hyperboxes
      keep_empty_boxes=False # False means hyperboxes that do not join the prediction process_
      ↪ within the pruning procedure are also eliminated
      model_comb_cv_bagging_class_balanced_base_learner_pruning.simple_pruning(X_val, y_val,
      ↪ acc_threshold, keep_empty_boxes)
```

```
[53]: ModelCombinationCrossValBagging(base_estimator=OnlineGFMM(C=array([], dtype=float64),
      V=array([], dtype=float64),
      W=array([], dtype=float64)),
      base_estimator_params={'gamma': [0.5, 1, 2, 4,
      8, 16],
```

(continues on next page)

(continued from previous page)

```

                                'theta': array([0.05, 0.1 , 0.15,
↪0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 , 0.55,
                                0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95, 1. ]),
                                'theta_min': [1]},
                                class_balanced=True,
                                model_level_estimator=AccelAgglomerativeLearningGFMM(min_
↪simil=0,
                                cv_bagging_class_balanced_base_learner_pruning.get_n_hyperboxes_comb_model()
↪simil_measure='long',
                                theta=0.1),
                                n_estimators=20, n_iter=20, n_jobs=4,
                                random_state=0)

```

```

[54]: print('Number of hyperboxes of the combined single model after pruning = %d'%model_comb_
↪cv_bagging_class_balanced_base_learner_pruning.get_n_hyperboxes_comb_model())

```

```

Number of hyperboxes of the combined single model after pruning = 42

```

### Prediction after doing a pruning procedure for the combined single model

```

[55]: y_pred_2 = model_comb_cv_bagging_class_balanced_base_learner_pruning.predict(X_test)
acc_pruned = accuracy_score(y_test, y_pred_2)
print(f'Testing accuracy after pruning the final model = {acc_pruned * 100: .2f}%')

```

```

Testing accuracy after pruning the final model = 90.35%

```

### Random Hyperboxes

This example shows how to use a random hyperboxes classifier, in which each base hyperbox-based model is trained on a subset of features and a subset of samples.

```

[1]: import warnings
warnings.filterwarnings('ignore')
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from hbbrain.numerical_data.ensemble_learner.random_hyperboxes import
↪RandomHyperboxesClassifier
from hbbrain.numerical_data.incremental_learner.onln_gfmm import OnlineGFMM

```

## Load dataset.

This example will use the breast cancer dataset available in sklearn to demonstrate how to use this ensemble classifier.

```
[2]: from sklearn.datasets import load_breast_cancer
     from sklearn.preprocessing import MinMaxScaler

[3]: df = load_breast_cancer()
     X = df.data
     y = df.target

[4]: # Normailise data into the range of [0, 1] as hyperbox-based models only work in the_
     ↪ unit cube
     scaler = MinMaxScaler()
     X = scaler.fit_transform(X)

[5]: # Split data into training, validation and testing sets
     Xtr_val, X_test, ytr_val, y_test = train_test_split(X, y, train_size=0.8, random_state=0)
     Xtr, X_val, ytr, y_val = train_test_split(X, y, train_size=0.75, random_state=0)
```

This example will use the GFMM classifier with the original online learning algorithm as base learners. However, any type of hyperbox-based learning algorithms in this library can also be used to train base learners.

## 1. Using random subsampling to generate training sets for various base learners

a. The number of features used in each base learner is different and is bounded by a maximum number of features

### Training

```
[6]: # Initialise parameters
     n_estimators = 20 # number of base learners
     max_samples = 0.5 # sampling rate for samples
     max_features = 0.5 # sampling rate to generate the maximum number of features
     class_balanced = False # do not use the class-balanced sampling mode
     feature_balanced = False # use different numbers of features for base learners
     n_jobs = 4 # number of processes is used to build base learners

[7]: # Init a hyperbox-based model used to train base learners
     # Using the GFMM classifier with the original online learning algorithm with the maximum_
     ↪ hyperbox size 0.1
     base_estimator = OnlineGFMM(theta=0.1)

[8]: rh_subsampling_diff_num_features_clf = RandomHyperboxesClassifier(base_estimator=base_
     ↪ estimator, n_estimators=n_estimators, max_samples=max_samples, max_features=max_
     ↪ features, class_balanced=class_balanced, feature_balanced=feature_balanced, n_jobs=n_
     ↪ jobs, random_state=0)
     rh_subsampling_diff_num_features_clf.fit(Xtr, ytr)
```

```
[8]: RandomHyperboxesClassifier(base_estimator=OnlineGFMM(C=array([], dtype=float64),
                                                         V=array([], dtype=float64),
                                                         W=array([], dtype=float64),
                                                         theta=0.1),
                               max_features=0.5, n_estimators=20, n_jobs=4,
                               random_state=0)
```

```
[9]: print("Training time: %.3f (s)"%(rh_subsampling_diff_num_features_clf.elapsed_training_
    ↪time))
```

```
Training time: 4.155 (s)
```

```
[10]: print('Total number of hyperboxes from all base learners = %d'%rh_subsampling_diff_num_
    ↪features_clf.get_n_hyperboxes())
```

```
Total number of hyperboxes from all base learners = 2212
```

## Prediction

```
[11]: y_pred = rh_subsampling_diff_num_features_clf.predict(X_test)
      acc = accuracy_score(y_test, y_pred)
      print(f'Testing accuracy = {acc * 100: .2f}%')
```

```
Testing accuracy = 92.11%
```

## Apply pruning for base learners

```
[12]: acc_threshold=0.5 # minimum accuracy score of the unpruned hyperboxes
      keep_empty_boxes=False # False means hyperboxes that do not join the prediction process,
    ↪within the pruning procedure are also eliminated
      rh_subsampling_diff_num_features_clf.simple_pruning_base_estimators(X_val, y_val, acc_
    ↪threshold, keep_empty_boxes)
```

```
[12]: RandomHyperboxesClassifier(base_estimator=OnlineGFMM(C=array([], dtype=float64),
                                                         V=array([], dtype=float64),
                                                         W=array([], dtype=float64),
                                                         theta=0.1),
                               max_features=0.5, n_estimators=20, n_jobs=4,
                               random_state=0)
```

```
[13]: print('Total number of hyperboxes from all base learners after pruning = %d'%rh_
    ↪subsampling_diff_num_features_clf.get_n_hyperboxes())
```

```
Total number of hyperboxes from all base learners after pruning = 1219
```

### Prediction after doing a pruning procedure

```
[14]: y_pred_2 = rh_subsampling_diff_num_features_clf.predict(X_test)
      acc_pruned = accuracy_score(y_test, y_pred_2)
      print(f'Testing accuracy (after pruning) = {acc_pruned * 100: .2f}%')
```

```
Testing accuracy (after pruning) = 95.61%
```

### b. The number of features used in each base learner is the same and is equal to the given maximum number of features

```
[15]: # Initialise parameters
      n_estimators = 20 # number of base learners
      max_samples = 0.5 # sampling rate for samples
      max_features = 0.5 # sampling rate to generate the maximum number of features
      class_balanced = False # do not use the class-balanced sampling mode
      # use the same numbers of features for base learners and the number of used features is,
      ↪ the given maximum number of features
      feature_balanced = True
      n_jobs = 4 # number of processes is used to build base learners
```

```
[16]: # Init a hyperbox-based model used to train base learners
      # Using the GFMM classifier with the original online learning algorithm with the maximum,
      ↪ hyperbox size 0.1
      base_estimator = OnlineGFMM(theta=0.1)
```

```
[17]: rh_subsampling_same_num_features_clf = RandomHyperboxesClassifier(base_estimator=base_
      ↪ estimator, n_estimators=n_estimators, max_samples=max_samples, max_features=max_
      ↪ features, class_balanced=class_balanced, feature_balanced=feature_balanced, n_jobs=n_
      ↪ jobs, random_state=0)
      rh_subsampling_same_num_features_clf.fit(Xtr, ytr)
```

```
[17]: RandomHyperboxesClassifier(base_estimator=OnlineGFMM(C=array([], dtype=float64),
                                                             V=array([], dtype=float64),
                                                             W=array([], dtype=float64),
                                                             theta=0.1),
                                  feature_balanced=True, max_features=0.5,
                                  n_estimators=20, n_jobs=4, random_state=0)
```

```
[18]: print("Training time: %.3f (s)"%(rh_subsampling_same_num_features_clf.elapsed_training_
      ↪ time))
```

```
Training time: 0.841 (s)
```

```
[19]: print('Total number of hyperboxes from all base learners = %d'%rh_subsampling_same_num_
      ↪ features_clf.get_n_hyperboxes())
```

```
Total number of hyperboxes from all base learners = 3241
```

## Prediction

```
[20]: y_pred = rh_subsampling_same_num_features_clf.predict(X_test)
      acc = accuracy_score(y_test, y_pred)
      print(f'Testing accuracy = {acc * 100: .2f}%')
```

```
Testing accuracy = 94.74%
```

## Apply pruning for base learners

```
[21]: acc_threshold=0.5 # minimum accuracy score of the unpruned hyperboxes
      keep_empty_boxes=False # False means hyperboxes that do not join the prediction process,
      ↪ within the pruning procedure are also eliminated
      rh_subsampling_same_num_features_clf.simple_pruning_base_estimators(X_val, y_val, acc_
      ↪ threshold, keep_empty_boxes)
```

```
[21]: RandomHyperboxesClassifier(base_estimator=OnlineGFMM(C=array([], dtype=float64),
                                                             V=array([], dtype=float64),
                                                             W=array([], dtype=float64),
                                                             theta=0.1),
                                feature_balanced=True, max_features=0.5,
                                n_estimators=20, n_jobs=4, random_state=0)
```

## Prediction after doing a pruning procedure

```
[22]: y_pred_2 = rh_subsampling_same_num_features_clf.predict(X_test)
      acc_pruned = accuracy_score(y_test, y_pred_2)
      print(f'Testing accuracy (after pruning) = {acc_pruned * 100: .2f}%')
```

```
Testing accuracy (after pruning) = 96.49%
```

## 2. Using random undersampling to generate class-balanced training sets for various base learners

a. The number of features used in each base learner is different and is bounded by a maximum number of features

## Training

```
[23]: # Initialise parameters
      n_estimators = 20 # number of base learners
      max_samples = 0.5 # sampling rate for samples
      max_features = 0.5 # sampling rate to generate the maximum number of features
      class_balanced = True # use the class-balanced sampling mode
      feature_balanced = False # use different numbers of features for base learners
      n_jobs = 4 # number of processes is used to build base learners
```

```
[24]: # Init a hyperbox-based model used to train base learners
# Using the GFMM classifier with the original online learning algorithm with the maximum_
↳ hyperbox size 0.1
base_estimator = OnlineGFMM(theta=0.1)
```

```
[25]: rh_class_balanced_diff_num_features_clf = RandomHyperboxesClassifier(base_estimator=base_
↳ estimator, n_estimators=n_estimators, max_samples=max_samples, max_features=max_
↳ features, class_balanced=class_balanced, feature_balanced=feature_balanced, n_jobs=n_
↳ jobs, random_state=0)
rh_class_balanced_diff_num_features_clf.fit(Xtr, ytr)
```

```
[25]: RandomHyperboxesClassifier(base_estimator=OnlineGFMM(C=array([], dtype=float64),
V=array([], dtype=float64),
W=array([], dtype=float64),
theta=0.1),
class_balanced=True, max_features=0.5,
n_estimators=20, n_jobs=4, random_state=0)
```

```
[26]: print("Training time: %.3f (s)"%(rh_class_balanced_diff_num_features_clf.elapsed_
↳ training_time))
```

```
Training time: 4.061 (s)
```

```
[27]: print('Total number of hyperboxes from all base learners = %d'%rh_class_balanced_diff_
↳ num_features_clf.get_n_hyperboxes())
```

```
Total number of hyperboxes from all base learners = 2288
```

## Prediction

```
[28]: y_pred = rh_class_balanced_diff_num_features_clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print(f'Testing accuracy = {acc * 100: .2f}%')
```

```
Testing accuracy = 91.23%
```

## Apply pruning for base learners

```
[29]: acc_threshold=0.5 # minimum accuracy score of the unpruned hyperboxes
keep_empty_boxes=False # False means hyperboxes that do not join the prediction process_
↳ within the pruning procedure are also eliminated
rh_class_balanced_diff_num_features_clf.simple_pruning_base_estimators(X_val, y_val, acc_
↳ threshold, keep_empty_boxes)
```

```
[29]: RandomHyperboxesClassifier(base_estimator=OnlineGFMM(C=array([], dtype=float64),
V=array([], dtype=float64),
W=array([], dtype=float64),
theta=0.1),
class_balanced=True, max_features=0.5,
n_estimators=20, n_jobs=4, random_state=0)
```



```
[30]: print('Total number of hyperboxes from all base learners after pruning = %d'%rh_class_
      ↪ balanced_diff_num_features_clf.get_n_hyperboxes())
```

Total number of hyperboxes from all base learners after pruning = 1546

### Prediction after doing a pruning procedure

```
[31]: y_pred_2 = rh_class_balanced_diff_num_features_clf.predict(X_test)
      acc_pruned = accuracy_score(y_test, y_pred_2)
      print(f'Testing accuracy (after pruning) = {acc_pruned * 100: .2f}%')
```

Testing accuracy (after pruning) = 97.37%

### b. The number of features used in each base learner is the same and is equal to the given maximum number of features

```
[32]: # Initialise parameters
      n_estimators = 20 # number of base learners
      max_samples = 0.5 # sampling rate for samples
      max_features = 0.5 # sampling rate to generate the maximum number of features
      class_balanced = True # use the class-balanced sampling mode
      # use the same numbers of features for base learners and the number of used features is,
      ↪ the given maximum number of features
      feature_balanced = True
      n_jobs = 4 # number of processes is used to build base learners
```

```
[33]: # Init a hyperbox-based model used to train base learners
      # Using the GFMM classifier with the original online learning algorithm with the maximum,
      ↪ hyperbox size 0.1
      base_estimator = OnlineGFMM(theta=0.1)
```

```
[34]: rh_class_balanced_same_num_features_clf = RandomHyperboxesClassifier(base_estimator=base_
      ↪ estimator, n_estimators=n_estimators, max_samples=max_samples, max_features=max_
      ↪ features, class_balanced=class_balanced, feature_balanced=feature_balanced, n_jobs=n_
      ↪ jobs, random_state=0)
      rh_class_balanced_same_num_features_clf.fit(Xtr, ytr)
```

```
[34]: RandomHyperboxesClassifier(base_estimator=OnlineGFMM(C=array([], dtype=float64),
      V=array([], dtype=float64),
      W=array([], dtype=float64),
      theta=0.1),
      class_balanced=True, feature_balanced=True,
      max_features=0.5, n_estimators=20, n_jobs=4,
      random_state=0)
```

```
[35]: print("Training time: %.3f (s)"%(rh_class_balanced_same_num_features_clf.elapsed_
      ↪ training_time))
```

Training time: 0.474 (s)

```
[36]: print('Total number of hyperboxes from all base learners = %d'%rh_class_balanced_same_
      ↪ num_features_clf.get_n_hyperboxes())
```

Total number of hyperboxes from all base learners = 3356

### Prediction

```
[37]: y_pred = rh_class_balanced_same_num_features_clf.predict(X_test)
      acc = accuracy_score(y_test, y_pred)
      print(f'Testing accuracy = {acc * 100: .2f}%')
```

Testing accuracy = 91.23%

### Apply pruning for base learners

```
[38]: acc_threshold=0.5 # minimum accuracy score of the unpruned hyperboxes
      keep_empty_boxes=False # False means hyperboxes that do not join the prediction process,
      ↪ within the pruning procedure are also eliminated
      rh_class_balanced_same_num_features_clf.simple_pruning_base_estimators(X_val, y_val, acc_
      ↪ threshold, keep_empty_boxes)
```

```
[38]: RandomHyperboxesClassifier(base_estimator=OnlineGFMM(C=array([], dtype=float64),
                                                             V=array([], dtype=float64),
                                                             W=array([], dtype=float64),
                                                             theta=0.1),
                                class_balanced=True, feature_balanced=True,
                                max_features=0.5, n_estimators=20, n_jobs=4,
                                random_state=0)
```

### Prediction after doing a pruning procedure

```
[39]: y_pred_2 = rh_class_balanced_same_num_features_clf.predict(X_test)
      acc_pruned = accuracy_score(y_test, y_pred_2)
      print(f'Testing accuracy (after pruning) = {acc_pruned * 100: .2f}%')
```

Testing accuracy (after pruning) = 96.49%

### Random Hyperboxes with Hyper-parameter Optimisation for Base Learners

This example shows how to use a random hyperboxes classifier, in which each base hyperbox-based model is trained on a subset of features and a subset of samples using random search-based hyper-parameter tuning and k-fold cross-validation.

While the original random hyperboxes model in the class `RandomHyperboxesClassifier` uses the same base learners with the same hyperparameters, the cross-validation random hyperboxes model in the class `CrossValRandomHyperboxesClassifier` allows each base learner to use specific hyperparameters depending on its training data by performing random research to find the best combination of hyperparameters for each base learner.

```
[1]: import warnings
warnings.filterwarnings('ignore')
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from hbbbrain.numerical_data.ensemble_learner.cross_val_random_hyperboxes import ↪
    CrossValRandomHyperboxesClassifier
from hbbbrain.numerical_data.incremental_learner.onln_gfmm import OnlineGFMM
```

### Load dataset.

This example will use the breast cancer dataset available in sklearn to demonstrate how to use this ensemble classifier.

```
[2]: from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import MinMaxScaler
```

```
[3]: df = load_breast_cancer()
X = df.data
y = df.target
```

```
[4]: # Normalise data into the range of [0, 1] as hyperbox-based models only work in the
    ↪ unit cube
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
```

```
[5]: # Split data into training, validation and testing sets
Xtr_val, X_test, ytr_val, y_test = train_test_split(X, y, train_size=0.8, random_state=0)
Xtr, X_val, ytr, y_val = train_test_split(X, y, train_size=0.75, random_state=0)
```

This example will use the GFMM classifier with the original online learning algorithm as base learners. However, any type of hyperbox-based learning algorithms in this library can also be used to train base learners.

## 1. Using random subsampling to generate training sets for various base learners

a. The number of features used in each base learner is different and is bounded by a maximum number of features

### Training

```
[6]: # Initialise parameters
n_estimators = 20 # number of base learners
max_samples = 0.5 # sampling rate for samples
max_features = 0.5 # sampling rate to generate the maximum number of features
class_balanced = False # do not use the class-balanced sampling mode
feature_balanced = False # use different numbers of features for base learners
n_jobs = 4 # number of processes is used to build base learners
n_iter = 20 # Number of parameter settings that are randomly sampled to choose the best
    ↪ combination of hyperparameters
```

(continues on next page)

(continued from previous page)

```
k_fold = 5 # Number of folds to conduct Stratified K-Fold cross-validation for
↳hyperparameter tuning
```

```
[7]: # Init a hyperbox-based model used to train base learners
# Using the GFMM classifier with the original online learning algorithm
base_estimator = OnlineGFMM()
```

```
[8]: # Init ranges for hyperparameters of base learners to perform a random search process
↳for hyperparameter tuning
base_estimator_params = {'theta': np.arange(0.05, 1.01, 0.05), 'theta_min': [1], 'gamma':
↳[0.5, 1, 2, 4, 8, 16]}
```

```
[9]: cross_val_rh_subsampling_diff_num_features_clf = CrossValRandomHyperboxesClassifier(base_
↳estimator=base_estimator, base_estimator_params=base_estimator_params, n_estimators=n_
↳estimators, max_samples=max_samples, max_features=max_features, class_balanced=class_
↳balanced, feature_balanced=feature_balanced, n_iter=n_iter, k_fold=k_fold, n_jobs=n_
↳jobs, random_state=0)
cross_val_rh_subsampling_diff_num_features_clf.fit(Xtr, ytr)
```

```
[9]: CrossValRandomHyperboxesClassifier(base_estimator=OnlineGFMM(C=array([], dtype=float64),
V=array([], dtype=float64),
W=array([], dtype=float64)),
base_estimator_params={'gamma': [0.5, 1, 2,
4, 8, 16],
'theta': array([0.05, 0.1 , 0.
↳15, 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 , 0.55,
0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95, 1. ]),
'theta_min': [1]},
max_features=0.5, n_estimators=20, n_iter=20,
n_jobs=4, random_state=0)
```

```
[10]: print("Training time: %.3f (s)"%(cross_val_rh_subsampling_diff_num_features_clf.elapsed_
↳training_time))
```

```
Training time: 37.453 (s)
```

```
[11]: print('Total number of hyperboxes from all base learners = %d'%cross_val_rh_subsampling_
↳diff_num_features_clf.get_n_hyperboxes())
```

```
Total number of hyperboxes from all base learners = 1110
```

## Prediction

```
[12]: y_pred = cross_val_rh_subsampling_diff_num_features_clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print(f'Testing accuracy = {acc * 100: .2f}%')
```

```
Testing accuracy = 93.86%
```

## Apply pruning for base learners

```
[13]: acc_threshold=0.5 # minimum accuracy score of the unpruned hyperboxes
keep_empty_boxes=False # False means hyperboxes that do not join the prediction process_
↳ within the pruning procedure are also eliminated
cross_val_rh_subsampling_diff_num_features_clf.simple_pruning_base_estimators(X_val, y_
↳ val, acc_threshold, keep_empty_boxes)

[13]: CrossValRandomHyperboxesClassifier(base_estimator=OnlineGFMM(C=array([], dtype=float64),
                                                                    V=array([], dtype=float64),
                                                                    W=array([], dtype=float64)),
                                          base_estimator_params={'gamma': [0.5, 1, 2,
                                                                    4, 8, 16],
                                                                'theta': array([0.05, 0.1 , 0.
↳ 15, 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 , 0.55,
                                                                    0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95, 1. ]),
                                                                'theta_min': [1]},
                                          max_features=0.5, n_estimators=20, n_iter=20,
                                          n_jobs=4, random_state=0)

[14]: print('Total number of hyperboxes from all base learners after pruning = %d'%cross_val_
↳ rh_subsampling_diff_num_features_clf.get_n_hyperboxes())

Total number of hyperboxes from all base learners after pruning = 671
```

## Prediction after doing a pruning procedure

```
[15]: y_pred_2 = cross_val_rh_subsampling_diff_num_features_clf.predict(X_test)
acc_pruned = accuracy_score(y_test, y_pred_2)
print(f'Testing accuracy (after pruning) = {acc_pruned * 100: .2f}%')

Testing accuracy (after pruning) = 95.61%
```

## b. The number of features used in each base learner is the same and is equal to the given maximum number of features

```
[16]: # Initialise parameters
n_estimators = 20 # number of base learners
max_samples = 0.5 # sampling rate for samples
max_features = 0.5 # sampling rate to generate the maximum number of features
class_balanced = False # do not use the class-balanced sampling mode
# use the same numbers of features for base learners and the number of used features is_
↳ the given maximum number of features
feature_balanced = True
n_jobs = 4 # number of processes is used to build base learners
n_iter = 20 # Number of parameter settings that are randomly sampled to choose the best_
↳ combination of hyperparameters
k_fold = 5 # Number of folds to conduct Stratified K-Fold cross-validation for_
↳ hyperparameter tuning
```

```
[17]: # Init a hyperbox-based model used to train base learners
      # Using the GFMM classifier with the original online learning algorithm
      base_estimator = OnlineGFMM()

[18]: # Init ranges for hyperparameters of base learners to perform a random search process_
      ↪ for hyperparameter tuning
      base_estimator_params = {'theta': np.arange(0.05, 1.01, 0.05), 'theta_min': [1], 'gamma':
      ↪ [0.5, 1, 2, 4, 8, 16]}

[19]: cross_val_rh_subsampling_same_num_features_clf = CrossValRandomHyperboxesClassifier(base_
      ↪ estimator=base_estimator, base_estimator_params=base_estimator_params, n_estimators=n_
      ↪ estimators, max_samples=max_samples, max_features=max_features, class_balanced=class_
      ↪ balanced, feature_balanced=feature_balanced, n_iter=n_iter, k_fold=k_fold, n_jobs=n_
      ↪ jobs, random_state=0)
      cross_val_rh_subsampling_same_num_features_clf.fit(Xtr, ytr)

[19]: CrossValRandomHyperboxesClassifier(base_estimator=OnlineGFMM(C=array([], dtype=float64),
      V=array([], dtype=float64),
      W=array([], dtype=float64)),
      base_estimator_params={'gamma': [0.5, 1, 2,
      4, 8, 16],
      'theta': array([0.05, 0.1 , 0.
      ↪ 15, 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 , 0.55,
      0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95, 1. ]),
      'theta_min': [1]},
      feature_balanced=True, max_features=0.5,
      n_estimators=20, n_iter=20, n_jobs=4,
      random_state=0)

[20]: print("Training time: %.3f (s)"%(cross_val_rh_subsampling_same_num_features_clf.elapsed_
      ↪ training_time))

      Training time: 45.047 (s)

[21]: print('Total number of hyperboxes from all base learners = %d'%cross_val_rh_subsampling_
      ↪ same_num_features_clf.get_n_hyperboxes())

      Total number of hyperboxes from all base learners = 973
```

## Prediction

```
[22]: y_pred = cross_val_rh_subsampling_same_num_features_clf.predict(X_test)
      acc = accuracy_score(y_test, y_pred)
      print(f'Testing accuracy = {acc * 100: .2f}%')
```

```
Testing accuracy = 93.86%
```

## Apply pruning for base learners

```
[23]: acc_threshold=0.5 # minimum accuracy score of the unpruned hyperboxes
      keep_empty_boxes=False # False means hyperboxes that do not join the prediction process_
      ↪ within the pruning procedure are also eliminated
      cross_val_rh_subsampling_same_num_features_clf.simple_pruning_base_estimators(X_val, y_
      ↪ val, acc_threshold, keep_empty_boxes)

[23]: CrossValRandomHyperboxesClassifier(base_estimator=OnlineGFMM(C=array([], dtype=float64),
                                                                    V=array([], dtype=float64),
                                                                    W=array([], dtype=float64)),
                                          base_estimator_params={'gamma': [0.5, 1, 2,
                                                                    4, 8, 16],
                                                                'theta': array([0.05, 0.1, 0.
      ↪ 15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55,
                                                                    0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 1. ]),
                                                                'theta_min': [1]},
                                          feature_balanced=True, max_features=0.5,
                                          n_estimators=20, n_iter=20, n_jobs=4,
                                          random_state=0)
```

## Prediction after doing a pruning procedure

```
[24]: y_pred_2 = cross_val_rh_subsampling_same_num_features_clf.predict(X_test)
      acc_pruned = accuracy_score(y_test, y_pred_2)
      print(f'Testing accuracy (after pruning) = {acc_pruned * 100: .2f}%')
```

```
Testing accuracy (after pruning) = 94.74%
```

## 2. Using random undersampling to generate class-balanced training sets for various base learners

a. The number of features used in each base learner is different and is bounded by a maximum number of features

## Training

```
[25]: # Initialise parameters
      n_estimators = 20 # number of base learners
      max_samples = 0.5 # sampling rate for samples
      max_features = 0.5 # sampling rate to generate the maximum number of features
```

(continues on next page)

(continued from previous page)

```

class_balanced = True # use the class-balanced sampling mode
feature_balanced = False # use different numbers of features for base learners
n_jobs = 4 # number of processes is used to build base learners
n_iter = 20 # Number of parameter settings that are randomly sampled to choose the best_
↳ combination of hyperparameters
k_fold = 5 # Number of folds to conduct Stratified K-Fold cross-validation for_
↳ hyperparameter tuning

```

```

[26]: # Init a hyperbox-based model used to train base learners
      # Using the GFMM classifier with the original online learning algorithm
      base_estimator = OnlineGFMM()

```

```

[27]: # Init ranges for hyperparameters of base learners to perform a random search process_
      ↳ for hyperparameter tuning
      base_estimator_params = {'theta': np.arange(0.05, 1.01, 0.05), 'theta_min': [1], 'gamma':
      ↳ [0.5, 1, 2, 4, 8, 16]}

```

```

[28]: cross_val_rh_class_balanced_diff_num_features_clf =_
      ↳ CrossValRandomHyperboxesClassifier(base_estimator=base_estimator, base_estimator_
      ↳ params=base_estimator_params, n_estimators=n_estimators, max_samples=max_samples, max_
      ↳ features=max_features, class_balanced=class_balanced, feature_balanced=feature_
      ↳ balanced, n_iter=n_iter, k_fold=k_fold, n_jobs=n_jobs, random_state=0)
      cross_val_rh_class_balanced_diff_num_features_clf.fit(Xtr, ytr)

```

```

[28]: CrossValRandomHyperboxesClassifier(base_estimator=OnlineGFMM(C=array([], dtype=float64),
                                                                    V=array([], dtype=float64),
                                                                    W=array([], dtype=float64)),
                                          base_estimator_params={'gamma': [0.5, 1, 2,
                                                                    4, 8, 16],
                                                                    'theta': array([0.05, 0.1 , 0.
↳ 15, 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 , 0.55,
                                                                    0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95, 1. ]),
                                                                    'theta_min': [1]},
                                          class_balanced=True, max_features=0.5,
                                          n_estimators=20, n_iter=20, n_jobs=4,
                                          random_state=0)

```

```

[29]: print("Training time: %.3f (s)"%(cross_val_rh_class_balanced_diff_num_features_clf.
      ↳ elapsed_training_time))

Training time: 33.372 (s)

```

```

[30]: print('Total number of hyperboxes from all base learners = %d'%cross_val_rh_class_
      ↳ balanced_diff_num_features_clf.get_n_hyperboxes())

Total number of hyperboxes from all base learners = 1123

```



## Prediction

```
[31]: y_pred = cross_val_rh_class_balanced_diff_num_features_clf.predict(X_test)
      acc = accuracy_score(y_test, y_pred)
      print(f'Testing accuracy = {acc * 100: .2f}%')
```

```
Testing accuracy = 92.11%
```

## Apply pruning for base learners

```
[32]: acc_threshold=0.5 # minimum accuracy score of the unpruned hyperboxes
      keep_empty_boxes=False # False means hyperboxes that do not join the prediction process,
      ↪ within the pruning procedure are also eliminated
      cross_val_rh_class_balanced_diff_num_features_clf.simple_pruning_base_estimators(X_val,
      ↪ y_val, acc_threshold, keep_empty_boxes)
```

```
[32]: CrossValRandomHyperboxesClassifier(base_estimator=OnlineGFMM(C=array([], dtype=float64),
                                                                    V=array([], dtype=float64),
                                                                    W=array([], dtype=float64)),
      base_estimator_params={'gamma': [0.5, 1, 2,
                                         4, 8, 16],
                             'theta': array([0.05, 0.1, 0.
      ↪ 15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55,
      ↪ 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 1. ]),
                             'theta_min': [1]},
      class_balanced=True, max_features=0.5,
      n_estimators=20, n_iter=20, n_jobs=4,
      random_state=0)
```

```
[33]: print('Total number of hyperboxes from all base learners after pruning = %d'%cross_val_
      ↪ rh_class_balanced_diff_num_features_clf.get_n_hyperboxes())
```

```
Total number of hyperboxes from all base learners after pruning = 663
```

## Prediction after doing a pruning procedure

```
[34]: y_pred_2 = cross_val_rh_class_balanced_diff_num_features_clf.predict(X_test)
      acc_pruned = accuracy_score(y_test, y_pred_2)
      print(f'Testing accuracy (after pruning) = {acc_pruned * 100: .2f}%')
```

```
Testing accuracy (after pruning) = 94.74%
```

**b. The number of features used in each base learner is the same and is equal to the given maximum number of features**

```
[35]: # Initialise parameters
n_estimators = 20 # number of base learners
max_samples = 0.5 # sampling rate for samples
max_features = 0.5 # sampling rate to generate the maximum number of features
class_balanced = True # use the class-balanced sampling mode
# use the same numbers of features for base learners and the number of used features is
# the given maximum number of features
feature_balanced = True
n_jobs = 4 # number of processes is used to build base learners
n_iter = 20 # Number of parameter settings that are randomly sampled to choose the best
# combination of hyperparameters
k_fold = 5 # Number of folds to conduct Stratified K-Fold cross-validation for
# hyperparameter tuning
```

```
[36]: # Init a hyperbox-based model used to train base learners
# Using the GFMM classifier with the original online learning algorithm
base_estimator = OnlineGFMM()
```

```
[37]: # Init ranges for hyperparameters of base learners to perform a random search process
# for hyperparameter tuning
base_estimator_params = {'theta': np.arange(0.05, 1.01, 0.05), 'theta_min': [1], 'gamma':
# [0.5, 1, 2, 4, 8, 16]}
```

```
[38]: cross_val_rh_class_balanced_same_num_features_clf =
# CrossValRandomHyperboxesClassifier(base_estimator=base_estimator, base_estimator_
# params=base_estimator_params, n_estimators=n_estimators, max_samples=max_samples, max_
# features=max_features, class_balanced=class_balanced, feature_balanced=feature_
# balanced, n_iter=n_iter, k_fold=k_fold, n_jobs=n_jobs, random_state=0)
cross_val_rh_class_balanced_same_num_features_clf.fit(Xtr, ytr)
```

```
[38]: CrossValRandomHyperboxesClassifier(base_estimator=OnlineGFMM(C=array([], dtype=float64),
V=array([], dtype=float64),
W=array([], dtype=float64)),
base_estimator_params={'gamma': [0.5, 1, 2,
4, 8, 16],
'theta': array([0.05, 0.1 , 0.
15, 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 , 0.55,
0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95, 1. ]),
'theta_min': [1]},
class_balanced=True, feature_balanced=True,
max_features=0.5, n_estimators=20, n_iter=20,
n_jobs=4, random_state=0)
```

```
[39]: print("Training time: %.3f (s)"%(cross_val_rh_class_balanced_same_num_features_clf.
# elapsed_training_time))
```

```
Training time: 30.501 (s)
```

```
[40]: print('Total number of hyperboxes from all base learners = %d'%cross_val_rh_class_
# balanced_same_num_features_clf.get_n_hyperboxes())
```

```
Total number of hyperboxes from all base learners = 1623
```

## Prediction

```
[41]: y_pred = cross_val_rh_class_balanced_same_num_features_clf.predict(X_test)
      acc = accuracy_score(y_test, y_pred)
      print(f'Testing accuracy = {acc * 100: .2f}%')
```

```
Testing accuracy = 91.23%
```

## Apply pruning for base learners

```
[42]: acc_threshold=0.5 # minimum accuracy score of the unpruned hyperboxes
      keep_empty_boxes=False # False means hyperboxes that do not join the prediction process_
      ↪ within the pruning procedure are also eliminated
      cross_val_rh_class_balanced_same_num_features_clf.simple_pruning_base_estimators(X_val,
      ↪ y_val, acc_threshold, keep_empty_boxes)
```

```
[42]: CrossValRandomHyperboxesClassifier(base_estimator=OnlineGFMM(C=array([], dtype=float64),
                                                                    V=array([], dtype=float64),
                                                                    W=array([], dtype=float64)),
      base_estimator_params={'gamma': [0.5, 1, 2,
                                         4, 8, 16],
                             'theta': array([0.05, 0.1, 0.
      ↪ 15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55,
      ↪ 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 1. ]),
                             'theta_min': [1]},
      class_balanced=True, feature_balanced=True,
      max_features=0.5, n_estimators=20, n_iter=20,
      n_jobs=4, random_state=0)
```

```
[43]: print('Total number of hyperboxes from all base learners after pruning = %d'%cross_val_
      ↪ rh_class_balanced_same_num_features_clf.get_n_hyperboxes())
```

```
Total number of hyperboxes from all base learners after pruning = 1234
```

## Prediction after doing a pruning procedure

```
[44]: y_pred_2 = cross_val_rh_class_balanced_same_num_features_clf.predict(X_test)
      acc_pruned = accuracy_score(y_test, y_pred_2)
      print(f'Testing accuracy (after pruning) = {acc_pruned * 100: .2f}%')
```

```
Testing accuracy (after pruning) = 95.61%
```

## 2.8.5 Mixed data learners

### Enhanced Improved Online Learning Algorithm with Mixed-Attribute Data for GFMM

This example shows how to use the general fuzzy min-max neural network trained by the extended improved incremental learning algorithm for mixed attribute data (EIOL-GFMM)

Note that the numerical features in training and testing datasets must be in the range of [0, 1] because the GFMM classifiers require features in the unit cube. Therefore, continuous features need to be normalised before training. For categorical feature, nothing needs to be done as the EIOL-GFMM does not require any categorical feature encoding methods.

#### 1. Execute directly from the python file

```
[1]: import os
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import accuracy_score
```

#### Get the path to the this jupyter notebook file

```
[2]: this_notebook_dir = os.path.dirname(os.path.abspath("__file__"))
this_notebook_dir

[2]: 'C:\\hyperbox-brain\\examples\\mixed_data'
```

#### Get the home folder of the Hyperbox-Brain project

```
[3]: from pathlib import Path
project_dir = Path(this_notebook_dir).parent.parent
project_dir

[3]: WindowsPath('C:/hyperbox-brain')
```

#### Create the path to the Python file containing the implementation of the GFMM classifier using the extended improved online learning algorithm for mixed attribute data

```
[4]: eiol_gfmm_file_path = os.path.join(project_dir, Path("hbbrain/mixed_data/eiol_gfmm.py"))
eiol_gfmm_file_path

[4]: 'C:\\hyperbox-brain\\hbbrain\\mixed_data\\eiol_gfmm.py'
```

## Run the found file by showing the execution directions

```
[5]: !python "{eiol_gfmm_file_path}" -h
```

```
usage: eiol_gfmm.py [-h] -training_file TRAINING_FILE -testing_file
                    TESTING_FILE -categorical_features CATEGORICAL_FEATURES
                    [--theta THETA] [--delta DELTA] [--gamma GAMMA]
                    [--alpha ALPHA]
```

The description of parameters

required arguments:

```
-training_file TRAINING_FILE
                        A required argument for the path to training data file
                        (including file name)
-testing_file TESTING_FILE
                        A required argument for the path to testing data file
                        (including file name)
-categorical_features CATEGORICAL_FEATURES
                        Indices of categorical features
```

optional arguments:

```
--theta THETA          Maximum hyperbox size (in the range of (0, 1])
                        (default: 0.5)
--delta DELTA          Maximum changing entropy for categorical features (in
                        the range of (0, 1]) (default: 0.5)
--gamma GAMMA          A sensitivity parameter describing the speed of
                        decreasing of the membership function in each
                        continous dimension (larger than 0) (default: 1)
--alpha ALPHA          The trade-off weighting factor between categorical
                        features and numerical features for membership values
                        (in the range of [0, 1]) (default: 0.5)
```

## Create the path to mixed-attribute training and testing datasets stored in the dataset folder.

This example uses the `japanese_credit` dataset for illustration purposes. The continuous features in this dataset were normalised into the range of `[0, 1]`, while categorical features were kept unchanged.

```
[6]: training_data_file = os.path.join(project_dir, Path("dataset/japanese_credit_train.csv"))
training_data_file
```

```
[6]: 'C:\\hyperbox-brain\\dataset\\japanese_credit_train.csv'
```

```
[7]: testing_data_file = os.path.join(project_dir, Path("dataset/japanese_credit_test.csv"))
testing_data_file
```

```
[7]: 'C:\\hyperbox-brain\\dataset\\japanese_credit_test.csv'
```

## Run a demo program

```
[8]: !python "{eiol_gfmm_file_path}" -training_file "{training_data_file}" -testing_file "
    ↪{testing_data_file}" -categorical_features "[0, 3, 4, 5, 6, 8, 9, 11,12]" --theta 0.1 -
    ↪-delta 0.6 --gamma 1 --alpha 0.5
```

```
Number of hyperboxes = 378
Testing accuracy = 82.44%
```

## 2. Using the EIOL-GFMM algorithm to train a GFMM classifier for mixed-attribute data through its init, fit, and predict functions

```
[9]: from hbbbrain.mixed_data.eiol_gfmm import ExtendedImprovedOnlineGFMM
import pandas as pd
```

### Create mixed attribute training, validation, and testing data sets.

This example will use the `japanese_credit` dataset for illustration purposes. The continuous features in this dataset were normalised into the range of `[0, 1]`, while categorical features were kept unchanged.

```
[10]: df_train = pd.read_csv(training_data_file, header=None)
df_test = pd.read_csv(testing_data_file, header=None)
```

```
Xy_train = df_train.to_numpy()
Xy_test = df_test.to_numpy()
```

```
Xtr = Xy_train[:, :-1]
ytr = Xy_train[:, -1].astype(int)
```

```
Xtest = Xy_test[:, :-1]
ytest = Xy_test[:, -1].astype(int)
```

```
[11]: val_data_file = os.path.join(project_dir, Path("dataset/japanese_credit_val.csv"))
df_val = pd.read_csv(val_data_file, header=None)
Xy_val = df_val.to_numpy()
Xval = Xy_val[:, :-1]
yval = Xy_val[:, -1].astype(int)
```

### Initializing parameters

```
[12]: theta = 0.1 # maximum hyperbox size for continuous features
delta = 0.6 # The maximum value of the increased entropy degree for each categorical_
    ↪dimension after extended.
gamma = 1 # speed of decreasing degree in the membership values of continuous features
alpha = 0.5 # the trade-off factor for the contribution of categorical features and_
    ↪continuous features to final membership value
```

### Indicate the indices of categorical features in the training data

```
[13]: categorical_features = [0, 3, 4, 5, 6, 8, 9, 11, 12]
```

### a. Training the EIOL-GFMM algorithm with the categorical feature expansion condition regarding the maximum entropy changing threshold be applied for every categorical dimension

#### Training

```
[14]: eiol_gfmm_clf = ExtendedImprovedOnlineGFMM(theta=theta, gamma=gamma, delta=delta,
↪ alpha=alpha)
eiol_gfmm_clf.fit(Xtr, ytr, categorical_features, type_cat_expansion=0)

[14]: ExtendedImprovedOnlineGFMM(C=array([0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0,
↪ 1, 0, 1, 1,
      0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1,
      1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0,
      1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
      0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0,
      0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0,
      0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0,
      0, 1, 1, ...
      [8.60317460e-02, 3.39285714e-01, 5.26315789e-02, 0.00000000e+00,
        6.00000000e-02, 2.20600000e-02],
      ...,
      [1.41587302e-01, 2.82142857e-02, 2.98245614e-03, 0.00000000e+00,
        7.20000000e-02, 0.00000000e+00],
      [6.93174603e-01, 3.03571429e-01, 2.45614035e-01, 4.47761194e-02,
        0.00000000e+00, 0.00000000e+00],
      [5.38412698e-01, 1.03571429e-02, 5.26315789e-01, 2.98507463e-01,
        0.00000000e+00, 1.50000000e-01]]),
      delta=0.6, theta=0.1)
```

```
[15]: print("Number of existing hyperboxes = %d"%(eiol_gfmm_clf.get_n_hyperboxes()))
Number of existing hyperboxes = 378
```

```
[16]: print("Training time: %.3f (s)"%eiol_gfmm_clf.elapsed_training_time)
Training time: 0.991 (s)
```

#### Prediction

```
[17]: from hbbrain.constants import MANHATTAN_DIS, PROBABILITY_MEASURE
```

**Predict the class label for input samples using a probability measure based on the number of samples included inside the winner hyperboxes for the samples located on the decision boundaries**

```
[18]: y_pred = eiol_gfmm_clf.predict(Xtest, PROBABILITY_MEASURE)
acc = accuracy_score(ytest, y_pred)
print(f'Accuracy = {acc * 100: .2f}%')
```

```
Accuracy = 82.44%
```

**Predict the class label for input samples using Manhattan distance measure (applied only for continuous features) for the samples located on the decision boundaries**

```
[19]: y_pred = eiol_gfmm_clf.predict(Xtest, MANHATTAN_DIS)
acc = accuracy_score(ytest, y_pred)
print(f'Accuracy (Manhattan distance for samples on the decision boundaries) = {acc * 100: .2f}%')
```

```
Accuracy (Manhattan distance for samples on the decision boundaries) = 78.63%
```

**Explaining the predicted result for the input sample by showing membership values and hyperboxes for each class**

```
[20]: sample_need_explain = 1
y_pred_input_0, mem_val_classes, min_points_classes, max_points_classes, dict_cat_bound_
classes = eiol_gfmm_clf.get_sample_explanation(Xtest[sample_need_explain])
print("Explain samples:")
print("Membership values for classes: ", mem_val_classes)
print("Predicted class = ", y_pred_input_0)
print("Minimum continuous points of the selected hyperbox for each class: ", min_points_
classes)
print("Maximum continuous points of the selected hyperbox for each class: ", max_points_
classes)
print("Categorical bounds of the selected hyperbox for each class: ", dict_cat_bound_
classes)
```

```
Explain samples:
```

```
Membership values for classes: {0: 0.8441127694859039, 1: 0.9191765873015874}
```

```
Predicted class = 1
```

```
Minimum continuous points of the selected hyperbox for each class: {0: array([0.
```

```
13888889, 0.30357143, 0.06140351, 0.14925373, 0.04
0.0099 ]), 1: array([0.08603175, 0.30660714, 0.02631579, 0.10447761, 0.048
0.
])}
```

```
Maximum continuous points of the selected hyperbox for each class: {0: array([0.
```

```
13888889, 0.30357143, 0.06140351, 0.14925373, 0.04
0.0099 ]), 1: array([0.08603175, 0.30660714, 0.02631579, 0.10447761, 0.048
0.
])}
```

```
Categorical bounds of the selected hyperbox for each class: {0: array(['a': 1], {'u': 1}, {'g': 1}, {'q': 1}, {'v': 1}, {'t': 1},
```

(continues on next page)



(continued from previous page)

```

        {'t': 1}, {'f': 1}, {'g': 1}], dtype=object), 1: array([{'a': 1}, {'u': 1}, {'g': 1},
↪ 1}, {'cc': 1}, {'h': 1}, {'t': 1},
        {'t': 1}, {'f': 1}, {'g': 1}], dtype=object))

```

### Apply pruning for the trained classifier

```

[21]: acc_threshold = 0.5 # minimum accuracy of hyperboxes being retained
keep_empty_boxes = False # do not keep the hyperboxes which do not join the prediction_
↪ process on the validation set
# using a probability measure based on the number of samples included in the hyperbox_
↪ for handling samples located on the boundary
type_boundary_handling = PROBABILITY_MEASURE
eiol_gfmm_clf.simple_pruning(Xval, yval, acc_threshold, keep_empty_boxes, type_boundary_
↪ handling)

[21]: ExtendedImprovedOnlineGFMM(C=array([0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0,
↪ 1, 0, 1, 1,
        0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0,
        1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0,
        1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0,
        0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
        0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1,
        1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0,
        1, 1, 1, ...
        [8.60317460e-02, 3.39285714e-01, 5.26315789e-02, 0.00000000e+00,
        6.00000000e-02, 2.20600000e-02],
        ...,
        [1.41587302e-01, 2.82142857e-02, 2.98245614e-03, 0.00000000e+00,
        7.20000000e-02, 0.00000000e+00],
        [6.93174603e-01, 3.03571429e-01, 2.45614035e-01, 4.47761194e-02,
        0.00000000e+00, 0.00000000e+00],
        [5.38412698e-01, 1.03571429e-02, 5.26315789e-01, 2.98507463e-01,
        0.00000000e+00, 1.50000000e-01])),
        delta=0.6, theta=0.1)

```

```

[22]: print('Number of hyperboxes after pruning = %d'%eiol_gfmm_clf.get_n_hyperboxes())

```

```

Number of hyperboxes after pruning = 358

```

### Make prediction after pruning

**Predict the class label for input samples using a probability measure based on the number of samples included inside the winner hyperboxes for the samples located on the decision boundaries**

```

[23]: y_pred_2 = eiol_gfmm_clf.predict(Xtest, PROBABILITY_MEASURE)
acc = accuracy_score(ytest, y_pred_2)
print(f'Accuracy after pruning = {acc * 100: .2f}%')

```

```

Accuracy after pruning = 83.21%

```

**Predict the class label for input samples using Manhattan distance measure (applied only for continuous features) for the samples located on the decision boundaries**

```
[24]: y_pred_2 = eiol_gfmm_clf.predict(Xtest, MANHATTAN_DIS)
acc = accuracy_score(ytest, y_pred_2)
print(f'Accuracy (Manhattan distance for samples on the decision boundaries) = {acc * 100: .2f}%')
```

```
Accuracy (Manhattan distance for samples on the decision boundaries) = 79.39%
```

**b. Training the EIOL-GFMM algorithm with the categorical feature expansion condition regarding the maximum entropy changing threshold be applied for the average changing entropy value over all categorical features.**

### Training

```
[25]: eiol_gfmm_clf = ExtendedImprovedOnlineGFMM(theta=theta, gamma=gamma, delta=delta,
        alpha=alpha)
eiol_gfmm_clf.fit(Xtr, ytr, categorical_features, type_cat_expansion=1)
```

```
[25]: ExtendedImprovedOnlineGFMM(C=array([0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0,
        0, 0, 1, 1,
        1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1,
        1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0,
        1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1,
        1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1,
        1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0,
        1, 0, 1, ...
        [2.67142857e-01, 3.80892857e-01, 2.98245614e-03, 1.79104478e-01,
        6.45000000e-02, 3.00000000e-05],
        [7.48730159e-01, 1.78571429e-01, 1.40350877e-01, 5.97014925e-02,
        0.00000000e+00, 9.90000000e-04],
        [5.33015873e-01, 2.32142857e-01, 3.50877193e-02, 0.00000000e+00,
        0.00000000e+00, 2.28000000e-03],
        [5.38412698e-01, 1.03571429e-02, 5.26315789e-01, 2.98507463e-01,
        0.00000000e+00, 1.50000000e-01]]),
        delta=0.6, theta=0.1)
```

```
[26]: print("Number of existing hyperboxes = %d"%(eiol_gfmm_clf.get_n_hyperboxes()))

Number of existing hyperboxes = 159
```

```
[27]: print("Training time: %.3f (s)"%eiol_gfmm_clf.elapsed_training_time)
```

Training time: 0.256 (s)

## Prediction

**Predict the class label for input samples using a probability measure based on the number of samples included inside the winner hyperboxes for the samples located on the decision boundaries**

```
[28]: y_pred = eiol_gfmm_clf.predict(Xtest, PROBABILITY_MEASURE)
      acc = accuracy_score(ytest, y_pred)
      print(f'Accuracy = {acc * 100: .2f}%')
```

Accuracy = 83.97%

**Predict the class label for input samples using Manhattan distance measure (applied only for continuous features) for the samples located on the decision boundaries**

```
[29]: y_pred = eiol_gfmm_clf.predict(Xtest, MANHATTAN_DIS)
      acc = accuracy_score(ytest, y_pred)
      print(f'Accuracy (Manhattan distance for samples on the decision boundaries) = {acc * 100: .2f}%')
```

Accuracy (Manhattan distance for samples on the decision boundaries) = 80.92%

**Explaining the predicted result for the input sample by showing membership values and hyperboxes for each class**

```
[30]: sample_need_explain = 1
      y_pred_input_0, mem_val_classes, min_points_classes, max_points_classes, dict_cat_bound_
      ↪ classes = eiol_gfmm_clf.get_sample_explanation(Xtest[sample_need_explain])
      print("Explain samples:")
      print("Membership values for classes: ", mem_val_classes)
      print("Predicted class = ", y_pred_input_0)
      print("Minimum continuous points of the selected hyperbox for each class: ", min_points_
      ↪ classes)
      print("Maximum continuous points of the selected hyperbox for each class: ", max_points_
      ↪ classes)
      print("Categorical bounds of the selected hyperbox for each class: ", dict_cat_bound_
      ↪ classes)
```

Explain samples:

Membership values for classes: {0: 0.818407960199005, 1: 0.8854166666666667}

Predicted class = 1

Minimum continuous points of the selected hyperbox for each class: {0: array([6.07936508e-02, 3.57142857e-01, 4.38596491e-03, 1.49253731e-02, 0.00000000e+00, 1.00000000e-05]), 1: array([1.46825397e-01, 4.19642857e-01, 1.75438596e-02, 1.49253731e-02, 6.00000000e-02, 1.10000000e-04])}

Maximum continuous points of the selected hyperbox for each class: {0: array([0.

(continues on next page)

(continued from previous page)

```

→ 15079365, 0.45089286, 0.03508772, 0.02985075, 0.06
    0.05552    ]), 1: array([0.21698413, 0.51785714, 0.10824561, 0.02985075, 0.15
→ ,
    0.00551    ])}
Categorical bounds of the selected hyperbox for each class: {0: array([{'b': 2, 'a': 1},
→ {'u': 3}, {'g': 3}, {'w': 2, 'c': 1},
    {'h': 1, 'v': 2}, {'f': 3}, {'t': 3}, {'f': 3}, {'g': 3}],
    dtype=object), 1: array([{'a': 2}, {'u': 2}, {'g': 2}, {'x': 2}, {'h': 2}, {'t': 2}
→ ,
    {'t': 2}, {'t': 1, 'f': 1}, {'g': 2}], dtype=object)}

```

### Apply pruning for the trained classifier

```

[31]: acc_threshold = 0.5 # minimum accuracy of hyperboxes being retained
keep_empty_boxes = False # do not keep the hyperboxes which do not join the prediction
→ process on the validation set
# using a probability measure based on the number of samples included in the hyperbox
→ for handling samples located on the boundary
type_boundary_handling = PROBABILITY_MEASURE
eiol_gfmm_clf.simple_pruning(Xval, yval, acc_threshold, keep_empty_boxes, type_boundary_
→ handling)

[31]: ExtendedImprovedOnlineGFMM(C=array([0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0,
→ 1, 0, 1, 1,
    0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1,
    0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1]),
    D=array([[{'a': 3, 'b': 12}, {'u': 10, 'y': 5}, {'g': 10, 'p':
→ 5},
    {'q': 1, 'w': 4, 'k': 5, 'c': 2, 'i': 1, 'x': 1, 'm': 1},
    {'v': 11, 'h': 3, 'ff': 1}, {'f': 15}, {'t': 5, 'f': 10},
    {'t': 6, 'f': 9}, {'g': 14, 's': 1}],
    [{'b': 2, 'a': 3}, {'u': 4...
    [4.35238095e-01, 2.32142857e-01, 1.75438596e-02, 4.47761194e-02,
    1.14000000e-01, 0.00000000e+00],
    [5.46349206e-01, 1.25000000e-01, 1.22807018e-01, 0.00000000e+00,
    1.15000000e-01, 0.00000000e+00],
    [6.09841270e-01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
    0.00000000e+00, 0.00000000e+00],
    [7.48730159e-01, 1.78571429e-01, 1.40350877e-01, 5.97014925e-02,
    0.00000000e+00, 9.90000000e-04]]),
    delta=0.6, theta=0.1)

```

### Make prediction after pruning

**Predict the class label for input samples using a probability measure based on the number of samples included inside the winner hyperboxes for the samples located on the decision boundaries**

```
[32]: y_pred = eiol_gfmm_clf.predict(Xtest, PROBABILITY_MEASURE)
      acc = accuracy_score(ytest, y_pred)
      print(f'Accuracy = {acc * 100: .2f}%')
```

Accuracy = 82.44%

**Predict the class label for input samples using Manhattan distance measure (applied only for continuous features) for the samples located on the decision boundaries**

```
[33]: y_pred = eiol_gfmm_clf.predict(Xtest, MANHATTAN_DIS)
      acc = accuracy_score(ytest, y_pred)
      print(f'Accuracy (Manhattan distance for samples on the decision boundaries) = {acc * 100: .2f}%')
```

Accuracy (Manhattan distance for samples on the decision boundaries) = 82.44%

### Batch-Incremental Learning Algorithm for GFMM using Probability-based Measures for Categorical Features

This example shows how to use the general fuzzy min-max neural network trained by the batch-incremental learning algorithm, in which categorical features are encoded using the ordinal encoding method and the similarity values among categorical feature are computed using frequency of categorical values.

Note that the numerical features in training and testing datasets must be in the range of [0, 1] because the GFMM classifiers require features in the unit cube. Therefore, continuous features need to be normalised before training. For categorical features, nothing needs to be done as this FreqCatOnlineGFMM classifier will apply the appropriate encoding method for the categorical values.

#### 1. Execute directly from the python file

```
[1]: import os
      import warnings
      warnings.filterwarnings('ignore')
      from sklearn.metrics import accuracy_score
```

### Get the path to the this jupyter notebook file

```
[2]: this_notebook_dir = os.path.dirname(os.path.abspath("__file__"))
    this_notebook_dir

[2]: 'C:\\hyperbox-brain\\examples\\mixed_data'
```

### Get the home folder of the Hyperbox-Brain project

```
[3]: from pathlib import Path
    project_dir = Path(this_notebook_dir).parent.parent
    project_dir

[3]: WindowsPath('C:/hyperbox-brain')
```

Create the path to the Python file containing the implementation of the GFMM classifier using the online learning algorithm with the cateogical feature similarity measure based on the frequency of occurence of categorical values for mixed attribute features

```
[4]: freq_cat_gfmm_file_path = os.path.join(project_dir, Path("hbbrain/mixed_data/freq_cat_
    ↪onln_gfmm.py"))
    freq_cat_gfmm_file_path

[4]: 'C:\\hyperbox-brain\\hbbrain\\mixed_data\\freq_cat_onln_gfmm.py'
```

### Run the found file by showing the execution directions

```
[5]: !python "{freq_cat_gfmm_file_path}" -h

usage: freq_cat_onln_gfmm.py [-h] -training_file TRAINING_FILE -testing_file
                             TESTING_FILE -categorical_features
                             CATEGORICAL_FEATURES [--theta THETA]
                             [--theta_min THETA_MIN] [--eta ETA]
                             [--gamma GAMMA] [--alpha ALPHA]

The description of parameters

required arguments:
  -training_file TRAINING_FILE
                        A required argument for the path to training data file
                        (including file name)
  -testing_file TESTING_FILE
                        A required argument for the path to testing data file
                        (including file name)
  -categorical_features CATEGORICAL_FEATURES
                        Indices of categorical features

optional arguments:
  --theta THETA        Maximum hyperbox size (in the range of (0, 1])
```

(continues on next page)

(continued from previous page)

```

--theta_min THETA_MIN      (default: 0.5)
                             Minimum value of the maximum hyperbox size to escape
                             the training loop (in the range of (0, 1]) (default:
                             0.5)
--eta ETA                  Maximum similarity value for each pair of categorical
                             values (in the range of (0, 1] (default: 0.5)
--gamma GAMMA              A sensitivity parameter describing the speed of
                             decreasing of the membership function in each
                             continuous dimension (larger than 0) (default: 1)
--alpha ALPHA              Multiplier showing the decrease of theta in each step
                             (default: 0.9)

```

### Create the path to mixed-attribute training and testing datasets stored in the dataset folder.

This example uses the `japanese_credit` dataset for illustration purposes. The continuous features in this dataset were normalised into the range of `[0, 1]`, while categorical features were kept unchanged.

```
[6]: training_data_file = os.path.join(project_dir, Path("dataset/japanese_credit_train.csv"))
      training_data_file
```

```
[6]: 'C:\\hyperbox-brain\\dataset\\japanese_credit_train.csv'
```

```
[7]: testing_data_file = os.path.join(project_dir, Path("dataset/japanese_credit_test.csv"))
      testing_data_file
```

```
[7]: 'C:\\hyperbox-brain\\dataset\\japanese_credit_test.csv'
```

### Run a demo program

```
[8]: !python "{freq_cat_gfmm_file_path}" -training_file "{training_data_file}" -testing_file "
      ↪{testing_data_file}" -categorical_features "[0, 3, 4, 5, 6, 8, 9, 11,12]" --theta 0.1 -
      ↪-theta_min 0.1 --eta 0.6 --gamma 1
```

```

Number of hyperboxes = 266
Testing accuracy = 80.92%

```

## 2. Using the `FreqCatOnlineGFMM` algorithm to train a GFMM classifier for mixed-attribute data through its `init`, `fit`, and `predict` functions

```
[9]: from hbbrain.mixed_data.freq_cat_onln_gfmm import FreqCatOnlineGFMM
      import pandas as pd
```

### Create mixed attribute training, validation, and testing data sets.

This example will use the `japanese_credit` dataset for illustration purposes. The continuous features in this dataset were normalised into the range of `[0, 1]`, while categorical features were kept unchanged.

```
[10]: df_train = pd.read_csv(training_data_file, header=None)
df_test = pd.read_csv(testing_data_file, header=None)

Xy_train = df_train.to_numpy()
Xy_test = df_test.to_numpy()

Xtr = Xy_train[:, :-1]
ytr = Xy_train[:, -1].astype(int)

Xtest = Xy_test[:, :-1]
ytest = Xy_test[:, -1].astype(int)

[11]: val_data_file = os.path.join(project_dir, Path("dataset/japanese_credit_val.csv"))
df_val = pd.read_csv(val_data_file, header=None)
Xy_val = df_val.to_numpy()
Xval = Xy_val[:, :-1]
yval = Xy_val[:, -1].astype(int)
```

### Initializing parameters

```
[12]: theta = 0.1 # maximum hyperbox size for continuous features
theta_min = 0.1 # Only performing one training loop
eta = 0.6 # Maximum similarity value for each pair of categorical values
gamma = 1 # speed of decreasing degree in the membership values of continuous features
```

### Indicate the indices of categorical features in the training data

```
[13]: categorical_features = [0, 3, 4, 5, 6, 8, 9, 11, 12]
```

### Training

```
[14]: freq_cat_onln_gfmm_clf = FreqCatOnlineGFMM(theta=theta, theta_min=theta_min, eta=eta,
→ gamma=gamma)
freq_cat_onln_gfmm_clf.fit(Xtr, ytr, categorical_features)

[14]: FreqCatOnlineGFMM(C=array([0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1,
→ 0,
    1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0,
    1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1,
    0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0,
    0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
    0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0,
    0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1,
```

(continues on next page)



(continued from previous page)

```

0, 0, 1, 1, 1, 0, 1, 0...
[1.32222222e-01, 3.92857143e-01, 5.26315789e-02, 0.00000000e+00,
 6.00000000e-02, 2.20600000e-02],
...,
[4.35238095e-01, 2.32142857e-01, 1.75438596e-02, 4.47761194e-02,
 7.25000000e-02, 0.00000000e+00],
[1.29682540e-01, 1.78571429e-02, 4.38596491e-03, 0.00000000e+00,
 1.80000000e-01, 0.00000000e+00],
[5.38412698e-01, 1.03571429e-02, 5.26315789e-01, 2.98507463e-01,
 0.00000000e+00, 1.50000000e-01]]],
      eta=0.6, theta=0.1, theta_min=0.1)

```

```
[15]: print('Number of hyperboxes = %d'%freq_cat_onln_gfmm_clf.get_n_hyperboxes())
Number of hyperboxes = 266
```

```
[16]: print("Training time: %.3f (s)"%freq_cat_onln_gfmm_clf.elapsed_training_time)
Training time: 1.256 (s)
```

## Prediction

```
[17]: y_pred = freq_cat_onln_gfmm_clf.predict(Xtest)
acc = accuracy_score(ytest, y_pred)
print(f'Accuracy = {acc * 100: .2f}%')
Accuracy = 80.92%
```

## Explaining the predicted result for the input sample by showing membership values and hyperboxes for each class

```
[18]: sample_need_explain = 1
y_pred_input_0, mem_val_classes, min_points_classes, max_points_classes, dict_min_point_
↪cat_classes, dict_max_point_cat_classes = freq_cat_onln_gfmm_clf.get_sample_
↪explanation(Xtest[sample_need_explain])
print("Explain samples:")
print("Membership values for classes: ", mem_val_classes)
print("Predicted class = ", y_pred_input_0)
print("Minimum continuous points of the selected hyperbox for each class: ", min_points_
↪classes)
print("Maximum continuous points of the selected hyperbox for each class: ", max_points_
↪classes)
print("Minimum categorical points of the selected hyperbox for each class: ", dict_min_
↪point_cat_classes)
print("Maximum categorical points of the selected hyperbox for each class: ", dict_max_
↪point_cat_classes)

Explain samples:
Membership values for classes:  {0: 0.6642512077294687, 1: 0.75}
Predicted class = 1
```

(continues on next page)

(continued from previous page)

```

Minimum continuous points of the selected hyperbox for each class: {0: array([0.1852381,
→ 0.04017857, 0.04526316, 0.02985075, 0.1
0.      ]), 1: array([0.03301587, 0.02089286, 0.00578947, 0.02985075, 0.05
→ 0.      ])}
Maximum continuous points of the selected hyperbox for each class: {0: array([0.1852381,
→ 0.04017857, 0.04526316, 0.02985075, 0.1
0.      ]), 1: array([0.10984127, 0.10714286, 0.07315789, 0.07462687, 0.11
→ 0.02503      ])}
Minimum categorical points of the selected hyperbox for each class: {0: array([0.0, 1.0,
→ 0.0, 10.0, 7.0, 1.0, 1.0, 0.0, 0.0], dtype=object), 1: array([0.0, 1.0, 0.0, 7.0, 7.0,
→ 1.0, 1.0, 0.0, 0.0], dtype=object)}
Maximum categorical points of the selected hyperbox for each class: {0: array([100000,
→ 100000, 100000, 100000, 100000, 100000, 100000,
100000]), 1: array([0, 1, 0, 1, 3, 1, 1, 0, 0])}

```

### Apply pruning for the trained classifier

```

[19]: acc_threshold = 0.5 # minimum accuracy of hyperboxes being retained
keep_empty_boxes = False # do not keep the hyperboxes which do not join the prediction
→ process on the validation set
freq_cat_onln_gfmm_clf.simple_pruning(Xval, yval, acc_threshold, keep_empty_boxes)

[19]: FreqCatOnlineGFMM(C=array([0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0,
→ 1,
0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0,
1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1,
0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0,
1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0,
0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0,
1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1,
0, 0, 1, 1, 1, 1, 0, 1...
[2.65873016e-01, 2.32142857e-01, 1.40350877e-01, 1.04477612e-01,
4.95000000e-02, 3.06500000e-02],
...,
[4.35238095e-01, 2.32142857e-01, 1.75438596e-02, 4.47761194e-02,
7.25000000e-02, 0.00000000e+00],
[1.29682540e-01, 1.78571429e-02, 4.38596491e-03, 0.00000000e+00,
1.80000000e-01, 0.00000000e+00],
[5.38412698e-01, 1.03571429e-02, 5.26315789e-01, 2.98507463e-01,
0.00000000e+00, 1.50000000e-01]]),
eta=0.6, theta=0.1, theta_min=0.1)

[20]: print('Number of hyperboxes after pruning = %d'%freq_cat_onln_gfmm_clf.get_n_
→ hyperboxes())

Number of hyperboxes after pruning = 246

```

### Make prediction after pruning

```
[21]: y_pred = freq_cat_onln_gfmm_clf.predict(Xtest)
      acc = accuracy_score(ytest, y_pred)
      print(f'Accuracy after pruning = {acc * 100: .2f}%')
```

```
Accuracy after pruning = 83.21%
```

### Batch-Incremental Learning Algorithm for GFMM using One-hot Encoding for Categorical Features

This example shows how to use the general fuzzy min-max neural network trained by the batch-incremental learning algorithm, in which categorical features are encoded using one-hot encoding.

Note that the numerical features in training and testing datasets must be in the range of [0, 1] because the GFMM classifiers require features in the unit cube. Therefore, continuous features need to be normalised before training. For categorical features, nothing needs to be done as this OneHotOnlineGFMM classifier will apply the appropriate encoding method for the categorical values.

#### 1. Execute directly from the python file

```
[1]: import os
      import warnings
      warnings.filterwarnings('ignore')
      from sklearn.metrics import accuracy_score
      import pandas as pd
```

#### Get the path to the this jupyter notebook file

```
[2]: this_notebook_dir = os.path.dirname(os.path.abspath("__file__"))
      this_notebook_dir
```

```
[2]: 'C:\\hyperbox-brain\\examples\\mixed_data'
```

#### Get the home folder of the Hyperbox-Brain project

```
[3]: from pathlib import Path
      project_dir = Path(this_notebook_dir).parent.parent
      project_dir
```

```
[3]: WindowsPath('C:/hyperbox-brain')
```

Create the path to the Python file containing the implementation of the GFMM classifier using the online learning algorithm with one-hot encoding for categorical values in mixed attribute features

```
[4]: onehot_gfmm_file_path = os.path.join(project_dir, Path("hbbrain/mixed_data/onehot_onln_
    ↪gfmm.py"))
    onehot_gfmm_file_path

[4]: 'C:\\hyperbox-brain\\hbbrain\\mixed_data\\onehot_onln_gfmm.py'
```

Run the found file by showing the execution directions

```
[5]: !python "{onehot_gfmm_file_path}" -h

usage: onehot_onln_gfmm.py [-h] -training_file TRAINING_FILE -testing_file
                        TESTING_FILE -categorical_features
                        CATEGORICAL_FEATURES [--theta THETA]
                        [--theta_min THETA_MIN]
                        [--min_percent_overlap_cat MIN_PERCENT_OVERLAP_CAT]
                        [--gamma GAMMA] [--alpha ALPHA]

The description of parameters

required arguments:
  -training_file TRAINING_FILE
                        A required argument for the path to training data file
                        (including file name)
  -testing_file TESTING_FILE
                        A required argument for the path to testing data file
                        (including file name)
  -categorical_features CATEGORICAL_FEATURES
                        Indices of categorical features

optional arguments:
  --theta THETA          Maximum hyperbox size (in the range of (0, 1])
                        (default: 0.5)
  --theta_min THETA_MIN  Mimimum value of the maximum hyperbox size to escape
                        the training loop (in the range of (0, 1]) (default:
                        0.5)
  --min_percent_overlap_cat MIN_PERCENT_OVERLAP_CAT
                        Mimimum rate of numbers of categorical features
                        overlapped for hyperbox expansion (default: 0.5)
  --gamma GAMMA          A sensitivity parameter describing the speed of
                        decreasing of the membership function in each
                        continous dimension (larger than 0) (default: 1)
  --alpha ALPHA          Multiplier showing the decrease of theta in each step
                        (default: 0.9)
```

**Create the path to mixed-attribute training and testing datasets stored in the dataset folder.**

This example uses the `japanese_credit` dataset for illustration purposes. The continuous features in this dataset were normalised into the range of `[0, 1]`, while categorical features were kept unchanged.

```
[6]: training_data_file = os.path.join(project_dir, Path("dataset/japanese_credit_train.csv"))
      training_data_file

[6]: 'C:\\hyperbox-brain\\dataset\\japanese_credit_train.csv'

[7]: testing_data_file = os.path.join(project_dir, Path("dataset/japanese_credit_test.csv"))
      testing_data_file

[7]: 'C:\\hyperbox-brain\\dataset\\japanese_credit_test.csv'
```

**Run a demo program**

```
[8]: !python "{onehot_gfmm_file_path}" -training_file "{training_data_file}" -testing_file "
      ↪{testing_data_file}" -categorical_features "[0, 3, 4, 5, 6, 8, 9, 11,12]" --theta 0.1 -
      ↪-theta_min 0.1 --min_percent_overlap_cat 0.6 --gamma 1

Number of hyperboxes = 166
Testing accuracy = 67.94%
```

**2. Using the OneHotOnlineGFMM algorithm to train a GFMM classifier for mixed-attribute data through its `init`, `fit`, and `predict` functions**

```
[9]: from hbbbrain.mixed_data.onehot_onln_gfmm import OneHotOnlineGFMM
      import pandas as pd
```

**Create mixed attribute training, validation, and testing data sets.**

This example will use the `japanese_credit` dataset for illustration purposes. The continuous features in this dataset were normalised into the range of `[0, 1]`, while categorical features were kept unchanged.

```
[10]: df_train = pd.read_csv(training_data_file, header=None)
      df_test = pd.read_csv(testing_data_file, header=None)

      Xy_train = df_train.to_numpy()
      Xy_test = df_test.to_numpy()

      Xtr = Xy_train[:, :-1]
      ytr = Xy_train[:, -1].astype(int)

      Xtest = Xy_test[:, :-1]
      ytest = Xy_test[:, -1].astype(int)

[11]: val_data_file = os.path.join(project_dir, Path("dataset/japanese_credit_val.csv"))
      df_val = pd.read_csv(val_data_file, header=None)
```

(continues on next page)

(continued from previous page)

```
Xy_val = df_val.to_numpy()
Xval = Xy_val[:, :-1]
yval = Xy_val[:, -1].astype(int)
```

## Initializing parameters

```
[12]: theta = 0.1 # maximum hyperbox size for continuous features
      theta_min = 0.1 # Only performing one training loop
      min_percent_overlap_cat = 0.6 # Minimum rate of numbers of categorical features,
      ↪ overlapped for hyperbox expansion
      gamma = 1 # speed of decreasing degree in the membership values of continuous features
```

## Indicate the indices of categorical features in the training data

```
[13]: categorical_features = [0, 3, 4, 5, 6, 8, 9, 11, 12]
```

## Training

```
[14]: onehot_onln_gfmm_clf = OneHotOnlineGFMM(theta=theta, theta_min=theta_min, min_percent_
      ↪ overlap_cat=min_percent_overlap_cat, gamma=gamma)
      onehot_onln_gfmm_clf.fit(Xtr, ytr, categorical_features)
```

```
[14]: OneHotOnlineGFMM(C=array([0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0,
      ↪ 1,
      1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0,
      1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1,
      0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0,
      0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0,
      1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1,
      0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 0, 0, 1,...
      1.00000000e-01, 4.00000000e-04],
      [2.67142857e-01, 3.80892857e-01, 2.98245614e-03, 1.79104478e-01,
      6.45000000e-02, 3.00000000e-05],
      [7.48730159e-01, 1.78571429e-01, 1.40350877e-01, 5.97014925e-02,
      0.00000000e+00, 9.90000000e-04],
      [5.38412698e-01, 1.03571429e-02, 5.26315789e-01, 2.98507463e-01,
      0.00000000e+00, 1.50000000e-01]]),
      min_percent_overlap_cat=0.6, theta=0.1, theta_min=0.1)
```

```
[15]: print('Number of hyperboxes = %d'%onehot_onln_gfmm_clf.get_n_hyperboxes())
      Number of hyperboxes = 166
```

```
[16]: print("Training time: %.3f (s)"%onehot_onln_gfmm_clf.elapsed_training_time)
      Training time: 1.326 (s)
```

## Prediction

```
[17]: y_pred = onehot_onln_gfmm_clf.predict(Xtest)
      acc = accuracy_score(ytest, y_pred)
      print(f'Accuracy = {acc * 100: .2f}%')
```

```
Accuracy = 67.94%
```

**Explaining the predicted result for the input sample by showing membership values and hyperboxes for each class**

```
[18]: sample_need_explain = 10
      y_pred_input_0, mem_val_classes, min_points_classes, max_points_classes, cat_poins_
      ↪classes = onehot_onln_gfmm_clf.get_sample_explanation(Xtest[sample_need_explain])
      print("Explain samples:")
      print("Membership values for classes: ", mem_val_classes)
      print("Predicted class = ", y_pred_input_0)
      print("Minimum points of the selected hyperbox for each class: ", min_points_classes)
      print("Maximum points of the selected hyperbox for each class: ", max_points_classes)
      print("Categorical features of the selected hyperbox for each class: ", cat_poins_
      ↪classes)
```

```
Explain samples:
```

```
Membership values for classes: {0: 0.9315476190476191, 1: 0.8133333333333332}
```

```
Predicted class = 0
```

```
Minimum points of the selected hyperbox for each class: {0: array([0.05031746, 0.
↪00892857, 0.          , 0.00094284, 0.05          ,
      0.01286   ]), 1: array([0.21031746, 0.05053571, 0.00140351, 0.
↪          , 0.12
↪          ,
      0.00050875])}
```

```
Maximum points of the selected hyperbox for each class: {0: array([0.14880952, 0.
↪10714286, 0.0877193 , 0.07462687, 0.14          ,
      0.04208   ]), 1: array([0.29095238, 0.14285714, 0.0877193 , 0.01492537, 0.1965
↪          ,
      0.0033975 ])}  
↪
```

```
Categorical features of the selected hyperbox for each class: {0: array([array([ True,
↪True]), array([False, True, True]),
      array([ True, False, True]),
      array([ True, True, True, False, True, True, True, True,
      False, True, True, True, False]),
      array([False, True, True, True, True, True, False, True, False]),
      array([ True, True]), array([ True, True]),
      array([ True, True]), array([ True, False, True])], dtype=object), 1:
↪array([array([ True, True]), array([False, True, True]),
      array([ True, False, True]),
      array([False, True, False, False, True, False, False, False,
      True, False, False, True, False]),
      array([False, False, False, True, False, False, False, True, False]),
      array([False, True]), array([ True, True]),
      array([ True, True]), array([ True, False, True])], dtype=object)}
```

## Apply pruning for the trained classifier

```
[19]: acc_threshold = 0.5 # minimum accuracy of hyperboxes being retained
keep_empty_boxes = False # do not keep the hyperboxes which do not join the prediction_
↳process on the validation set
onehot_onln_gfmm_clf.simple_pruning(Xval, yval, acc_threshold, keep_empty_boxes)

[19]: OneHotOnlineGFMM(C=array([0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1,
↳1,
    0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1,
    1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0,
    0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0,
    0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 0, 0, 1, 1, 0, 1, 1]...
    1.00000000e-01, 4.00000000e-04],
    [2.67142857e-01, 3.80892857e-01, 2.98245614e-03, 1.79104478e-01,
    6.45000000e-02, 3.00000000e-05],
    [7.48730159e-01, 1.78571429e-01, 1.40350877e-01, 5.97014925e-02,
    0.00000000e+00, 9.90000000e-04],
    [5.38412698e-01, 1.03571429e-02, 5.26315789e-01, 2.98507463e-01,
    0.00000000e+00, 1.50000000e-01]]),
    min_percent_overlap_cat=0.6, theta=0.1, theta_min=0.1)
```

```
[20]: print('Number of hyperboxes after pruning = %d'%onehot_onln_gfmm_clf.get_n_hyperboxes())
Number of hyperboxes after pruning = 162
```

## Make prediction after pruning

```
[21]: y_pred = onehot_onln_gfmm_clf.predict(Xtest)
acc = accuracy_score(ytest, y_pred)
print(f'Accuracy after pruning = {acc * 100: .2f}%')
```

Accuracy after pruning = 69.47%

## 2.8.6 Integration with sklearn pipeline

### Integration of Single Hyperbox-based Models with Sklearn Pipeline and Hyperopt

This example shows how to integrate the GFMM classifier into the Pipeline class implemented by scikit-learn

Note that this example is illustrated by using the original online learning algorithm for GFMM model. However, it can be used for any GFMM models using other learning algorithms

```
[1]: import warnings
warnings.filterwarnings('ignore')
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from hbbbrain.numerical_data.incremental_learner.onln_gfmm import OnlineGFMM
```



## Load Iris dataset and prepare training and testing sets

```
[2]: from sklearn.datasets import load_iris
```

```
[3]: df = load_iris()
      X = df.data
      y = df.target
```

```
[4]: X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=0)
```

## Create a pipeline of pre-processing method (i.e., normalization of data in the range of [0, 1]) and a GFMM classifier.

**Note:** The GFMM classifier using an original online learning algorithm requires the input data in the range of [0, 1].

```
[5]: theta = 0.1
      theta_min = 0.1
      onln_gfmm_clf = OnlineGFMM(theta=theta, theta_min=theta_min)
```

```
[6]: pipe = Pipeline([('scaler', MinMaxScaler()), ('onln_gfmm', onln_gfmm_clf)])
```

## Training

```
[7]: pipe.fit(X_train, y_train)
```

```
[7]: Pipeline(steps=[('scaler', MinMaxScaler()),
                     ('onln_gfmm',
                      OnlineGFMM(C=array([2, 1, 0, 2, 2, 1, 0, 1, 1, 1, 2, 0, 0, 1, 2, 2, 2, 2,
↪2, 1, 2, 2, 2,
                      1, 2, 1, 1, 2, 0, 1, 0, 0, 1, 0, 1, 2, 2, 0, 2, 0, 2, 1, 1, 0, 2,
                      2, 0, 0, 2, 2, 1, 1, 0, 1, 0, 2, 1])),
                      V=array([[0.58333333, 0.41666667, 0.68965517, 0.70833333],
                      [0.30555556, 0.41666667, 0.5862069 , 0.58333333],
                      [0.19444444, 0.625      , 0.05172414, 0.04166667],
                      [0.44444444, 0.41666667, 0.67241379, 0.66666667],
                      [0.86111111, 0.33333333, 0.86206897, 0.75      ],
                      [0.38888889, 0.25      , 0.44827586, 0.375      ],
                      [0.66666667, 0.41666667, 0.67241379, 0.66666667],
                      [0.19444444, 0.41666667, 0.0862069 , 0.04166667],
                      [0.44444444, 0.5      , 0.63793103, 0.70833333],
                      [0.33333333, 0.625      , 0.03448276, 0.04166667],
                      [0.55555556, 0.375      , 0.77586207, 0.70833333],
                      [0.41666667, 0.29166667, 0.51724138, 0.375      ]])),
                      theta=0.1, theta_min=0.1))])
```

## Testing

```
[8]: acc = pipe.score(X_test, y_test)
print(f'Testing accuracy = {acc * 100: .2f}%')

>>> The testing sample 26 with the coordinate [ 0.08333333  0.66666667 -0.01724138  0.
↪ 0.41666667] is outside the range [0, 1]. Membership value = 0.916667. The prediction is ↪
↪ more likely incorrect.
Testing accuracy = 96.67%
```

The example below shows how to use the HyperOpt library in combination with Pipeline and Cross-validation to find the best model

```
[9]: from hyperopt import fmin, hp, tpe, Trials, space_eval, STATUS_OK
from hyperopt.pyll import scope as ho_scope
from hyperopt.pyll.stochastic import sample as ho_sample
```

## Define search space

```
[10]: hp_space_gfmm = {
    'theta': hp.uniform('theta', 0, 1),
    'gamma': hp.uniform('gamma', 0, 10)
}

# Draw random sample to see if hyperspace is correctly defined
ho_sample(hp_space_gfmm)

[10]: {'gamma': 4.542173063848446, 'theta': 0.8064001273117817}
```

## Defining model

```
[11]: def init_model(hps):
    """
    Constructs estimator

    Parameters:
    -----
    hps : sample point from search space

    Returns:
    -----
    model : sklearn.Pipeline.pipeline with hyperparameters set up as per hps
    """

    # Assembling pipeline
    model = Pipeline([
        ('scale', MinMaxScaler()),
        ('clf', OnlineGFMM(**hps))
```

(continues on next page)

(continued from previous page)

```

])

return model

```

```

[12]: from sklearn.model_selection import cross_val_score, StratifiedKFold
      from sklearn.metrics import accuracy_score

```

Now we have to define function to minimize. We'll stick with cross-validation score on train set. Our function should take a sample from search space and return negative mean Acc score. As noted above, it is very important to return negative score here, since otherwise we'll seek for hyperparameters that minimize Acc

```

[13]: def f_to_min1(hps, X, y, ncv=5):
      """
      Target function for optimization

      Parameters:
      -----
      hps : sample point from search space
      X : feature matrix
      y : target array
      ncv : number of folds for cross-validation

      Returns:
      -----
      : target function value (negative mean cross-val Acc score)
      """

      model = init_model(hps)
      cv_res = cross_val_score(model, X, y, cv=StratifiedKFold(ncv, shuffle=True, random_
↪state=0),
                              scoring='accuracy', n_jobs=1)

      return -cv_res.mean()

```

## Running optimization

```

[14]: from functools import partial
      import numpy as np

```

All right, let's run optimization for 100 rounds using TPE algorithm, meaning that we use TPE to suggest next sample values based on previous function evaluations. We'll use Trials class objects to keep track of optimization history. Note: We're binding X and y arguments of target function to X\_train and y\_train respectively, using functools.partial, since target function of fmin may accept only a search space point.

```

[15]: trials_clf = Trials()
      best_clf = fmin(partial(f_to_min1, X=X_train, y=y_train),
                      hp_space_gfmm, algo=tpe.suggest, max_evals=100,
                      trials=trials_clf, rststate = np.random.default_rng(0))

```

```

>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↪79166667] is outside the range [0, 1]. Membership value = 0.888627. The prediction is_

```

(continues on next page)

(continued from previous page)

```

↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.945905. The prediction is↳
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.842222. The prediction is↳
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375↳
↳      ] is outside the range [0, 1]. Membership value = 0.800702. The prediction is more↳
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  ↳
↳      ] is outside the range [0, 1]. Membership value = 0.701052. The prediction is more↳
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.837797. The prediction is↳
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.  ↳
↳      ] is outside the range [0, 1]. Membership value = 0.921216. The prediction is↳
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.899729. The prediction is↳
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375↳
↳      ] is outside the range [0, 1]. Membership value = 0.709743. The prediction is more↳
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  ↳
↳      ] is outside the range [0, 1]. Membership value = 0.564614. The prediction is more↳
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.414332. The prediction is↳
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.  ↳
↳      ] is outside the range [0, 1]. Membership value = 0.799199. The prediction is↳
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.414332. The prediction is↳
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375↳
↳      ] is outside the range [0, 1]. Membership value = 0.260208. The prediction is more↳
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  ↳
↳      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more↳
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.814764. The prediction is↳
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.  ↳
↳      ] is outside the range [0, 1]. Membership value = 0.910028. The prediction is↳
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.885491. The prediction is↳
↳more likely incorrect.

```

(continues on next page)

(continued from previous page)

```

>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳      ] is outside the range [0, 1]. Membership value = 0.668525. The prediction is more
↳ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.502788. The prediction is more
↳ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳ 79166667] is outside the range [0, 1]. Membership value = 0.227886. The prediction is
↳ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.663078. The prediction is
↳ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳ 91666667] is outside the range [0, 1]. Membership value = 0.227886. The prediction is
↳ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳      ] is outside the range [0, 1]. Membership value = 0.024698. The prediction is more
↳ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
↳ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳ 79166667] is outside the range [0, 1]. Membership value = 0.633662. The prediction is
↳ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.822064. The prediction is
↳ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳ 91666667] is outside the range [0, 1]. Membership value = 0.773537. The prediction is
↳ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳      ] is outside the range [0, 1]. Membership value = 0.344448. The prediction is more
↳ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.016672. The prediction is more
↳ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳ 79166667] is outside the range [0, 1]. Membership value = 0.417466. The prediction is
↳ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.717055. The prediction is
↳ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳ 91666667] is outside the range [0, 1]. Membership value = 0.174743. The prediction is
↳ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
↳ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
↳ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.

```

(continues on next page)

(continued from previous page)

```

↪79166667] is outside the range [0, 1]. Membership value = 0.971811. The prediction is_
↪more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↪      ] is outside the range [0, 1]. Membership value = 0.987699. The prediction is_
↪more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↪91666667] is outside the range [0, 1]. Membership value = 0.957717. The prediction is_
↪more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375_
↪      ] is outside the range [0, 1]. Membership value = 0.946590. The prediction is more_
↪likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  _
↪      ] is outside the range [0, 1]. Membership value = 0.946590. The prediction is more_
↪likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↪79166667] is outside the range [0, 1]. Membership value = 0.917323. The prediction is_
↪more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↪      ] is outside the range [0, 1]. Membership value = 0.971654. The prediction is_
↪more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↪91666667] is outside the range [0, 1]. Membership value = 0.917323. The prediction is_
↪more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375_
↪      ] is outside the range [0, 1]. Membership value = 0.895566. The prediction is more_
↪likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  _
↪      ] is outside the range [0, 1]. Membership value = 0.843349. The prediction is more_
↪likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↪79166667] is outside the range [0, 1]. Membership value = 0.586375. The prediction is_
↪more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↪      ] is outside the range [0, 1]. Membership value = 0.793187. The prediction is_
↪more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↪91666667] is outside the range [0, 1]. Membership value = 0.379562. The prediction is_
↪more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375_
↪      ] is outside the range [0, 1]. Membership value = 0.310625. The prediction is more_
↪likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  _
↪      ] is outside the range [0, 1]. Membership value = 0.216289. The prediction is more_
↪likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↪79166667] is outside the range [0, 1]. Membership value = 0.595076. The prediction is_
↪more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↪      ] is outside the range [0, 1]. Membership value = 0.823306. The prediction is_
↪more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↪91666667] is outside the range [0, 1]. Membership value = 0.595076. The prediction is_

```

(continues on next page)

(continued from previous page)

```

↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳      ] is outside the range [0, 1]. Membership value = 0.232775. The prediction is more
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.232775. The prediction is more
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.536979. The prediction is
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.797955. The prediction is
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.536979. The prediction is
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳      ] is outside the range [0, 1]. Membership value = 0.415131. The prediction is more
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.122697. The prediction is more
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.225142. The prediction is
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.734335. The prediction is
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.225142. The prediction is
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.527092. The prediction is
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.770302. The prediction is
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.707657. The prediction is
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳      ] is outside the range [0, 1]. Membership value = 0.153744. The prediction is more
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
↳likely incorrect.

```

(continues on next page)



(continued from previous page)

```

>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931 0.
↳79166667] is outside the range [0, 1]. Membership value = 0.207652. The prediction is_
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143 0.41666667 -0.01818182 0.
↳      ] is outside the range [0, 1]. Membership value = 0.728338. The prediction is_
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.207652. The prediction is_
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316 0.4137931 0.375_
↳      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more_
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more_
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931 0.
↳79166667] is outside the range [0, 1]. Membership value = 0.484989. The prediction is_
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143 0.41666667 -0.01818182 0.
↳      ] is outside the range [0, 1]. Membership value = 0.823425. The prediction is_
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.484989. The prediction is_
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316 0.4137931 0.375_
↳      ] is outside the range [0, 1]. Membership value = 0.024189. The prediction is more_
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.024189. The prediction is more_
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931 0.
↳79166667] is outside the range [0, 1]. Membership value = 0.673487. The prediction is_
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143 0.41666667 -0.01818182 0.
↳      ] is outside the range [0, 1]. Membership value = 0.836743. The prediction is_
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.510230. The prediction is_
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316 0.4137931 0.375_
↳      ] is outside the range [0, 1]. Membership value = 0.381343. The prediction is more_
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.381343. The prediction is more_
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931 0.
↳79166667] is outside the range [0, 1]. Membership value = 0.738973. The prediction is_
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143 0.41666667 -0.01818182 0.
↳      ] is outside the range [0, 1]. Membership value = 0.886097. The prediction is_
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.

```

(continues on next page)



(continued from previous page)

```

↪91666667] is outside the range [0, 1]. Membership value = 0.738973. The prediction is ↪
↪more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375 ↪
↪      ] is outside the range [0, 1]. Membership value = 0.505423. The prediction is more ↪
↪likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  ↪
↪      ] is outside the range [0, 1]. Membership value = 0.505423. The prediction is more ↪
↪likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0. ↪
↪79166667] is outside the range [0, 1]. Membership value = 0.505573. The prediction is ↪
↪more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0. ↪
↪      ] is outside the range [0, 1]. Membership value = 0.759850. The prediction is ↪
↪more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0. ↪
↪91666667] is outside the range [0, 1]. Membership value = 0.694354. The prediction is ↪
↪more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375 ↪
↪      ] is outside the range [0, 1]. Membership value = 0.115235. The prediction is more ↪
↪likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  ↪
↪      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more ↪
↪likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0. ↪
↪79166667] is outside the range [0, 1]. Membership value = 0.585234. The prediction is ↪
↪more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0. ↪
↪      ] is outside the range [0, 1]. Membership value = 0.857794. The prediction is ↪
↪more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0. ↪
↪91666667] is outside the range [0, 1]. Membership value = 0.585234. The prediction is ↪
↪more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375 ↪
↪      ] is outside the range [0, 1]. Membership value = 0.476084. The prediction is more ↪
↪likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  ↪
↪      ] is outside the range [0, 1]. Membership value = 0.214127. The prediction is more ↪
↪likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0. ↪
↪79166667] is outside the range [0, 1]. Membership value = 0.650744. The prediction is ↪
↪more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0. ↪
↪      ] is outside the range [0, 1]. Membership value = 0.825372. The prediction is ↪
↪more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0. ↪
↪91666667] is outside the range [0, 1]. Membership value = 0.476116. The prediction is ↪
↪more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375 ↪
↪      ] is outside the range [0, 1]. Membership value = 0.338252. The prediction is more ↪
↪likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  ↪
↪      ] is outside the range [0, 1]. Membership value = 0.338252. The prediction is more ↪

```

(continues on next page)

(continued from previous page)

```

↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.679227. The prediction is↳
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.839613. The prediction is↳
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.518840. The prediction is↳
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375↳
↳      ] is outside the range [0, 1]. Membership value = 0.392219. The prediction is more↳
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.392219. The prediction is more↳
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.666495. The prediction is↳
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.833248. The prediction is↳
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.499743. The prediction is↳
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375↳
↳      ] is outside the range [0, 1]. Membership value = 0.368097. The prediction is more↳
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.368097. The prediction is more↳
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.847477. The prediction is↳
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.923738. The prediction is↳
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.847477. The prediction is↳
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375↳
↳      ] is outside the range [0, 1]. Membership value = 0.711009. The prediction is more↳
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.711009. The prediction is more↳
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.869514. The prediction is↳
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.934757. The prediction is↳
↳more likely incorrect.

```

(continues on next page)

(continued from previous page)

```

>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.869514. The prediction is_
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375_
↳      ] is outside the range [0, 1]. Membership value = 0.752763. The prediction is more_
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  _
↳      ] is outside the range [0, 1]. Membership value = 0.752763. The prediction is more_
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.977404. The prediction is_
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.  _
↳      ] is outside the range [0, 1]. Membership value = 0.990140. The prediction is_
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.977404. The prediction is_
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375_
↳      ] is outside the range [0, 1]. Membership value = 0.957186. The prediction is more_
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  _
↳      ] is outside the range [0, 1]. Membership value = 0.957186. The prediction is more_
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.792677. The prediction is_
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.  _
↳      ] is outside the range [0, 1]. Membership value = 0.896339. The prediction is_
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.792677. The prediction is_
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375_
↳      ] is outside the range [0, 1]. Membership value = 0.607178. The prediction is more_
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  _
↳      ] is outside the range [0, 1]. Membership value = 0.607178. The prediction is more_
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.876281. The prediction is_
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.  _
↳      ] is outside the range [0, 1]. Membership value = 0.946013. The prediction is_
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.876281. The prediction is_
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375_
↳      ] is outside the range [0, 1]. Membership value = 0.765584. The prediction is more_
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  _
↳      ]

```

(continues on next page)

(continued from previous page)

```

→ ] is outside the range [0, 1]. Membership value = 0.765584. The prediction is more
→ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
→ 79166667] is outside the range [0, 1]. Membership value = 0.849137. The prediction is
→ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
→      ] is outside the range [0, 1]. Membership value = 0.934169. The prediction is
→ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
→ 91666667] is outside the range [0, 1]. Membership value = 0.849137. The prediction is
→ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
→      ] is outside the range [0, 1]. Membership value = 0.748562. The prediction is more
→ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
→      ] is outside the range [0, 1]. Membership value = 0.714154. The prediction is more
→ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
→ 79166667] is outside the range [0, 1]. Membership value = 0.912143. The prediction is
→ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
→      ] is outside the range [0, 1]. Membership value = 0.961663. The prediction is
→ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
→ 91666667] is outside the range [0, 1]. Membership value = 0.912143. The prediction is
→ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
→      ] is outside the range [0, 1]. Membership value = 0.833535. The prediction is more
→ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
→      ] is outside the range [0, 1]. Membership value = 0.833535. The prediction is more
→ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
→ 79166667] is outside the range [0, 1]. Membership value = 0.810589. The prediction is
→ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
→      ] is outside the range [0, 1]. Membership value = 0.905295. The prediction is
→ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
→ 91666667] is outside the range [0, 1]. Membership value = 0.715884. The prediction is
→ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
→      ] is outside the range [0, 1]. Membership value = 0.684315. The prediction is more
→ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
→      ] is outside the range [0, 1]. Membership value = 0.641116. The prediction is more
→ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
→ 79166667] is outside the range [0, 1]. Membership value = 0.939373. The prediction is
→ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
→      ] is outside the range [0, 1]. Membership value = 0.969686. The prediction is
→

```

(continues on next page)

(continued from previous page)

```

↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.909059. The prediction is↳
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375↳
↳      ] is outside the range [0, 1]. Membership value = 0.885127. The prediction is more↳
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  ↳
↳      ] is outside the range [0, 1]. Membership value = 0.885127. The prediction is more↳
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.937304. The prediction is↳
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.  ↳
↳      ] is outside the range [0, 1]. Membership value = 0.968652. The prediction is↳
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.905956. The prediction is↳
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375↳
↳      ] is outside the range [0, 1]. Membership value = 0.881207. The prediction is more↳
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  ↳
↳      ] is outside the range [0, 1]. Membership value = 0.881207. The prediction is more↳
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.719306. The prediction is↳
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.  ↳
↳      ] is outside the range [0, 1]. Membership value = 0.877516. The prediction is↳
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.719306. The prediction is↳
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375↳
↳      ] is outside the range [0, 1]. Membership value = 0.532177. The prediction is more↳
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  ↳
↳      ] is outside the range [0, 1]. Membership value = 0.468160. The prediction is more↳
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.398965. The prediction is↳
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.  ↳
↳      ] is outside the range [0, 1]. Membership value = 0.737730. The prediction is↳
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.398965. The prediction is↳
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375↳
↳      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more↳
↳likely incorrect.

```

(continues on next page)

(continued from previous page)

```

>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳ ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
↳ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳ 79166667] is outside the range [0, 1]. Membership value = 0.389053. The prediction is
↳ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳ ] is outside the range [0, 1]. Membership value = 0.733405. The prediction is
↳ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳ 91666667] is outside the range [0, 1]. Membership value = 0.389053. The prediction is
↳ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳ ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
↳ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳ ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
↳ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳ 79166667] is outside the range [0, 1]. Membership value = 0.448095. The prediction is
↳ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳ ] is outside the range [0, 1]. Membership value = 0.759169. The prediction is
↳ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳ 91666667] is outside the range [0, 1]. Membership value = 0.448095. The prediction is
↳ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳ ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
↳ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳ ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
↳ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳ 79166667] is outside the range [0, 1]. Membership value = 0.457615. The prediction is
↳ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳ ] is outside the range [0, 1]. Membership value = 0.763323. The prediction is
↳ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳ 91666667] is outside the range [0, 1]. Membership value = 0.186423. The prediction is
↳ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳ ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
↳ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳ ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
↳ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳ 79166667] is outside the range [0, 1]. Membership value = 0.551708. The prediction is
↳ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳ ] is outside the range [0, 1]. Membership value = 0.763323. The prediction is
↳ more likely incorrect.

```

(continues on next page)



(continued from previous page)

```

→      ] is outside the range [0, 1]. Membership value = 0.782258. The prediction is
→more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
→91666667] is outside the range [0, 1]. Membership value = 0.364920. The prediction is
→more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
→      ] is outside the range [0, 1]. Membership value = 0.197794. The prediction is more
→likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
→      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
→likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
→79166667] is outside the range [0, 1]. Membership value = 0.659884. The prediction is
→more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
→      ] is outside the range [0, 1]. Membership value = 0.834801. The prediction is
→more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
→91666667] is outside the range [0, 1]. Membership value = 0.789746. The prediction is
→more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
→      ] is outside the range [0, 1]. Membership value = 0.391371. The prediction is more
→likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
→      ] is outside the range [0, 1]. Membership value = 0.087057. The prediction is more
→likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
→79166667] is outside the range [0, 1]. Membership value = 0.627892. The prediction is
→more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
→      ] is outside the range [0, 1]. Membership value = 0.837625. The prediction is
→more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
→91666667] is outside the range [0, 1]. Membership value = 0.627892. The prediction is
→more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
→      ] is outside the range [0, 1]. Membership value = 0.379819. The prediction is more
→likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
→      ] is outside the range [0, 1]. Membership value = 0.294953. The prediction is more
→likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
→79166667] is outside the range [0, 1]. Membership value = 0.649130. The prediction is
→more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
→      ] is outside the range [0, 1]. Membership value = 0.824565. The prediction is
→more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
→91666667] is outside the range [0, 1]. Membership value = 0.473694. The prediction is
→more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
→      ] is outside the range [0, 1]. Membership value = 0.335193. The prediction is more
→

```

(continues on next page)

(continued from previous page)

```

↳ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳ ] is outside the range [0, 1]. Membership value = 0.335193. The prediction is more
↳ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳ 79166667] is outside the range [0, 1]. Membership value = 0.277219. The prediction is
↳ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳ ] is outside the range [0, 1]. Membership value = 0.684605. The prediction is
↳ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳ 91666667] is outside the range [0, 1]. Membership value = 0.000000. The prediction is
↳ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳ ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
↳ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳ ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
↳ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳ 79166667] is outside the range [0, 1]. Membership value = 0.708327. The prediction is
↳ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳ ] is outside the range [0, 1]. Membership value = 0.899998. The prediction is
↳ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳ 91666667] is outside the range [0, 1]. Membership value = 0.708327. The prediction is
↳ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳ ] is outside the range [0, 1]. Membership value = 0.631571. The prediction is more
↳ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳ ] is outside the range [0, 1]. Membership value = 0.447357. The prediction is more
↳ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳ 79166667] is outside the range [0, 1]. Membership value = 0.683243. The prediction is
↳ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳ ] is outside the range [0, 1]. Membership value = 0.846147. The prediction is
↳ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳ 91666667] is outside the range [0, 1]. Membership value = 0.551261. The prediction is
↳ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳ ] is outside the range [0, 1]. Membership value = 0.433172. The prediction is more
↳ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳ ] is outside the range [0, 1]. Membership value = 0.149758. The prediction is more
↳ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳ 79166667] is outside the range [0, 1]. Membership value = 0.760300. The prediction is
↳ more likely incorrect.

```

(continues on next page)



(continued from previous page)

```

>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.917817. The prediction is
↳ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳ 91666667] is outside the range [0, 1]. Membership value = 0.760300. The prediction is
↳ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳      ] is outside the range [0, 1]. Membership value = 0.697221. The prediction is more
↳ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.545832. The prediction is more
↳ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳ 79166667] is outside the range [0, 1]. Membership value = 0.690040. The prediction is
↳ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.845020. The prediction is
↳ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳ 91666667] is outside the range [0, 1]. Membership value = 0.535061. The prediction is
↳ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳      ] is outside the range [0, 1]. Membership value = 0.412708. The prediction is more
↳ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.412708. The prediction is more
↳ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳ 79166667] is outside the range [0, 1]. Membership value = 0.805245. The prediction is
↳ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.902623. The prediction is
↳ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳ 91666667] is outside the range [0, 1]. Membership value = 0.707868. The prediction is
↳ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳      ] is outside the range [0, 1]. Membership value = 0.630991. The prediction is more
↳ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.630991. The prediction is more
↳ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳ 79166667] is outside the range [0, 1]. Membership value = 0.620962. The prediction is
↳ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.870044. The prediction is
↳ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳ 91666667] is outside the range [0, 1]. Membership value = 0.620962. The prediction is
↳ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳      ] is outside the range [0, 1]. Membership value = 0.630991. The prediction is more
↳ likely incorrect.

```

(continues on next page)

(continued from previous page)

```

→ ] is outside the range [0, 1]. Membership value = 0.521215. The prediction is more_
→ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
→ ] is outside the range [0, 1]. Membership value = 0.281822. The prediction is more_
→ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75 0.9137931 0.
→ 79166667] is outside the range [0, 1]. Membership value = 0.906224. The prediction is_
→ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143 0.41666667 -0.01818182 0.
→ ] is outside the range [0, 1]. Membership value = 0.959079. The prediction is_
→ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25 1.03636364 0.
→ 91666667] is outside the range [0, 1]. Membership value = 0.906224. The prediction is_
→ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316 0.4137931 0.375_
→ ] is outside the range [0, 1]. Membership value = 0.822319. The prediction is more_
→ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
→ ] is outside the range [0, 1]. Membership value = 0.822319. The prediction is more_
→ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75 0.9137931 0.
→ 79166667] is outside the range [0, 1]. Membership value = 0.864965. The prediction is_
→ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143 0.41666667 -0.01818182 0.
→ ] is outside the range [0, 1]. Membership value = 0.941076. The prediction is_
→ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25 1.03636364 0.
→ 91666667] is outside the range [0, 1]. Membership value = 0.864965. The prediction is_
→ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316 0.4137931 0.375_
→ ] is outside the range [0, 1]. Membership value = 0.744145. The prediction is more_
→ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
→ ] is outside the range [0, 1]. Membership value = 0.744145. The prediction is more_
→ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75 0.9137931 0.
→ 79166667] is outside the range [0, 1]. Membership value = 0.251775. The prediction is_
→ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143 0.41666667 -0.01818182 0.
→ ] is outside the range [0, 1]. Membership value = 0.625888. The prediction is_
→ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25 1.03636364 0.
→ 91666667] is outside the range [0, 1]. Membership value = 0.000000. The prediction is_
→ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316 0.4137931 0.375_
→ ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more_
→ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
→ ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more_
→ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75 0.9137931 0.
→ 79166667] is outside the range [0, 1]. Membership value = 0.172844. The prediction is_

```

(continues on next page)

(continued from previous page)

```

↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.586422. The prediction is↳
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.000000. The prediction is↳
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375↳
↳      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more↳
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  ↳
↳      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more↳
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.532322. The prediction is↳
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.  ↳
↳      ] is outside the range [0, 1]. Membership value = 0.772842. The prediction is↳
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.566335. The prediction is↳
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375↳
↳      ] is outside the range [0, 1]. Membership value = 0.163103. The prediction is more↳
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  ↳
↳      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more↳
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.415407. The prediction is↳
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.  ↳
↳      ] is outside the range [0, 1]. Membership value = 0.744905. The prediction is↳
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.123111. The prediction is↳
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375↳
↳      ] is outside the range [0, 1]. Membership value = 0.025679. The prediction is more↳
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  ↳
↳      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more↳
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.431079. The prediction is↳
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.  ↳
↳      ] is outside the range [0, 1]. Membership value = 0.751744. The prediction is↳
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.146619. The prediction is↳
↳more likely incorrect.

```

(continues on next page)

(continued from previous page)

```

>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳      ] is outside the range [0, 1]. Membership value = 0.051799. The prediction is more
↳ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
↳ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳ 79166667] is outside the range [0, 1]. Membership value = 0.496359. The prediction is
↳ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.780230. The prediction is
↳ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳ 91666667] is outside the range [0, 1]. Membership value = 0.496359. The prediction is
↳ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳      ] is outside the range [0, 1]. Membership value = 0.045733. The prediction is more
↳ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.045733. The prediction is more
↳ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳ 79166667] is outside the range [0, 1]. Membership value = 0.518529. The prediction is
↳ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.766143. The prediction is
↳ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳ 91666667] is outside the range [0, 1]. Membership value = 0.702363. The prediction is
↳ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳      ] is outside the range [0, 1]. Membership value = 0.138421. The prediction is more
↳ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
↳ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳ 79166667] is outside the range [0, 1]. Membership value = 0.999140. The prediction is
↳ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.999625. The prediction is
↳ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳ 91666667] is outside the range [0, 1]. Membership value = 0.999140. The prediction is
↳ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳      ] is outside the range [0, 1]. Membership value = 0.998913. The prediction is more
↳ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.998370. The prediction is more
↳ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.

```

(continues on next page)

(continued from previous page)

```

↪79166667] is outside the range [0, 1]. Membership value = 0.559096. The prediction is_
↪more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↪      ] is outside the range [0, 1]. Membership value = 0.807605. The prediction is_
↪more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↪91666667] is outside the range [0, 1]. Membership value = 0.559096. The prediction is_
↪more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375_
↪      ] is outside the range [0, 1]. Membership value = 0.164603. The prediction is more_
↪likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  _
↪      ] is outside the range [0, 1]. Membership value = 0.164603. The prediction is more_
↪likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↪79166667] is outside the range [0, 1]. Membership value = 0.379206. The prediction is_
↪more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↪      ] is outside the range [0, 1]. Membership value = 0.729108. The prediction is_
↪more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↪91666667] is outside the range [0, 1]. Membership value = 0.068810. The prediction is_
↪more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375_
↪      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more_
↪likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  _
↪      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more_
↪likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↪79166667] is outside the range [0, 1]. Membership value = 0.377855. The prediction is_
↪more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↪      ] is outside the range [0, 1]. Membership value = 0.728518. The prediction is_
↪more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↪91666667] is outside the range [0, 1]. Membership value = 0.066782. The prediction is_
↪more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375_
↪      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more_
↪likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  _
↪      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more_
↪likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↪79166667] is outside the range [0, 1]. Membership value = 0.224555. The prediction is_
↪more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↪      ] is outside the range [0, 1]. Membership value = 0.612278. The prediction is_
↪more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↪91666667] is outside the range [0, 1]. Membership value = 0.000000. The prediction is_

```

(continues on next page)

(continued from previous page)

```

↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.959355. The prediction is
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.980258. The prediction is
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.942419. The prediction is
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳      ] is outside the range [0, 1]. Membership value = 0.927266. The prediction is more
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.890899. The prediction is more
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.516105. The prediction is
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.834093. The prediction is
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.516105. The prediction is
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳      ] is outside the range [0, 1]. Membership value = 0.388765. The prediction is more
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.083147. The prediction is more
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.356339. The prediction is
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.719130. The prediction is
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.356339. The prediction is
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
↳likely incorrect.

```

(continues on next page)



(continued from previous page)

```

>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.753394. The prediction is_
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.876697. The prediction is_
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.753394. The prediction is_
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375_
↳      ] is outside the range [0, 1]. Membership value = 0.532747. The prediction is more_
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.532747. The prediction is more_
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.957302. The prediction is_
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.978651. The prediction is_
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.935953. The prediction is_
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375_
↳      ] is outside the range [0, 1]. Membership value = 0.919098. The prediction is more_
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.919098. The prediction is more_
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.973522. The prediction is_
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.986761. The prediction is_
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.960283. The prediction is_
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375_
↳      ] is outside the range [0, 1]. Membership value = 0.949831. The prediction is more_
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.949831. The prediction is more_
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.997910. The prediction is_
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.999088. The prediction is_
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.

```

(continues on next page)

(continued from previous page)

```

↪91666667] is outside the range [0, 1]. Membership value = 0.997910. The prediction is ↪
↪more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375 ↪
↪ ] is outside the range [0, 1]. Membership value = 0.996040. The prediction is more ↪
↪likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125 ↪
↪ ] is outside the range [0, 1]. Membership value = 0.996040. The prediction is more ↪
↪likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0. ↪
↪79166667] is outside the range [0, 1]. Membership value = 0.821118. The prediction is ↪
↪more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0. ↪
↪ ] is outside the range [0, 1]. Membership value = 0.910559. The prediction is ↪
↪more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0. ↪
↪91666667] is outside the range [0, 1]. Membership value = 0.731677. The prediction is ↪
↪more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375 ↪
↪ ] is outside the range [0, 1]. Membership value = 0.661065. The prediction is more ↪
↪likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125 ↪
↪ ] is outside the range [0, 1]. Membership value = 0.661065. The prediction is more ↪
↪likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0. ↪
↪79166667] is outside the range [0, 1]. Membership value = 0.886690. The prediction is ↪
↪more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0. ↪
↪ ] is outside the range [0, 1]. Membership value = 0.943345. The prediction is ↪
↪more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0. ↪
↪91666667] is outside the range [0, 1]. Membership value = 0.886690. The prediction is ↪
↪more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375 ↪
↪ ] is outside the range [0, 1]. Membership value = 0.785307. The prediction is more ↪
↪likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125 ↪
↪ ] is outside the range [0, 1]. Membership value = 0.785307. The prediction is more ↪
↪likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0. ↪
↪79166667] is outside the range [0, 1]. Membership value = 0.194951. The prediction is ↪
↪more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0. ↪
↪ ] is outside the range [0, 1]. Membership value = 0.597475. The prediction is ↪
↪more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0. ↪
↪91666667] is outside the range [0, 1]. Membership value = 0.000000. The prediction is ↪
↪more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375 ↪
↪ ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more ↪
↪likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125 ↪
↪ ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more ↪

```

(continues on next page)



(continued from previous page)

```

↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931 0.
↳79166667] is outside the range [0, 1]. Membership value = 0.459018. The prediction is↳
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143 0.41666667 -0.01818182 0.
↳      ] is outside the range [0, 1]. Membership value = 0.763935. The prediction is↳
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.188527. The prediction is↳
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316 0.4137931 0.375↳
↳      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more↳
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more↳
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931 0.
↳79166667] is outside the range [0, 1]. Membership value = 0.933546. The prediction is↳
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143 0.41666667 -0.01818182 0.
↳      ] is outside the range [0, 1]. Membership value = 0.971002. The prediction is↳
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.933546. The prediction is↳
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316 0.4137931 0.375↳
↳      ] is outside the range [0, 1]. Membership value = 0.874086. The prediction is more↳
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.874086. The prediction is more↳
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931 0.
↳79166667] is outside the range [0, 1]. Membership value = 0.726992. The prediction is↳
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143 0.41666667 -0.01818182 0.
↳      ] is outside the range [0, 1]. Membership value = 0.863496. The prediction is↳
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.726992. The prediction is↳
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316 0.4137931 0.375↳
↳      ] is outside the range [0, 1]. Membership value = 0.482722. The prediction is more↳
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.482722. The prediction is more↳
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931 0.
↳79166667] is outside the range [0, 1]. Membership value = 0.604519. The prediction is↳
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143 0.41666667 -0.01818182 0.
↳      ] is outside the range [0, 1]. Membership value = 0.827427. The prediction is↳
↳more likely incorrect.

```

(continues on next page)

(continued from previous page)

```

>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.604519. The prediction is.
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375.
↳      ] is outside the range [0, 1]. Membership value = 0.250668. The prediction is more.
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.250668. The prediction is more.
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.716278. The prediction is.
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.858139. The prediction is.
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.574417. The prediction is.
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375.
↳      ] is outside the range [0, 1]. Membership value = 0.462421. The prediction is more.
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.462421. The prediction is more.
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.795196. The prediction is.
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.897598. The prediction is.
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.692794. The prediction is.
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375.
↳      ] is outside the range [0, 1]. Membership value = 0.611951. The prediction is more.
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.611951. The prediction is more.
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.656288. The prediction is.
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.828144. The prediction is.
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.484432. The prediction is.
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375.
↳      ] is outside the range [0, 1]. Membership value = 0.348757. The prediction is more.
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳

```

(continues on next page)

(continued from previous page)

```

→ ] is outside the range [0, 1]. Membership value = 0.348757. The prediction is more
→ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931 0.
→ 79166667] is outside the range [0, 1]. Membership value = 0.825042. The prediction is
→ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143 0.41666667 -0.01818182 0.
→      ] is outside the range [0, 1]. Membership value = 0.923655. The prediction is
→ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
→ 91666667] is outside the range [0, 1]. Membership value = 0.825042. The prediction is
→ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316 0.4137931 0.375
→      ] is outside the range [0, 1]. Membership value = 0.668500. The prediction is more
→ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
→      ] is outside the range [0, 1]. Membership value = 0.668500. The prediction is more
→ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931 0.
→ 79166667] is outside the range [0, 1]. Membership value = 0.771271. The prediction is
→ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143 0.41666667 -0.01818182 0.
→      ] is outside the range [0, 1]. Membership value = 0.921579. The prediction is
→ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
→ 91666667] is outside the range [0, 1]. Membership value = 0.771271. The prediction is
→ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316 0.4137931 0.375
→      ] is outside the range [0, 1]. Membership value = 0.711079. The prediction is more
→ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
→      ] is outside the range [0, 1]. Membership value = 0.566619. The prediction is more
→ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931 0.
→ 79166667] is outside the range [0, 1]. Membership value = 0.696148. The prediction is
→ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143 0.41666667 -0.01818182 0.
→      ] is outside the range [0, 1]. Membership value = 0.867410. The prediction is
→ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
→ 91666667] is outside the range [0, 1]. Membership value = 0.696148. The prediction is
→ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316 0.4137931 0.375
→      ] is outside the range [0, 1]. Membership value = 0.493580. The prediction is more
→ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
→      ] is outside the range [0, 1]. Membership value = 0.424281. The prediction is more
→ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931 0.
→ 79166667] is outside the range [0, 1]. Membership value = 0.566989. The prediction is
→ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143 0.41666667 -0.01818182 0.
→      ] is outside the range [0, 1]. Membership value = 0.811050. The prediction is
→

```

(continues on next page)

(continued from previous page)

```

↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.566989. The prediction is↳
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375↳
↳      ] is outside the range [0, 1]. Membership value = 0.453039. The prediction is more↳
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  ↳
↳      ] is outside the range [0, 1]. Membership value = 0.179558. The prediction is more↳
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.658673. The prediction is↳
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.  ↳
↳      ] is outside the range [0, 1]. Membership value = 0.829336. The prediction is↳
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.488009. The prediction is↳
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375↳
↳      ] is outside the range [0, 1]. Membership value = 0.353275. The prediction is more↳
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  ↳
↳      ] is outside the range [0, 1]. Membership value = 0.353275. The prediction is more↳
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.854032. The prediction is↳
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.  ↳
↳      ] is outside the range [0, 1]. Membership value = 0.927016. The prediction is↳
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.854032. The prediction is↳
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375↳
↳      ] is outside the range [0, 1]. Membership value = 0.723428. The prediction is more↳
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125  ↳
↳      ] is outside the range [0, 1]. Membership value = 0.723428. The prediction is more↳
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.775829. The prediction is↳
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.  ↳
↳      ] is outside the range [0, 1]. Membership value = 0.902180. The prediction is↳
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.775829. The prediction is↳
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375↳
↳      ] is outside the range [0, 1]. Membership value = 0.575256. The prediction is more↳
↳likely incorrect.

```

(continues on next page)

(continued from previous page)

```

>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳ ] is outside the range [0, 1]. Membership value = 0.575256. The prediction is more
↳ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳ 79166667] is outside the range [0, 1]. Membership value = 0.898866. The prediction is
↳ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳ ] is outside the range [0, 1]. Membership value = 0.955869. The prediction is
↳ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳ 91666667] is outside the range [0, 1]. Membership value = 0.898866. The prediction is
↳ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳ ] is outside the range [0, 1]. Membership value = 0.808378. The prediction is more
↳ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳ ] is outside the range [0, 1]. Membership value = 0.808378. The prediction is more
↳ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳ 79166667] is outside the range [0, 1]. Membership value = 0.687186. The prediction is
↳ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳ ] is outside the range [0, 1]. Membership value = 0.863499. The prediction is
↳ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳ 91666667] is outside the range [0, 1]. Membership value = 0.687186. The prediction is
↳ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳ ] is outside the range [0, 1]. Membership value = 0.407299. The prediction is more
↳ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳ ] is outside the range [0, 1]. Membership value = 0.407299. The prediction is more
↳ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳ 79166667] is outside the range [0, 1]. Membership value = 0.636231. The prediction is
↳ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳ ] is outside the range [0, 1]. Membership value = 0.841264. The prediction is
↳ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳ 91666667] is outside the range [0, 1]. Membership value = 0.636231. The prediction is
↳ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳ ] is outside the range [0, 1]. Membership value = 0.310753. The prediction is more
↳ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳ ] is outside the range [0, 1]. Membership value = 0.310753. The prediction is more
↳ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳ 79166667] is outside the range [0, 1]. Membership value = 0.607021. The prediction is
↳ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳ ]

```

(continues on next page)

(continued from previous page)

```

→      ] is outside the range [0, 1]. Membership value = 0.865264. The prediction is
→ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
→ 91666667] is outside the range [0, 1]. Membership value = 0.607021. The prediction is
→ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
→      ] is outside the range [0, 1]. Membership value = 0.503605. The prediction is more
→ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
→      ] is outside the range [0, 1]. Membership value = 0.255408. The prediction is more
→ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
→ 79166667] is outside the range [0, 1]. Membership value = 0.814544. The prediction is
→ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
→      ] is outside the range [0, 1]. Membership value = 0.909921. The prediction is
→ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
→ 91666667] is outside the range [0, 1]. Membership value = 0.885354. The prediction is
→ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
→      ] is outside the range [0, 1]. Membership value = 0.668131. The prediction is more
→ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
→      ] is outside the range [0, 1]. Membership value = 0.502196. The prediction is more
→ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
→ 79166667] is outside the range [0, 1]. Membership value = 0.683919. The prediction is
→ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
→      ] is outside the range [0, 1]. Membership value = 0.841960. The prediction is
→ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
→ 91666667] is outside the range [0, 1]. Membership value = 0.525879. The prediction is
→ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
→      ] is outside the range [0, 1]. Membership value = 0.401110. The prediction is more
→ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
→      ] is outside the range [0, 1]. Membership value = 0.401110. The prediction is more
→ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75      0.9137931  0.
→ 79166667] is outside the range [0, 1]. Membership value = 0.284114. The prediction is
→ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
→      ] is outside the range [0, 1]. Membership value = 0.687613. The prediction is
→ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25      1.03636364 0.
→ 91666667] is outside the range [0, 1]. Membership value = 0.284114. The prediction is
→ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
→      ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
→

```

(continues on next page)



(continued from previous page)

```

↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳ ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.809766. The prediction is
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳ ] is outside the range [0, 1]. Membership value = 0.934777. The prediction is
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.809766. The prediction is
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳ ] is outside the range [0, 1]. Membership value = 0.759704. The prediction is more
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳ ] is outside the range [0, 1]. Membership value = 0.639556. The prediction is more
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.888344. The prediction is
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳ ] is outside the range [0, 1]. Membership value = 0.951277. The prediction is
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.832516. The prediction is
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳ ] is outside the range [0, 1]. Membership value = 0.813907. The prediction is more
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳ ] is outside the range [0, 1]. Membership value = 0.788442. The prediction is more
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.394295. The prediction is
↳more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳ ] is outside the range [0, 1]. Membership value = 0.735692. The prediction is
↳more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳91666667] is outside the range [0, 1]. Membership value = 0.394295. The prediction is
↳more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳ ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
↳likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳ ] is outside the range [0, 1]. Membership value = 0.000000. The prediction is more
↳likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳79166667] is outside the range [0, 1]. Membership value = 0.840977. The prediction is
↳more likely incorrect.

```

(continues on next page)

(continued from previous page)

```

>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.930608. The prediction is
↳ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳ 91666667] is outside the range [0, 1]. Membership value = 0.840977. The prediction is
↳ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳      ] is outside the range [0, 1]. Membership value = 0.734961. The prediction is more
↳ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.698693. The prediction is more
↳ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳ 79166667] is outside the range [0, 1]. Membership value = 0.478630. The prediction is
↳ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.739315. The prediction is
↳ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳ 91666667] is outside the range [0, 1]. Membership value = 0.478630. The prediction is
↳ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳      ] is outside the range [0, 1]. Membership value = 0.012141. The prediction is more
↳ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.012141. The prediction is more
↳ likely incorrect.
>>> The testing sample 19 with the coordinate [1.05882353 0.75          0.9137931  0.
↳ 79166667] is outside the range [0, 1]. Membership value = 0.540064. The prediction is
↳ more likely incorrect.
>>> The testing sample 11 with the coordinate [-0.02857143  0.41666667 -0.01818182  0.
↳      ] is outside the range [0, 1]. Membership value = 0.770032. The prediction is
↳ more likely incorrect.
>>> The testing sample 21 with the coordinate [0.94285714 0.25          1.03636364 0.
↳ 91666667] is outside the range [0, 1]. Membership value = 0.540064. The prediction is
↳ more likely incorrect.
>>> The testing sample 6 with the coordinate [ 0.19444444 -0.10526316  0.4137931  0.375
↳      ] is outside the range [0, 1]. Membership value = 0.128542. The prediction is more
↳ likely incorrect.
>>> The testing sample 12 with the coordinate [0.38888889 1.15789474 0.06896552 0.125
↳      ] is outside the range [0, 1]. Membership value = 0.128542. The prediction is more
↳ likely incorrect.
100%| 100/100 [00:16<00:00, 6.08trial/s, best loss: -0.9666666666666666]

```

```

[16]: # Building and fitting classifier with best parameters
      clf = init_model(space_eval(hp_space_gfmm, best_clf)).fit(X_train, y_train)

      # Calculating performance on validation set
      clf_val_score = accuracy_score(y_test, clf.predict(X_test))
      print('Cross-val score: {0:.5f}; testing score: {1:.5f}'.\
            format(-trials_clf.best_trial['result']['loss'], clf_val_score))
      print('Best parameters:')

```

(continues on next page)



(continued from previous page)

```
print(space_eval(hp_space_gfmm, best_clf))

>>> The testing sample 26 with the coordinate [ 0.08333333  0.66666667 -0.01724138  0.
↳ 0.41666667] is outside the range [0, 1]. Membership value = 0.478630. The prediction is.
↳ more likely incorrect.
Cross-val score: 0.96667; testing score: 0.96667
Best parameters:
{'gamma': 6.256443392850102, 'theta': 0.09273471494941352}
```

```
[17]: def f_wrap_space_eval(hp_space, trial):
    """
    Utility function for more consise optimization history extraction

    Parameters:
    -----
    hp_space : hyperspace from which points are sampled
    trial : hyperopt.Trials object

    Returns:
    -----
    : dict(
        k: v
    ), where k - label of hyperparameter, v - value of hyperparameter in trial
    """

    return space_eval(hp_space, {k: v[0] for (k, v) in trial['misc']['vals'].items() if
↳ len(v) > 0})

def f_unpack_dict(dct):
    """
    Unpacks all sub-dictionaries in given dictionary recursively. There should be no
↳ duplicated keys
    across all nested subdictionaries, or some instances will be lost without warning

    Parameters:
    -----
    dct : dictionary to unpack

    Returns:
    -----
    : unpacked dictionary
    """

    res = {}
    for (k, v) in dct.items():
        if isinstance(v, dict):
            res = {**res, **f_unpack_dict(v)}
        else:
            res[k] = v

    return res
```

```
[18]: import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from matplotlib.animation import FuncAnimation
import seaborn as sns
```

```
[19]: fig0 = plt.figure(figsize=(12, 10))
gs = gridspec.GridSpec(nrows=3, ncols=1, hspace=0.5, wspace=0.3)

# =====
# Plotting optimization history

ax = fig0.add_subplot(gs[0, 0])
ax.plot(range(1, len(trials_clf) + 1), [-x['result']['loss'] for x in trials_clf],
        color='red', marker='.', linewidth=0)

ax.set_xlabel('Iteration', fontsize=12)
ax.set_ylabel('Accuracy', fontsize=12)
ax.set_title('Optimization history', fontsize=14)

ax.grid(True)

# =====
# Plotting sampled points

samples = [f_unpack_dict(f_wrap_space_eval(hp_space_gfmm, x)) for x in trials_clf.trials]

ax = fig0.add_subplot(gs[1, 0])
sns.histplot(x=[x['gamma'] for x in samples], bins=10, ax=ax)

ax.set_xlabel('$gamma$', fontsize=10)
ax.set_ylabel('Counts', fontsize=10)

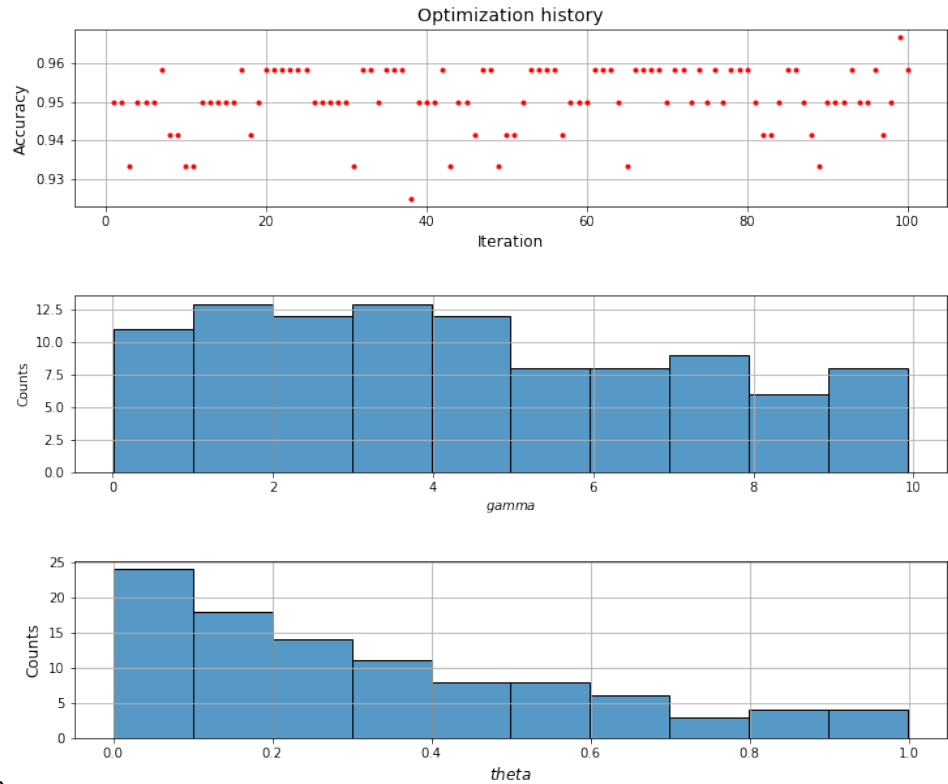
ax.grid(True)

# -----

ax = fig0.add_subplot(gs[2, 0])
sns.histplot(x=[x['theta'] for x in samples], bins=10, ax=ax)

ax.set_xlabel('$theta$', fontsize=12)
ax.set_ylabel('Counts', fontsize=12)

ax.grid(True)
```



nbsphinx-code-borderwhite

## Integration of Ensemble Models with Sklearn Pipeline

This example shows how to integrate the random hyperboxes classifier into the Pipeline class implemented by scikit-learn.

Note that this example is illustrated by using the random hyperboxes model with original online learning algorithm for training base learners. However, it can be used for any ensemble model of GFMM classifiers using other learning algorithms.

```
[1]: import warnings
warnings.filterwarnings('ignore')
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from hbbrain.numerical_data.incremental_learner.online_gfmm import OnlineGFMM
from hbbrain.numerical_data.ensemble_learner.random_hyperboxes import
↳ RandomHyperboxesClassifier
```

### Load dataset.

This example will use the breast cancer dataset in sklearn for illustration purposes.

```
[2]: from sklearn.datasets import load_breast_cancer
```

```
[3]: df = load_breast_cancer()
      X = df.data
      y = df.target
```

```
[4]: X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=0)
```

### Create a pipeline of pre-processing method (i.e., normalization of data in the range of [0, 1]) and a Random hyperboxes model.

**Note:** the GFMM classifiers requires the input data in the range of [0, 1].

```
[5]: theta = 0.1
      theta_min = 0.1
      base_estimator = OnlineGFMM(theta=theta, theta_min=theta_min)
      n_estimators = 50
      max_samples = 0.5
      max_features = 0.5
      class_balanced = False
      feature_balanced = False
      n_jobs = 4
      # Init a classifier
      rh_clf = RandomHyperboxesClassifier(base_estimator=base_estimator, n_estimators=n_
      ↪ estimators, max_samples=max_samples, max_features=max_features, class_balanced=class_
      ↪ balanced, feature_balanced=feature_balanced, n_jobs=n_jobs, random_state=0)
```

```
[6]: # create a pipeline including data pre-processing and a classifier
      pipe = Pipeline([('scaler', MinMaxScaler()), ('rh_clf', rh_clf)])
```

### Training

```
[7]: pipe.fit(X_train, y_train)
```

```
[7]: Pipeline(steps=[('scaler', MinMaxScaler()),
                      ('rh_clf',
                       RandomHyperboxesClassifier(base_estimator=OnlineGFMM(C=array([],
      ↪ dtype=float64),
                                                                                       V=array([],
      ↪ dtype=float64),
                                                                                       W=array([],
      ↪ dtype=float64),
                                                                                       theta=0.1,
                                                                                       theta_min=0.1),
                                                                                       max_features=0.5, n_estimators=50,
                                                                                       n_jobs=4, random_state=0)))]])
```

## Prediction

```
[8]: acc = pipe.score(X_test, y_test)
print(f'Testing accuracy = {acc * 100: .2f}%')
```

```
Testing accuracy = 96.49%
```

## 2.8.7 Integration with sklearn hyperparameter optimisation

### Integration of Single Hyperbox-based Estimators with Grid-Search and Random-Search in sklearn

This example shows how to integrate the GFMM classifier with the Grid Search Cross-Validation and Random Search Cross-Validation functionalities implemented by scikit-learn

Note that this example will use the original online learning algorithm of GFMM model for demonstration of the integration of Grid Search and Random Search with hyperbox-based model. However, this characteristic can be similarly applied for all of the other hyperbox-based machine learning algorithms.

```
[1]: import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from hbbbrain.numerical_data.incremental_learner.onln_gfmm import OnlineGFMM
```

### Load Iris dataset, normalize it into the range of [0, 1] and build training and testing datasets

```
[2]: from sklearn.datasets import load_iris
```

```
[3]: df = load_iris()
X = df.data
y = df.target
```

```
[4]: scaler = MinMaxScaler()
X = scaler.fit_transform(X)
```

```
[5]: X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=0)
```

### 1. Using Grid Search with 5-fold cross-validation

```
[6]: import numpy as np
from sklearn.metrics import accuracy_score
```

```
[7]: parameters = {'theta': np.arange(0.05, 1.01, 0.05), 'theta_min':[1], 'gamma':[0.5, 1, 2, 4, 8, 16]}
```

```
[8]: onln_gfmm = OnlineGFMM()
     clf_grid_search = GridSearchCV(onln_gfmm, parameters, cv=5, scoring='accuracy',
     ↪refit=True)
```

```
[9]: clf_grid_search.fit(X_train, y_train)
     print("Best average score = ", clf_grid_search.best_score_)
     print("Best params: ", clf_grid_search.best_params_)
```

```
Best average score = 0.9583333333333334
Best params: {'gamma': 0.5, 'theta': 0.3, 'theta_min': 1}
```

```
[10]: best_gfmm_grid_search = clf_grid_search.best_estimator_
```

```
[11]: # Testing the performance on the test set
     y_pred = best_gfmm_grid_search.predict(X_test)
```

```
[12]: acc_grid_search = accuracy_score(y_test, y_pred)
     print(f'Accuracy (grid-search) = {acc_grid_search * 100: .2f}%')
```

```
Accuracy (grid-search) = 96.67%
```

```
[13]: # Try another way to create the best classifier
     best_gfmm_grid_search_2 = OnlineGFMM(**clf_grid_search.best_params_)
     #best_gfmm_grid_search_2.set_params(**clf_grid_search.best_params_)
```

```
[14]: # Training
     best_gfmm_grid_search_2.fit(X_train, y_train)
```

```
[14]: OnlineGFMM(C=array([2, 1, 0, 1, 2, 2, 1, 2, 0, 0, 1, 0, 2, 2, 1]),
     V=array([[0.44444444, 0.29166667, 0.6440678 , 0.70833333],
     [0.25      , 0.125      , 0.42372881, 0.375      ],
     [0.11111111, 0.45833333, 0.03389831, 0.04166667],
     [0.16666667, 0.        , 0.33898305, 0.375      ],
     [0.38888889, 0.08333333, 0.68221339, 0.58333333],
     [0.77777778, 0.41666667, 0.83050847, 0.70833333],
     [0.47222222, 0.375      , 0.55932203, 0.5        ],
     [0.166666...
     [0.16666667, 0.20833333, 0.59322034, 0.66666667],
     [0.19444444, 0.58333333, 0.10169492, 0.08333333],
     [0.41666667, 1.        , 0.11864407, 0.125      ],
     [0.55555556, 0.20833333, 0.66101695, 0.58333333],
     [0.05555556, 0.125      , 0.05084746, 0.08333333],
     [0.94444444, 0.41666667, 1.        , 0.91666667],
     [1.        , 0.75      , 0.96610169, 0.875      ],
     [0.44444444, 0.5        , 0.6440678 , 0.70833333]]),
     gamma=0.5, theta=0.3, theta_min=0.3)
```

```
[15]: # predict
     y_pred_2 = best_gfmm_grid_search_2.predict(X_test)
```

```
[16]: acc_grid_search_2 = accuracy_score(y_test, y_pred_2)
     print(f'Accuracy (grid-search) = {acc_grid_search_2 * 100: .2f}%')
```

```
Accuracy (grid-search) = 96.67%
```

## 2. Using Random Search with 5-fold cross-validation

```
[17]: # Using random search with only 20 random combinations of parameters
      onln_gfmm_rd_search = OnlineGFMM()
      clf_rd_search = RandomizedSearchCV(onln_gfmm_rd_search, parameters, n_iter=20, cv=5,
      ↪random_state=0)
```

```
[18]: clf_rd_search.fit(X_train, y_train)
      print("Best average score = ", clf_rd_search.best_score_)
      print("Best params: ", clf_rd_search.best_params_)

      Best average score = 0.9583333333333334
      Best params: {'theta_min': 1, 'theta': 0.3, 'gamma': 2}
```

```
[19]: best_gfmm_rd_search = clf_rd_search.best_estimator_
```

```
[20]: # Testing the performance on the test set
      y_pred_rd_search = best_gfmm_rd_search.predict(X_test)
```

```
[21]: acc_rd_search = accuracy_score(y_test, y_pred_rd_search)
      print(f'Accuracy (random-search) = {acc_rd_search * 100: .2f}%')

      Accuracy (random-search) = 96.67%
```

### Try to show explanation for an input sample

```
[22]: sample_need_explain = 10
      y_pred_input_0, mem_val_classes, min_points_classes, max_points_classes = best_gfmm_rd_
      ↪search.get_sample_explanation(X_test[sample_need_explain], X_test[sample_need_explain])
```

```
[23]: print("Predicted class for sample X = %s is %d and real class is %d" % (X_test[sample_
      ↪need_explain], y_pred_input_0, y_test[sample_need_explain]))

      Predicted class for sample X = [0.5          0.25          0.77966102 0.54166667] is 2 and
      ↪real class is 2
```

```
[24]: print("Membership values:")
      for key, val in mem_val_classes.items():
          print("Class %d has the maximum membership value = %f" % (key, val))

      for key in min_points_classes:
          print("Class %d has the representative hyperbox: V = %s and W = %s" % (key, min_
          ↪points_classes[key], max_points_classes[key]))

      Membership values:
      Class 0 has the maximum membership value = 0.000000
      Class 1 has the maximum membership value = 0.805085
```

(continues on next page)

(continued from previous page)

```

Class 2 has the maximum membership value = 0.916667
Class 0 has the representative hyperbox: V = [0.11111111 0.45833333 0.03389831 0.
↪ 0.41666667] and W = [0.38888889 0.75      0.11864407 0.20833333]
Class 1 has the representative hyperbox: V = [0.25      0.125      0.42372881 0.375
↪ ] and W = [0.5      0.41666667 0.68220339 0.625      ]
Class 2 has the representative hyperbox: V = [0.38888889 0.08333333 0.68221339 0.
↪ 0.58333333] and W = [0.66666667 0.33333333 0.81355932 0.79166667]

```

### Show explanation results by parallel coordinates

```

[25]: # Create a parallel coordinates graph
best_gfmm_rd_search.show_sample_explanation(X_test[sample_need_explain], X_test[sample_
↪ need_explain], min_points_classes, max_points_classes, y_pred_input_0, file_path="par_
↪ cord/iris_par_cord.html")

```

```

[26]: # Load parallel coordinates to display on the notebook
from IPython.display import IFrame
# We load the parallel coordinates from GitHub here for demonstration in readthedocs
# On the local notebook, we only need to load from the graph storing at 'par_cord/iris_
↪ par_cord.html'
IFrame('https://uts-caslab.github.io/hyperbox-brain/docs/tutorials/par_cord/iris_par_
↪ cord.html', width=820, height=520)

```

```

[26]: <IPython.lib.display.IFrame at 0x202278d7d68>

```

## Integration of Algorithms for Mixed-Attribute Data with Hyper-parameter Optimisation in Sklearn

This example shows how to integrate the GFMM classifiers for mixed-attribute with the Random Search Cross-Validation functionality implemented by scikit-learn

Note that this example uses the extended improved incremental learning algorithm and Random Search for illustration. However, other learning algorithms for mixed-attribute data in the library can be used similarly for any hyper-parameter tuning methods.

```

[1]: import os
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split
from hbbrain.mixed_data.eiol_gfmm import ExtendedImprovedOnlineGFMM

```



## Load dataset.

This example uses the `japanese_credit` dataset for illustration purposes. The continuous features in this dataset were normalised into the range of `[0, 1]`, while categorical features were kept unchanged. Note that the numerical features in training and testing datasets must be in the range of `[0, 1]` because the GFMM classifiers require features in the unit cube.

```
[2]: from pathlib import Path
      this_notebook_dir = os.path.dirname(os.path.abspath("__file__"))
      project_dir = Path(this_notebook_dir).parent.parent

[3]: training_data_file = os.path.join(project_dir, Path("dataset/japanese_credit_train.csv"))
      testing_data_file = os.path.join(project_dir, Path("dataset/japanese_credit_test.csv"))

[4]: df_train = pd.read_csv(training_data_file, header=None)
      df_test = pd.read_csv(testing_data_file, header=None)

      Xy_train = df_train.to_numpy()
      Xy_test = df_test.to_numpy()

      Xtr = Xy_train[:, :-1]
      ytr = Xy_train[:, -1].astype(int)

      Xtest = Xy_test[:, :-1]
      ytest = Xy_test[:, -1].astype(int)
```

## Using Random Search with 5-fold cross-validation

```
[5]: parameters = {'theta': np.arange(0.05, 1.01, 0.05), 'delta': np.arange(0.05, 1.01, 0.05),
      ↪ 'alpha': np.arange(0.1, 1.1, 0.1), 'gamma': [0.5, 1, 2, 4, 8, 16]}

[6]: # Using random search with only 20 random combinations of parameters
      eiol_gfmm_rd_search = ExtendedImprovedOnlineGFMM()
      clf_rd_search = RandomizedSearchCV(eiol_gfmm_rd_search, parameters, n_iter=20, cv=5,
      ↪ random_state=0)

[7]: # create parameters in the fit function apart from X and y
      # we use the expansion condition for categorical featurers using the average entropy
      ↪ changing values over all categorical features
      fit_params={'categorical_features':[0, 3, 4, 5, 6, 8, 9, 11, 12], 'type_cat_expansion':1}
      clf_rd_search.fit(Xtr, ytr, **fit_params)

[7]: RandomizedSearchCV(cv=5,
      estimator=ExtendedImprovedOnlineGFMM(C=array([], dtype=float64),
      D=array([], dtype=float64),
      N_samples=array([],
      ↪ dtype=float64),
      V=array([], dtype=float64),
      W=array([], dtype=float64)),
      n_iter=20,
      param_distributions={'alpha': array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.
```

(continues on next page)

(continued from previous page)

```

→7, 0.8, 0.9, 1. ]),
                                'delta': array([0.05, 0.1 , 0.15, 0.2 , 0.25, 0.
→3 , 0.35, 0.4 , 0.45, 0.5 , 0.55,
                                0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95, 1. ]),
                                'gamma': [0.5, 1, 2, 4, 8, 16],
                                'theta': array([0.05, 0.1 , 0.15, 0.2 , 0.25, 0.
→3 , 0.35, 0.4 , 0.45, 0.5 , 0.55,
                                0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95, 1. ])}},
                                random_state=0)

```

```

[8]: print("Best average score = ", clf_rd_search.best_score_)
     print("Best params: ", clf_rd_search.best_params_)

```

```

Best average score = 0.8209672184355729
Best params: {'theta': 0.5, 'gamma': 2, 'delta': 0.15000000000000002, 'alpha': 0.8}

```

```

[9]: best_gfmm_rd_search = clf_rd_search.best_estimator_

```

```

[10]: # Testing the performance on the test set
      y_pred_rd_search = best_gfmm_rd_search.predict(Xtest)

```

```

[11]: acc_rd_search = accuracy_score(ytest, y_pred_rd_search)
      print(f'Accuracy (random-search) = {acc_rd_search * 100: .2f}%')

```

```

Accuracy (random-search) = 79.39%

```

## Integration of Ensemble Models with Hyper-parameter Optimisation in Sklearn

This example shows how to integrate the random hyperboxes classifier with the Random Search Cross-Validation functionality implemented by scikit-learn.

Note that this example uses the random hyperboxes model and Random Search for illustration. However, other hyperbox-based ensemble learning algorithms in the library can be used similarly for any hyper-parameter tuning methods.

```

[1]: import warnings
     warnings.filterwarnings('ignore')
     import os
     import pandas as pd
     import numpy as np
     from sklearn.metrics import accuracy_score
     from sklearn.model_selection import RandomizedSearchCV
     from sklearn.model_selection import train_test_split
     from hbbbrain.numerical_data.ensemble_learner.random_hyperboxes import_
→RandomHyperboxesClassifier
     from hbbbrain.numerical_data.incremental_learner.onln_gfmm import OnlineGFMM

```

**Load dataset, normalize numerical features into the range of [0, 1] and build training and testing datasets.**

This example will use the breast cancer dataset in sklearn for illustration purposes.

```
[2]: from sklearn.datasets import load_breast_cancer
      from sklearn.preprocessing import MinMaxScaler

[3]: df = load_breast_cancer()
      X = df.data
      y = df.target

[4]: scaler = MinMaxScaler()
      X = scaler.fit_transform(X)

[5]: X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=0)
```

**Using Random Search with 5-fold cross-validation.**

```
[6]: parameters = {'n_estimators': [20, 30, 50, 100, 200, 500],
                   'max_samples': [0.2, 0.3, 0.4, 0.5, 0.6],
                   'max_features' : [0.2, 0.3, 0.4, 0.5, 0.6],
                   'class_balanced' : [True, False],
                   'feature_balanced' : [True, False],
                   'n_jobs' : [4],
                   'random_state' : [0],
                   'base_estimator__theta' : np.arange(0.05, 0.61, 0.05),
                   'base_estimator__gamma' : [0.5, 1, 2, 4, 8, 16]}

[7]: # Init base learner. This example uses the original online learning algorithm to train a
      ↪ GFMM classifier
      base_estimator = OnlineGFMM()

[8]: # Using random search with only 40 random combinations of parameters
      random_hyperboxes_clf = RandomHyperboxesClassifier(base_estimator=base_estimator)
      clf_rd_search = RandomizedSearchCV(random_hyperboxes_clf, parameters, n_iter=40, cv=5,
      ↪ random_state=0)

[9]: clf_rd_search.fit(X_train, y_train)

[9]: RandomizedSearchCV(cv=5,
                       estimator=RandomHyperboxesClassifier(base_
      ↪ estimator=OnlineGFMM(C=array([], dtype=float64),
      ↪ V=array([], dtype=float64),
      ↪ W=array([], dtype=float64))),
                       n_iter=40,
                       param_distributions={'base_estimator__gamma': [0.5, 1, 2, 4,
                                                                    8, 16],
```

(continues on next page)

(continued from previous page)

```

        'base_estimator__theta': array([0.05, 0.1 , 0.15,
→ 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 , 0.55,
        0.6 ]),
        'class_balanced': [True, False],
        'feature_balanced': [True, False],
        'max_features': [0.2, 0.3, 0.4, 0.5,
                        0.6],
        'max_samples': [0.2, 0.3, 0.4, 0.5,
                        0.6],
        'n_estimators': [20, 30, 50, 100, 200,
                        500],
        'n_jobs': [4], 'random_state': [0]},
        random_state=0)

```

```

[10]: print("Best average score = ", clf_rd_search.best_score_)
      print("Best params: ", clf_rd_search.best_params_)

Best average score = 0.9714285714285715
Best params: {'random_state': 0, 'n_jobs': 4, 'n_estimators': 500, 'max_samples': 0.6,
→ 'max_features': 0.5, 'feature_balanced': True, 'class_balanced': False, 'base_
→ estimator__theta': 0.15000000000000002, 'base_estimator__gamma': 16}

```

```

[12]: best_gfmm_rd_search = clf_rd_search.best_estimator_

```

```

[13]: # Testing the performance on the test set
      y_pred_rd_search = best_gfmm_rd_search.predict(X_test)

```

```

[14]: acc_rd_search = accuracy_score(y_test, y_pred_rd_search)
      print(f'Accuracy (random-search) = {acc_rd_search * 100 : .2f}%')

Accuracy (random-search) = 96.49%

```

## 2.8.8 Other learning abilities of GFMM models

### Learning from labelled and unlabelled data using GFMM

This example shows how to use various learning algorithms of the general fuzzy min-max (GFMM) classifier to learning from the datasets including both labelled and unlabelled samples.

#### Loading the labelled and unlabelled samples from an example dataset in the folder ‘dataset’

File ‘syn\_num\_train.csv’ contains all labelled samples for a training dataset. File ‘syn\_num\_train\_labeled\_unlabelled\_mix.csv’ contains both labelled and unlabelled samples which are created from the ‘syn\_num\_train.csv’ file by eliminating randomly the class labels of several samples. File ‘syn\_num\_test.csv’ contains 1000 testing samples all of which are labelled.

This example compares a trained GFMM model learning from fully labelled training data and a GFMM model trained on labelled and unlabelled data based on the resulting hyperboxes and their classification performance.

```

[1]: %matplotlib notebook

[2]: import os
import warnings
warnings.filterwarnings('ignore')

[3]: import pandas as pd

[4]: # Get the path to the this jupyter notebook file
this_notebook_dir = os.path.dirname(os.path.abspath("__file__"))
this_notebook_dir

[4]: 'C:\\hyperbox-brain\\examples\\other_learning_ability_gfmm'

[5]: # Get the home folder of the hyperbox-brain toolbox
from pathlib import Path
project_dir = Path(this_notebook_dir).parent.parent
project_dir

[5]: WindowsPath('C:/hyperbox-brain')

[6]: # Create the path to the training and testing files
labelled_training_file = os.path.join(project_dir, Path("dataset/syn_num_train.csv"))
labelled_unlabelled_training_file = os.path.join(project_dir, Path("dataset/syn_num_
↪train_labeled_unlabelled_mix.csv"))
testing_file = os.path.join(project_dir, Path("dataset/syn_num_test.csv"))

[7]: labelled_training_file

[7]: 'C:\\hyperbox-brain\\dataset\\syn_num_train.csv'

[8]: labelled_unlabelled_training_file

[8]: 'C:\\hyperbox-brain\\dataset\\syn_num_train_labeled_unlabelled_mix.csv'

[9]: testing_file

[9]: 'C:\\hyperbox-brain\\dataset\\syn_num_test.csv'

[10]: # Create training and testing data sets
df_labelled_train = pd.read_csv(labelled_training_file, header=None)
df_labelled_unlabelled_train = pd.read_csv(labelled_unlabelled_training_file, ↪
↪header=None)
df_test = pd.read_csv(testing_file, header=None)

Xy_labelled_train = df_labelled_train.to_numpy()
Xy_labelled_unlabelled_train = df_labelled_unlabelled_train.to_numpy()
Xy_test = df_test.to_numpy()

Xtr_labelled = Xy_labelled_train[:, :-1]
ytr_labelled = Xy_labelled_train[:, -1]

```

(continues on next page)

(continued from previous page)

```
Xtr_labelled_unlabelled = Xy_lablled_unlabelled_train[:, :-1]
ytr_labelled_unlabelled = Xy_lablled_unlabelled_train[:, -1]

Xtest = Xy_test[:, :-1]
ytest = Xy_test[:, -1]
```

```
[11]: import numpy as np
      from matplotlib import pyplot
```

```
[12]: class_label = ['Unlabelled class', '1', '2']
      n_missing = np.isnan(ytr_labelled_unlabelled).sum()
      n_class_1 = (ytr_labelled_unlabelled == 1).sum()
      n_class_2 = (ytr_labelled_unlabelled == 2).sum()
      class_distribution = [n_missing, n_class_1, n_class_2]
```

```
[13]: # plot the distribution
      pyplot.bar(class_label, class_distribution)
      pyplot.xlabel('Class labels')
      pyplot.ylabel('Number of samples')
      pyplot.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

### Showing samples corresponding to missing values and class labels

```
[14]: import matplotlib.pyplot as plt
      colours = ['red', 'blue', 'green']
      labels = ['Unlabelled class', 'class 1', 'class 2']
      classes = np.unique(ytr_labelled_unlabelled)
      fig1, ax1 = plt.subplots()
      for i in classes:
          if np.isnan(i) == True:
              data = Xtr_labelled_unlabelled[np.isnan(ytr_labelled_unlabelled)]
              color = colours[0]
              label = labels[0]
          else:
              data = Xtr_labelled_unlabelled[ytr_labelled_unlabelled == i]
              color = colours[int(i)]
              label = labels[int(i)]
          ax1.scatter(
              data[:, 0],
              data[:, 1],
              color=color,
              label=label
          )

      ax1.set_xlabel('X1')
      ax1.set_ylabel('X2')
```

(continues on next page)

(continued from previous page)

```
ax1.legend(loc='lower right')

plt.show()

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>
```

Next, we will build general fuzzy min-max neural networks using various learning algorithms from unlabelled and labelled data.

## 1. Original incremental learning algorithm for GFMM

```
[15]: from hbbrain.numerical_data.incremental_learner.onln_gfmm import OnlineGFMM
```

```
[16]: # Initializing parameters
theta = 0.2
theta_min = 0.2
gamma = 1
is_draw = False
```

### 1.1 Training a GFMM model using the original incremental learning algorithm on a fully labelled training set

```
[17]: onln_gfmm_clf_labelled = OnlineGFMM(theta=theta, theta_min=theta_min, gamma=gamma, is_
      ↪ draw=is_draw)
onln_gfmm_clf_labelled.fit(Xtr_labelled, ytr_labelled)

[17]: OnlineGFMM(C=array([1, 2, 1, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 1, 2, 2, 2, 1]),
      V=array([[0.334755 , 0.54567  ],
      [0.60358 , 0.413255 ],
      [0.71185 , 0.67467  ],
      [0.66395 , 0.48815495],
      [0.15331 , 0.03      ],
      [0.17884 , 0.26124  ],
      [0.50058 , 0.62858  ],
      [0.13903 , 0.44711  ],
      [0.03     , 0.28498  ],
      [0.56487 , 0.17003  ],
      [0.49255 , 0.38293  ],
      [0.39868 , 0.4497275 ],
      [0.30123 , 0.7592   ],
      [0.581545 , 0.54123  ],
      [0.25929 , 0.81558  ],
      [0.6...
      [0.83355 , 0.65933  ],
      [0.33484 , 0.22603  ],
      [0.37033 , 0.44704  ],
      [0.68491 , 0.7921   ],
      [0.3197  , 0.62174  ],
```

(continues on next page)

(continued from previous page)

```

[0.14737 , 0.48239 ],
[0.76254 , 0.34978 ],
[0.59077 , 0.4497175 ],
[0.581535 , 0.54842 ],
[0.47952 , 0.91371 ],
[0.66394 , 0.62857 ],
[0.25929 , 0.81558 ],
[0.815 , 0.413245 ],
[0.67906 , 0.85165 ],
[0.36745 , 0.6747 ],
[0.80583 , 0.43242 ],
[0.80969 , 0.2317 ],
[0.24341 , 0.2432 ],
[0.91185 , 0.48697 ]]),
    theta=0.2, theta_min=0.2)

```

### Display decision boundaries among classes if input data are 2-dimensional

```
[18]: onln_gfmm_clf_labelled.draw_hyperbox_and_boundary("The trained GFMM classifier and its_
↳ decision boundaries on a fully labelled dataset")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[19]: print("Number of hyperboxes = ", onln_gfmm_clf_labelled.get_n_hyperboxes())
```

```
Number of hyperboxes = 22
```

### Prediction

```
[20]: from sklearn.metrics import accuracy_score
```

```
[21]: # Using prediction returning only a predicted class for each input sample
y_pred_labelled = onln_gfmm_clf_labelled.predict(Xtest)
acc = accuracy_score(ytest, y_pred_labelled)
print(f'Accuracy (trained on a fully labelled dataset) = {acc * 100: .2f}%')
```

```
Accuracy (trained on a fully labelled dataset) = 86.10%
```

```
[22]: # Using prediction returning probability values for classes with respect to each input_
↳ sample
onln_gfmm_clf_labelled.predict_proba(Xtest[0:10])
```

```
[22]: array([[0.50692844, 0.49307156],
[0.5067865 , 0.4932135 ],
[0.48419044, 0.51580956],
[0.51756754, 0.48243246],
[0.49898253, 0.50101747],
[0.50386085, 0.49613915],
```

(continues on next page)



(continued from previous page)

```
[0.49633584, 0.50366416],
[0.44839455, 0.55160545],
[0.4878477 , 0.5121523 ],
[0.48650536, 0.51349464]])
```

```
[23]: # Using prediction returning membership values for classes with respect to each input,
      ↪ sample
      onln_gfmm_clf_labelled.predict_with_membership(Xtest[0:10])
```

```
[23]: array([[1.          , 0.972665   ],
          [1.          , 0.9732175  ],
          [0.9387       , 1.          ],
          [1.          , 0.932115   ],
          [0.986965     , 0.99099    ],
          [1.          , 0.98467495 ],
          [0.98545      , 1.          ],
          [0.81289      , 1.          ],
          [0.93918      , 0.98597    ],
          [0.94744      , 1.          ]])
```

## 1.2 Training a GFMM model using the original incremental learning algorithm on a labelled and unlabelled training set

```
[24]: onln_gfmm_clf_labelled_unlabelled = OnlineGFMM(theta=theta, theta_min=theta_min,
      ↪ gamma=gamma, is_draw=is_draw)
      onln_gfmm_clf_labelled_unlabelled.fit(Xtr_labelled_unlabelled, ytr_labelled_unlabelled)
```

```
[24]: OnlineGFMM(C=array([1, 2, 1, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 1, 2, 2, 2, 1]),
      V=array([[0.334755 , 0.54567   ],
          [0.60358   , 0.413255   ],
          [0.71185   , 0.67467    ],
          [0.66395   , 0.48815495 ],
          [0.15331   , 0.03        ],
          [0.17884   , 0.26124    ],
          [0.50058   , 0.62858    ],
          [0.13903   , 0.44711    ],
          [0.03      , 0.28498    ],
          [0.56487   , 0.17003    ],
          [0.49255   , 0.38293    ],
          [0.39868   , 0.439005   ],
          [0.30123   , 0.7592     ],
          [0.57962   , 0.56029    ],
          [0.25929   , 0.81558    ],
          [0.665...
          [0.83355   , 0.65933    ],
          [0.33484   , 0.22603    ],
          [0.37033   , 0.44704    ],
          [0.68491   , 0.7921     ],
          [0.3197    , 0.62174    ],
          [0.14737   , 0.48239    ],
          [0.76254   , 0.34978    ],
```

(continues on next page)

(continued from previous page)

```
[0.5843      , 0.438995  ],
[0.59235     , 0.54842   ],
[0.47952     , 0.91371   ],
[0.66394     , 0.62857   ],
[0.25929     , 0.81558   ],
[0.815       , 0.413245  ],
[0.67906     , 0.85165   ],
[0.36745     , 0.6747    ],
[0.80583     , 0.43242   ],
[0.80969     , 0.2317    ],
[0.24341     , 0.2432    ],
[0.91185     , 0.48697   ]]),
    theta=0.2, theta_min=0.2)
```

### Display decision boundaries among classes if input data are 2-dimensional

```
[25]: onln_gfmm_clf_labelled_unlabelled.draw_hyperbox_and_boundary("The trained GFMM_
↪ classifier and its decision boundaries on a labelled and unlabelled dataset")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[26]: print("Number of hyperboxes = ", onln_gfmm_clf_labelled_unlabelled.get_n_hyperboxes())
```

```
Number of hyperboxes = 22
```

We can see that in this case all unlabelled samples were absorbed and assigned to suitable class labels to form hyperboxes. All unlabelled samples located in the areas dominated by samples belonging to only one class were classified correctly to the same class. Only samples in the overlapping areas between two classes resulted in little difference in the trained GFMM models between the model trained on a fully labelled dataset and the model trained on an unlabelled and labelled dataset.

### Prediction

```
[27]: # Using prediction returning only a predicted class for each input sample
y_pred_labelled_unlabelled = onln_gfmm_clf_labelled_unlabelled.predict(Xtest)
acc = accuracy_score(ytest, y_pred_labelled_unlabelled)
print(f'Accuracy (trained on a mixed labelled and unlabelled dataset) = {acc * 100: .2f}%
↪')
```

```
Accuracy (trained on a mixed labelled and unlabelled dataset) = 86.00%
```

```
[28]: # Using prediction returning probability values for classes with respect to each input_
↪ sample
onln_gfmm_clf_labelled_unlabelled.predict_proba(Xtest[0:10])
```

```
[28]: array([[0.50692844, 0.49307156],
            [0.50955544, 0.49044456],
            [0.48419044, 0.51580956],
            [0.51705239, 0.48294761],
```

(continues on next page)

(continued from previous page)

```
[0.50170709, 0.49829291],
[0.50386085, 0.49613915],
[0.49633584, 0.50366416],
[0.44839455, 0.55160545],
[0.49076276, 0.50923724],
[0.48650536, 0.51349464]]])
```

```
[29]: # Using prediction returning membership values for classes with respect to each input,
↪ sample
onln_gfmm_clf_labelled_unlabelled.predict_with_membership(Xtest[0:10])
```

```
[29]: array([[1.          , 0.972665   ],
[1.          , 0.962495   ],
[0.9387       , 1.          ],
[1.          , 0.93404    ],
[0.99778      , 0.99099    ],
[1.          , 0.98467495],
[0.98545      , 1.          ],
[0.81289      , 1.          ],
[0.943965     , 0.9795     ],
[0.94744      , 1.          ]])
```

## 2. Improved online learning algorithm for GFMM (IOL-GFMM)

```
[30]: from hbbrain.numerical_data.incremental_learner.iol_gfmm import ImprovedOnlineGFMM
```

### 2.1 Training a GFMM model using the improved online learning algorithm on a fully labelled training set

```
[31]: iol_gfmm_clf_labelled = ImprovedOnlineGFMM(theta=theta, gamma=gamma, is_draw=is_draw)
iol_gfmm_clf_labelled.fit(Xtr_labelled, ytr_labelled)

[31]: ImprovedOnlineGFMM(C=array([1, 2, 1, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 1, 2, 1, 1, 2, 2, 1,
↪ 1, 1,
2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2,
1, 1, 2, 2, 2, 1]),
N_samples=array([29, 3, 17, 13, 12, 29, 11, 8, 2, 9, 27, 1, 10,
↪ 1, 1, 4, 13,
1, 3, 3, 13, 3, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1,
1, 1, 1, 5, 2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1]),
V=array([[0.33593, 0.55942],
[0.77074, 0.48234],
[0.71185, 0.67467],
[0.64586, 0.41654],
[0.15331, 0.
[0.65327, 0.51585],
[0.57962, 0.57837],
[0.66562, 0.36352],
[0.68408, 0.43479],
```

(continues on next page)

(continued from previous page)

```

[0.70743, 0.50325],
[0.70647, 0.65933],
[0.28822, 0.62174],
[0.24341, 0.2432 ],
[0.34044, 0.55512],
[0.75421, 0.41466],
[0.55819, 0.39125],
[0.59655, 0.56029],
[0.71946, 0.45413],
[0.36745, 0.52006],
[0.91185, 0.48697],
[0.6504 , 0.51624],
[0.56487, 0.17003],
[0.59235, 0.54123],
[0.55763, 0.43813]]),
    theta=0.2)

```

### Display decision boundaries among classes if input data are 2-dimensional

```
[32]: iol_gfmm_clf_labelled.draw_hyperbox_and_boundary("The trained GFMM classifier and its_
↳ decision boundaries on a fully labelled dataset")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[33]: print("Number of hyperboxes = ", iol_gfmm_clf_labelled.get_n_hyperboxes())
```

```
Number of hyperboxes = 50
```

### Prediction

```
[34]: # Using prediction returning only a predicted class for each input sample
y_pred_labelled_iol = iol_gfmm_clf_labelled.predict(Xtest)
acc_iol = accuracy_score(ytest, y_pred_labelled_iol)
print(f'Accuracy (trained on a fully labelled dataset) = {acc_iol * 100: .2f}%')
```

```
Accuracy (trained on a fully labelled dataset) = 87.90%
```

```
[35]: # Using prediction returning probability values for classes with respect to each input_
↳ sample
iol_gfmm_clf_labelled.predict_proba(Xtest[0:10])
```

```
[35]: array([[0.49576188, 0.50423812],
[0.48655857, 0.51344143],
[0.48198338, 0.51801662],
[0.50614609, 0.49385391],
[0.49059122, 0.50940878],
[0.50281577, 0.49718423],
[0.49979902, 0.50020098],
```

(continues on next page)

(continued from previous page)

```
[0.44839455, 0.55160545],
[0.49441327, 0.50558673],
[0.48773256, 0.51226744]])
```

```
[36]: # Using prediction returning membership values for classes with respect to each input,
      ↪ sample
      iol_gfmm_clf_labelled.predict_with_membership(Xtest[0:10])
```

```
[36]: array([[0.98319, 1.      ],
            [0.94134, 0.99335],
            [0.93044, 1.      ],
            [0.98576, 0.96182],
            [0.96306, 1.      ],
            [1.      , 0.9888 ],
            [0.98227, 0.98306],
            [0.81289, 1.      ],
            [0.9779 , 1.      ],
            [0.94744, 0.9951 ]])
```

## 2.2 Training a GFMM model using the improved online learning algorithm on a labelled and unlabelled training set

```
[37]: iol_gfmm_clf_labelled_unlabelled = ImprovedOnlineGFMM(theta=theta, gamma=gamma, is_
      ↪ draw=is_draw)
      iol_gfmm_clf_labelled_unlabelled.fit(Xtr_labelled_unlabelled, ytr_labelled_unlabelled)
```

```
[37]: ImprovedOnlineGFMM(C=array([1, 2, 1, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 1, 2, 1, 1, 2, 1,
      ↪ 2, 1,
            1, 1, 2, 2, 2, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2,
            2, 2, 2, 1, 2, 2, 1])),
      N_samples=array([22,  3, 15, 15,  7,  3, 22, 10, 11,  1, 10, 23,  1,
      ↪ 13,  1,  1,  4,
            7,  7,  3,  3,  3,  9,  5,  2,  1,  1,  1,  1,  1,  1,  1,  2,  1,
            2,  1,  1,  1,  1,  1,  1,  3,  1,  2,  2,  1,  1,  1,  1,  1,  1]),
      V=array([[0.33593, 0.55942],
            [0.77074, 0.48234],
            [0.71185, 0.67467],
            [0.64586, 0.41654],
            [0.1533...,
            [0.76369, 0.48412],
            [0.80583, 0.43242],
            [0.80969, 0.31878],
            [0.65327, 0.51585],
            [0.57962, 0.57837],
            [0.66562, 0.36352],
            [0.68408, 0.43479],
            [0.70743, 0.50325],
            [0.70647, 0.65933],
            [0.28822, 0.62174],
            [0.34044, 0.55512],
            [0.75421, 0.41466],
```

(continues on next page)

(continued from previous page)

```
[0.55819, 0.39125],
[0.59655, 0.56029],
[0.71946, 0.45413],
[0.91185, 0.48697],
[0.6504 , 0.51624],
[0.56487, 0.17003],
[0.55763, 0.43813]]),
    theta=0.2)
```

## Display decision boundaries among classes if input data are 2-dimensional

```
[38]: iol_gfmm_clf_labelled_unlabelled.draw_hyperbox_and_boundary("The trained GFMM classifier,
↳ and its decision boundaries on a labelled and unlabelled dataset")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

We can see that the IOL-GFMM algorithm can also learn from a labelled and unlabelled dataset. Hyperboxes which have been formed from unlabelled samples will be assigned a suitable class when they cover new labelled samples. If not, these hyperboxes will create unlabelled clusters. In this case, all unlabelled hyperboxes were assigned corresponding labels. The classification performance of the model trained on both labelled and unlabelled samples is slightly less than that of the model trained on a fully labelled dataset.

```
[39]: print("Number of hyperboxes = ", iol_gfmm_clf_labelled_unlabelled.get_n_hyperboxes())
```

```
Number of hyperboxes = 51
```

## Prediction

```
[40]: # Using prediction returning only a predicted class for each input sample
y_pred_labelled_unlabelled_iol = iol_gfmm_clf_labelled_unlabelled.predict(Xtest)
acc_iol_labelled_unlabelled = accuracy_score(ytest, y_pred_labelled_unlabelled_iol)
print(f'Accuracy (trained on a mixed labelled and unlabelled dataset) = {acc_iol_
↳ labelled_unlabelled * 100: .2f}%')
```

```
Accuracy (trained on a mixed labelled and unlabelled dataset) = 87.50%
```

```
[41]: # Using prediction returning membership values for classes with respect to each input_
↳ sample
iol_gfmm_clf_labelled_unlabelled.predict_with_membership(Xtest[0:10])
```

```
[41]: array([[0.98319, 1.      ],
[0.98204, 0.99335],
[0.93044, 1.      ],
[0.97525, 1.      ],
[0.96306, 1.      ],
[1.      , 0.9888 ],
[0.98227, 0.98306],
[0.81937, 1.      ],
```

(continues on next page)

(continued from previous page)

```
[0.9779 , 1.      ],
[0.94744, 1.      ]])
```

```
[42]: # Using prediction returning probability values for classes with respect to each input.
      ↪sample
      iol_gfmm_clf_labelled_unlabelled.predict_proba(Xtest[0:10])
```

```
[42]: array([[0.49576188, 0.50423812],
          [0.49713727, 0.50286273],
          [0.48198338, 0.51801662],
          [0.49373497, 0.50626503],
          [0.49059122, 0.50940878],
          [0.50281577, 0.49718423],
          [0.49979902, 0.50020098],
          [0.45035919, 0.54964081],
          [0.49441327, 0.50558673],
          [0.48650536, 0.51349464]])
```

### 3. Accelerated agglomerative learning algorithm for GFMM (AGGLO-2)

```
[43]: from hbbrain.numerical_data.batch_learner.accel_agglo_gfmm import
      ↪AccelAgglomerativeLearningGFMM
```

#### 3.1 Training a GFMM model using the accelerated agglomerative learning algorithm on a fully labelled training set

```
[44]: # Initialise parameters
      theta = 0.2
      gamma = 1
      min_simil = 0
      simil_measure = 'long'
      is_draw = False
```

```
[45]: aggro2_clf_labelled = AccelAgglomerativeLearningGFMM(theta=theta, gamma=gamma, min_
      ↪simil=min_simil, simil_measure=simil_measure, is_draw=is_draw)
      aggro2_clf_labelled.fit(Xtr_labelled, ytr_labelled)
```

```
[45]: AccelAgglomerativeLearningGFMM(min_simil=0, simil_measure='long', theta=0.2)
```

#### Display decision boundaries among classes if input data are 2-dimensional

```
[46]: aggro2_clf_labelled.draw_hyperbox_and_boundary("The trained GFMM classifier and its
      ↪decision boundaries on a fully labelled dataset")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[47]: print("Number of hyperboxes = ", agгло2_clf_labelled.get_n_hyperboxes())
```

```
Number of hyperboxes = 43
```

## Prediction

```
[48]: # Using prediction returning only a predicted class for each input sample
y_pred_labelled_agglo2 = agгло2_clf_labelled.predict(Xtest)
acc_agglo2_labelled = accuracy_score(ytest, y_pred_labelled_agglo2)
print(f'Accuracy (trained on a fully labelled dataset) = {acc_agglo2_labelled * 100: .2f}
↪ %')
```

```
Accuracy (trained on a fully labelled dataset) = 87.30%
```

```
[49]: # Using prediction returning membership values for classes with respect to each input
↪ sample
aggло2_clf_labelled.predict_with_membership(Xtest[0:10])
```

```
[49]: array([[0.98924, 0.98094],
          [0.98112, 0.96336],
          [0.93044, 0.99767],
          [0.98576, 0.93404],
          [0.96956, 1.      ],
          [0.9989 , 0.9888 ],
          [0.98227, 0.98306],
          [0.81289, 1.      ],
          [0.9779 , 1.      ],
          [0.94744, 0.9951 ]])
```

```
[50]: # Using prediction returning probability values for classes with respect to each input
↪ sample
aggло2_clf_labelled.predict_proba(Xtest[0:10])
```

```
[50]: array([[0.50210641, 0.49789359],
          [0.50456677, 0.49543323],
          [0.48256583, 0.51743417],
          [0.51347015, 0.48652985],
          [0.49227239, 0.50772761],
          [0.50254062, 0.49745938],
          [0.49979902, 0.50020098],
          [0.44839455, 0.55160545],
          [0.49441327, 0.50558673],
          [0.48773256, 0.51226744]])
```



### 3.2 Training a GFMM model using the accelerated agglomerative learning algorithm on a mixed labelled and unlabelled training set

```
[51]: aggro2_clf_labelled_unlabelled = AccelAgglomerativeLearningGFMM(theta=theta, gamma=gamma,
    ↪ min_simil=min_simil, simil_measure=simil_measure, is_draw=is_draw)
    aggro2_clf_labelled_unlabelled.fit(Xtr_labelled_unlabelled, ytr_labelled_unlabelled)

[51]: AccelAgglomerativeLearningGFMM(min_simil=0, simil_measure='long', theta=0.2)

[52]: aggro2_clf_labelled_unlabelled.draw_hyperbox_and_boundary("The trained GFMM classifier,
    ↪ and its decision boundaries on a mixed labelled and unlabelled dataset")

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

[53]: print("Number of hyperboxes = ", aggro2_clf_labelled_unlabelled.get_n_hyperboxes())

Number of hyperboxes = 42
```

Similarly to the original online learning algorithm and improved online learning algorithm, the accelerated agglomerative learning algorithm is also capable of learning from a mixed labelled and unlabelled dataset. Unlabelled samples located in the regions surrounded by many samples belonging to only one class were correctly assigned the same class label. The difference between two models trained on the fully labelled dataset and the mixed labelled and unlabelled dataset happens in the unlabelled samples within the overlapping regions between two classes.

#### Prediction

```
[54]: # Using prediction returning only a predicted class for each input sample
    y_pred_labelled_unlabelled_agglo2 = aggro2_clf_labelled_unlabelled.predict(Xtest)
    acc_agglo2_labelled_unlabelled = accuracy_score(ytest, y_pred_labelled_unlabelled_agglo2)
    print(f'Accuracy (trained on a mixed labelled and unlabelled dataset) = {acc_agglo2_
    ↪ labelled_unlabelled * 100: .2f}%')

Accuracy (trained on a mixed labelled and unlabelled dataset) = 87.00%

[55]: # Using prediction returning membership values for classes with respect to each input,
    ↪ sample
    aggro2_clf_labelled_unlabelled.predict_with_membership(Xtest[0:10])

[55]: array([[0.98924, 0.98094],
           [0.98112, 0.96336],
           [0.93044, 0.99767],
           [0.98576, 0.93404],
           [0.96956, 1.      ],
           [0.99933, 0.9888 ],
           [0.97989, 0.98306],
           [0.81289, 1.      ],
           [0.9779 , 1.      ],
           [0.94744, 0.9951 ]])

[56]: # Using prediction returning probability values for classes with respect to each input,
    ↪ sample
    aggro2_clf_labelled_unlabelled.predict_proba(Xtest[0:10])
```

```
[56]: array([[0.50210641, 0.49789359],
          [0.50456677, 0.49543323],
          [0.48256583, 0.51743417],
          [0.51347015, 0.48652985],
          [0.49227239, 0.50772761],
          [0.50264822, 0.49735178],
          [0.49919254, 0.50080746],
          [0.44839455, 0.55160545],
          [0.49441327, 0.50558673],
          [0.48773256, 0.51226744]])
```

#### 4. Agglomerative learning algorithm with a full similarity matrix for GFMM (AGGLO-SM)

```
[57]: from hbbrain.numerical_data.batch_learner.agglo_gfmm import AgglomerativeLearningGFMM
```

##### 4.1 Training a GFMM model using the agglomerative learning algorithm with a full similarity matrix on a fully labelled training set

```
[58]: aggro_sm_clf_labelled = AgglomerativeLearningGFMM(theta=theta, gamma=gamma, min_
      ↪simil=min_simil, simil_measure=simil_measure, is_draw=is_draw)
      aggro_sm_clf_labelled.fit(Xtr_labelled, ytr_labelled)
```

```
[58]: AgglomerativeLearningGFMM(min_simil=0, simil_measure='long', theta=0.2)
```

##### Display decision boundaries among classes if input data are 2-dimensional

```
[59]: aggro_sm_clf_labelled.draw_hyperbox_and_boundary("The trained GFMM classifier and its_
      ↪decision boundaries on a fully labelled dataset")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[60]: print("Number of hyperboxes = ", aggro_sm_clf_labelled.get_n_hyperboxes())
```

```
Number of hyperboxes = 45
```

##### Prediction

```
[61]: # Using prediction returning only a predicted class for each input sample
      y_pred_labelled_agglosm = aggro_sm_clf_labelled.predict(Xtest)
      acc_agglosm_labelled = accuracy_score(ytest, y_pred_labelled_agglosm)
      print(f'Accuracy (trained on a fully labelled dataset) = {acc_agglosm_labelled * 100: .
      ↪2f}%')
```

```
Accuracy (trained on a fully labelled dataset) = 86.20%
```

```
[62]: # Using prediction returning membership values for classes with respect to each input,
      ↪sample
      agгло_sm_clf_labelled.predict_with_membership(Xtest[0:10])
```

```
[62]: array([[0.98924, 0.98094],
           [0.94134, 0.96336],
           [0.93044, 1.      ],
           [0.98576, 0.93404],
           [0.96956, 1.      ],
           [0.99933, 0.9888 ],
           [0.98227, 0.98306],
           [0.81937, 1.      ],
           [0.9779 , 1.      ],
           [0.94744, 1.      ]])
```

```
[63]: # Using prediction returning probability values for classes with respect to each input,
      ↪sample
      agгло_sm_clf_labelled.predict_proba(Xtest[0:10])
```

```
[63]: array([[0.50210641, 0.49789359],
           [0.49421956, 0.50578044],
           [0.48198338, 0.51801662],
           [0.51347015, 0.48652985],
           [0.49227239, 0.50772761],
           [0.50264822, 0.49735178],
           [0.49979902, 0.50020098],
           [0.45035919, 0.54964081],
           [0.49441327, 0.50558673],
           [0.48650536, 0.51349464]])
```

## 4.2 Training a GFMM model using the agglomerative learning algorithm with a full similarity matrix on a mixed labelled and unlabelled training set

```
[64]: agгло_sm_clf_labelled_unlabelled = AgglomerativeLearningGFMM(theta=theta, gamma=gamma,
      ↪min_simil=min_simil, simil_measure=simil_measure, is_draw=is_draw)
      agгло_sm_clf_labelled_unlabelled.fit(Xtr_labelled_unlabelled, ytr_labelled_unlabelled)
```

```
[64]: AgglomerativeLearningGFMM(min_simil=0, simil_measure='long', theta=0.2)
```

### Display decision boundaries among classes if input data are 2-dimensional

```
[65]: agгло_sm_clf_labelled_unlabelled.draw_hyperbox_and_boundary("The trained GFMM classifier,
      ↪and its decision boundaries on a mixed labelled and unlabelled dataset")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[67]: print("Number of hyperboxes = ", agгло_sm_clf_labelled_unlabelled.get_n_hyperboxes())
```

```
Number of hyperboxes = 45
```

We can see that the agglomerative learning algorithm can also learn from a mixed labelled and unlabelled training set. Similarly to above learning algorithms, unlabelled samples surrounded by many samples belonging to only one class were assigned to the same class as their neighbours. The difference between the model trained by the AGGLO-SM algorithm on a fully labelled data set and a mixed labelled and unlabelled data set occurs in the regions that unlabelled samples are surrounded by overlapping labelled samples.

## Prediction

```
[68]: # Using prediction returning only a predicted class for each input sample
y_pred_labelled_unlabelled_agglosm = agglo_sm_clf_labelled_unlabelled.predict(Xtest)
acc_agglosm_labelled_unlabelled = accuracy_score(ytest, y_pred_labelled_unlabelled_
↪agglosm)
print(f'Accuracy (trained on a mixed labelled and unlabelled dataset) = {acc_agglosm_
↪labelled_unlabelled * 100: .2f}%')
```

```
Accuracy (trained on a mixed labelled and unlabelled dataset) = 86.40%
```

```
[69]: # Using prediction returning membership values for classes with respect to each input_
↪sample
agglo_sm_clf_labelled_unlabelled.predict_with_membership(Xtest[0:10])
```

```
[69]: array([[0.98924, 0.98094],
          [0.94134, 0.96336],
          [0.93044, 1.      ],
          [0.98576, 0.93404],
          [0.96956, 1.      ],
          [0.99933, 0.9888 ],
          [0.98227, 0.98306],
          [0.81937, 1.      ],
          [0.9779 , 1.      ],
          [0.94744, 1.      ]])
```

```
[70]: # Using prediction returning probability values for classes with respect to each input_
↪sample
agglo_sm_clf_labelled_unlabelled.predict_proba(Xtest[0:10])
```

```
[70]: array([[0.50210641, 0.49789359],
          [0.49421956, 0.50578044],
          [0.48198338, 0.51801662],
          [0.51347015, 0.48652985],
          [0.49227239, 0.50772761],
          [0.50264822, 0.49735178],
          [0.49979902, 0.50020098],
          [0.45035919, 0.54964081],
          [0.49441327, 0.50558673],
          [0.48650536, 0.51349464]])
```

## Continuous learning from a trained hyperbox-based model

This example shows how to perform a continuous learning ability for a trained hyperbox-based model.

```
[1]: %matplotlib notebook
```

```
[2]: import os
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
```

## Load training and testing datasets

In this example, we create two training datasets. One data set is used to trained a model and store it to the data storage. The other data set is used to continuously train the stored model after reloading that trained model.

```
[3]: from sklearn.model_selection import StratifiedKFold
```

```
[4]: # Get the path to the this jupyter notebook file
this_notebook_dir = os.path.dirname(os.path.abspath("__file__"))
this_notebook_dir
```

```
[4]: 'C:\\hyperbox-brain\\examples\\other_learning_ability_gfmm'
```

```
[5]: # Get the home folder of the hyperbox-brain toolbox
from pathlib import Path
project_dir = Path(this_notebook_dir).parent.parent
project_dir
```

```
[5]: WindowsPath('C:/hyperbox-brain')
```

```
[6]: # Create the path to the training and testing files
training_file = os.path.join(project_dir, Path("dataset/syn_num_train.csv"))
testing_file = os.path.join(project_dir, Path("dataset/syn_num_test.csv"))
```

```
[7]: # Create training and testing data sets
df_train = pd.read_csv(training_file, header=None)
df_test = pd.read_csv(testing_file, header=None)
```

```
Xy_train = df_train.to_numpy()
Xy_test = df_test.to_numpy()
```

```
Xtr = Xy_train[:, :-1]
ytr = Xy_train[:, -1]
```

```
Xtest = Xy_test[:, :-1]
ytest = Xy_test[:, -1]
```

```
[8]: # Split the existing training set into two training sets
skf = StratifiedKFold(n_splits=2)
```

(continues on next page)

(continued from previous page)

```

for train1_index, train2_index in skf.split(Xtr, ytr):
    X_tr1, X_tr2 = Xtr[train1_index], Xtr[train2_index]
    y_tr1, y_tr2 = ytr[train1_index], ytr[train2_index]

```

This example shows how to continue to train a deployed model. This example will use an incremental learning algorithm and an agglomerative learning algorithm without retraining from scratch for demonstration.

## 1. Training a General fuzzy min-max neural network model using an incremental learning algorithm

```
[9]: from hbbrain.numerical_data.incremental_learner.onln_gfmm import OnlineGFMM
```

```

[10]: # Initializing parameters
theta = 0.1
theta_min = 0.1
gamma = 1
is_draw = False

```

```

[11]: onln_gfmm_clf = OnlineGFMM(theta=theta, theta_min=theta_min, gamma=gamma, is_draw=is_
    ↪draw)
onln_gfmm_clf.fit(X_tr1, y_tr1)

```

```

[11]: OnlineGFMM(C=array([[1, 2, 1, 1, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 2, 1,
    1, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 2, 2, 1, 1, 2, 2]]),
    V=array([[0.42413    , 0.53516    ],
    [0.70577    , 0.39146    ],
    [0.82785    , 0.78025    ],
    [0.6416     , 0.4824875   ],
    [0.48794    , 0.672     ],
    [0.26651    , 0.18424    ],
    [0.32527    , 0.60393    ],
    [0.19944    , 0.03      ],
    [0.29343    , 0.28975    ],
    [0.63683    , 0.6936     ],
    [0.32782    , 0.55997    ],
    [0.03       , 0.47757    ],
    [0.54181    , 0.43986    ],
    [0.73368    , 0.2170... ],
    [0.49255    , 0.43164    ],
    [0.81877    , 0.51294    ],
    [0.4713     , 0.77996    ],
    [0.25281    , 0.41059    ],
    [0.25858    , 0.30637    ],
    [0.3378     , 0.451095   ],
    [0.68354    , 0.41653    ],
    [0.34544    , 0.85954    ],
    [0.59061    , 0.62471    ],
    [0.68628    , 0.65662    ],
    [0.14324    , 0.47785    ],
    [0.41517    , 0.51424    ],
    [0.83355    , 0.5761     ],
    [0.15331    , 0.13567    ],

```

(continues on next page)

(continued from previous page)

```
[0.25929 , 0.81558 ],
[0.815   , 0.35526 ],
[0.67625 , 0.80457 ],
[0.37033 , 0.26124 ],
[0.67914 , 0.48785 ]]),
    theta=0.1, theta_min=0.1)
```

### Display decision boundaries among classes if input data are 2-dimensional

```
[12]: onln_gfmm_clf.draw_hyperbox_and_boundary("The trained GFMM classifier and its decision_
      ↪ boundaries")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[13]: print("Number of hyperboxes = ", onln_gfmm_clf.get_n_hyperboxes())
```

```
Number of hyperboxes = 39
```

### Make prediction

```
[14]: from sklearn.metrics import accuracy_score
```

```
[15]: y_pred = onln_gfmm_clf.predict(Xtest)
      acc = accuracy_score(ytest, y_pred)
      print(f'Accuracy = {acc * 100: .2f}%')
```

```
Accuracy = 86.40%
```

### Store the trained model

```
[16]: from hbbrain.utils.model_storage import store_model
```

```
[17]: store_model(onln_gfmm_clf, "store_cont_model.dummy")
```

### Reload the trained model and make prediction

```
[18]: from hbbrain.utils.model_storage import load_model
```

```
[19]: trained_onln_gfmm_clf = load_model("store_cont_model.dummy")
```

```
[20]: y_pred_trained_model = trained_onln_gfmm_clf.predict(Xtest)
      acc = accuracy_score(ytest, y_pred_trained_model)
      print(f'Accuracy = {acc * 100: .2f}%')
```

```
Accuracy = 86.40%
```

### Continue to train the deployed model using another training set

```
[21]: trained_onln_gfmm_clf.fit(X_tr2, y_tr2)
[21]: OnlineGFMM(C=array([1, 2, 1, 1, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 2, 1,
    1, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 2, 2, 1, 1, 2, 2, 1, 1, 2, 1, 1,
    2, 2, 1, 2, 2, 2, 2, 2, 1])),
    V=array([[0.42413    , 0.53516    ],
    [0.70577    , 0.397105   ],
    [0.82785    , 0.78025    ],
    [0.66038    , 0.51128    ],
    [0.48794    , 0.672      ],
    [0.26651    , 0.18424    ],
    [0.32289    , 0.60194    ],
    [0.19944    , 0.03       ],
    [0.29343    , 0.28975    ],
    [0.63683    , 0.6936     ],
    [0.32906    , 0.55512    ],
    [0.03       , 0.47757    ],
    [0.54...
    [0.25929    , 0.81558    ],
    [0.815      , 0.397095   ],
    [0.67906    , 0.83605    ],
    [0.37033    , 0.26124    ],
    [0.66037    , 0.57837    ],
    [0.52197    , 0.91371    ],
    [0.52621    , 0.66846    ],
    [0.80583    , 0.43242    ],
    [0.79935    , 0.7757     ],
    [0.35813    , 0.58772    ],
    [0.79516    , 0.32629    ],
    [0.70743    , 0.50325    ],
    [0.36057    , 0.71561    ],
    [0.72496    , 0.38674    ],
    [0.28822    , 0.62174    ],
    [0.14737    , 0.28498    ],
    [0.56487    , 0.17003    ],
    [0.68469    , 0.2221     ],
    [0.55763    , 0.43813    ]]),
    theta=0.1, theta_min=0.1)
```



**Display decision boundaries among classes if input data are 2-dimensional**

```
[22]: trained_onln_gfmm_clf.draw_hyperbox_and_boundary("The trained GFMM classifier and its_
↪decision boundaries")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[23]: print("Number of hyperboxes = ", trained_onln_gfmm_clf.get_n_hyperboxes())
```

```
Number of hyperboxes = 53
```

**Make prediction**

```
[24]: y_pred = trained_onln_gfmm_clf.predict(Xtest)
acc = accuracy_score(ytest, y_pred)
print(f'Accuracy = {acc * 100: .2f}%')
```

```
Accuracy = 84.50%
```

**2. Training a General fuzzy min-max neural network model using an agglomerative learning algorithm**

```
[26]: from hbbrain.numerical_data.batch_learner.agglo_gfmm import AgglomerativeLearningGFMM
```

```
[27]: # Initialise parameters
theta = 0.2
gamma = 1
min_simil = 0
simil_measure = 'long'
is_draw = False
```

```
[28]: agglo_sm_clf = AgglomerativeLearningGFMM(theta=theta, gamma=gamma, min_simil=min_simil,
↪simil_measure=simil_measure, is_draw=is_draw)
agglo_sm_clf.fit(X_tr1, y_tr1)
```

```
[28]: AgglomerativeLearningGFMM(min_simil=0, simil_measure='long', theta=0.2)
```

**Display decision boundaries among classes if input data are 2-dimensional**

```
[29]: agglo_sm_clf.draw_hyperbox_and_boundary("The trained GFMM classifier and its decision_
↪boundaries")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[30]: print("Number of hyperboxes = ", agglo_sm_clf.get_n_hyperboxes())
```

```
Number of hyperboxes = 30
```

### Make prediction

```
[31]: y_pred_agglosm = agglo_sm_clf.predict(Xtest)
      acc = accuracy_score(ytest, y_pred_agglosm)
      print(f'Accuracy = {acc * 100: .2f}%')
```

```
Accuracy = 84.90%
```

### Store the trained model

```
[32]: store_model(agglo_sm_clf, "store_agglosm_model.dummy")
```

### Reload the trained model and make prediction

```
[33]: trained_agglo_sm_clf = load_model("store_agglosm_model.dummy")
```

```
[34]: y_pred_trained_agglosm_model = trained_agglo_sm_clf.predict(Xtest)
      acc = accuracy_score(ytest, y_pred_trained_agglosm_model)
      print(f'Accuracy = {acc * 100: .2f}%')
```

```
Accuracy = 84.90%
```

### Continue to train the deployed model using another training set

```
[35]: trained_agglo_sm_clf.fit(X_tr2, y_tr2)
```

```
[35]: AgglomerativeLearningGFMM(min_simil=0, simil_measure='long', theta=0.2)
```

### Display decision boundaries among classes if input data are 2-dimensional

```
[36]: trained_agglo_sm_clf.draw_hyperbox_and_boundary("The trained GFMM classifier and its_
      ↪decision boundaries")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[37]: print("Number of hyperboxes = ", trained_agglo_sm_clf.get_n_hyperboxes())
```

```
Number of hyperboxes = 51
```

## Make prediction

```
[38]: y_pred_agglosm = trained_agglo_sm_clf.predict(Xtest)
      acc = accuracy_score(ytest, y_pred_agglosm)
      print(f'Accuracy = {acc * 100: .2f}%')
```

```
Accuracy = 87.90%
```

## Learning from data sets with missing values of features

This tutorial illustrates the ability of learning from missing feature values of general fuzzy min-max neural networks using various training algorithms. This tutorial also shows how to use membership values to make prediction for testing data with missing feature values.

```
[1]: %matplotlib notebook
```

```
[2]: import os
      import warnings
      warnings.filterwarnings('ignore')
      import pandas as pd
      import numpy as np
      from sklearn.metrics import accuracy_score
```

## Load training and testing datasets

In this tutorial, we will load a training set without missing feature values, a training set with missing feature values, and a testing set without missing feature values. Next, we will compare the classification performance on the same testing set of the model trained on data with missing feature values and the one trained on data without missing feature values.

```
[3]: # Get the path to the this jupyter notebook file
      this_notebook_dir = os.path.dirname(os.path.abspath("__file__"))
      this_notebook_dir
```

```
[3]: 'C:\\hyperbox-brain\\examples\\other_learning_ability_gfmm'
```

```
[4]: # Get the home folder of the hyperbox-brain toolbox
      from pathlib import Path
      project_dir = Path(this_notebook_dir).parent.parent
      project_dir
```

```
[4]: WindowsPath('C:/hyperbox-brain')
```

```
[5]: # Create the path to the training and testing files
      training_file_without_missing = os.path.join(project_dir, Path("dataset/syn_num_train.csv"
      ↪"))
      training_file_with_missing = os.path.join(project_dir, Path("dataset/syn_num_train_
      ↪missing_values.csv"))
      testing_file = os.path.join(project_dir, Path("dataset/syn_num_test.csv"))
```

```
[6]: # Create training and testing data sets
df_train_without_missing = pd.read_csv(training_file_without_missing, header=None)
df_train_with_missing = pd.read_csv(training_file_with_missing, header=None)
df_test = pd.read_csv(testing_file, header=None)

Xy_train_without_missing = df_train_without_missing.to_numpy()
Xy_train_with_missing = df_train_with_missing.to_numpy()
Xy_test = df_test.to_numpy()

Xtr_without_missing = Xy_train_without_missing[:, :-1]
ytr_without_missing = Xy_train_without_missing[:, -1]

Xtr_with_missing = Xy_train_with_missing[:, :-1]
ytr_with_missing = Xy_train_with_missing[:, -1]

Xtest = Xy_test[:, :-1]
ytest = Xy_test[:, -1]
```

```
[7]: # Count number of samples with missing feature values
n_samples = Xtr_with_missing.shape[0]
n_samples_non_missing = sum(np.isnan(Xtr_with_missing).sum(axis=1) == 0)
print("Total number of training samples = ", n_samples)
print("Number of samples with missing feature values = ", n_samples-n_samples_non_
      missing)
```

```
Total number of training samples = 250
Number of samples with missing feature values = 26
```

This tutorial will demonstrate the capability of learning from missing feature values of four learning algorithms of the GFMMNN including original online learning algorithm, improved online learning algorithm, agglomerative learning algorithm, and accelerated agglomerative learning algorithm.

## 1. Original online learning algorithm (OnIn-GFMM) for General Fuzzy Min-Max Neural Network

```
[8]: from hbbrain.numerical_data.incremental_learner.onln_gfmm import OnlineGFMM
```

```
[9]: # Initialise parameters
theta=0.1
theta_min=0.1
gamma=1
is_draw=False
```

### Training a GFMMNN model on the data set without missing feature values

```
[10]: onln_gfmm_clf_without_missing = OnlineGFMM(theta=theta, theta_min=theta_min, gamma=gamma,
↳ is_draw=is_draw)
```

```
[11]: onln_gfmm_clf_without_missing.fit(Xtr_without_missing, ytr_without_missing)
```

```
[11]: OnlineGFMM(C=array([1, 2, 1, 1, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 2, 1,
1, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 2, 2, 1, 1, 2, 1, 2, 1, 2, 1, 1,
2, 2, 1, 2, 2, 2, 2, 2, 1])),
V=array([[0.42413, 0.53516],
[0.70577, 0.397105],
[0.82785, 0.78025],
[0.66038, 0.51128],
[0.48794, 0.672],
[0.26651, 0.18424],
[0.32289, 0.60194],
[0.19944, 0.03],
[0.29343, 0.28975],
[0.63683, 0.6936],
[0.32906, 0.55512],
[0.03, 0.47757],
[0.54...,
[0.25929, 0.81558],
[0.815, 0.397095],
[0.67906, 0.83605],
[0.37033, 0.26124],
[0.52197, 0.91371],
[0.66037, 0.57837],
[0.52621, 0.66846],
[0.80583, 0.43242],
[0.79935, 0.7757],
[0.35813, 0.58772],
[0.79516, 0.32629],
[0.70743, 0.50325],
[0.36057, 0.71561],
[0.72496, 0.38674],
[0.28822, 0.62174],
[0.14737, 0.28498],
[0.56487, 0.17003],
[0.68469, 0.2221],
[0.55763, 0.43813]]),
theta=0.1, theta_min=0.1)
```

```
[12]: print("Number of hyperboxes of the GFMM model trained on the data set without missing_
↳ feature values = ", onln_gfmm_clf_without_missing.get_n_hyperboxes())
```

```
Number of hyperboxes of the GFMM model trained on the data set without missing feature_
↳ values = 53
```

```
[13]: onln_gfmm_clf_without_missing.draw_hyperbox_and_boundary("The trained GFMM classifier_
↳ and its decision boundaries")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

### Training a GFMMN model on the data set WITH missing feature values

```
[14]: onln_gfmm_clf_with_missing = OnlineGFMM(theta=theta, theta_min=theta_min, gamma=gamma,
↪is_draw=is_draw)
```

```
[15]: onln_gfmm_clf_with_missing.fit(Xtr_with_missing, ytr_with_missing)
```

```
[15]: OnlineGFMM(C=array([1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 2, 2, 1, 2, 2, 2, 1, 1, 2, 1, 1, 2,
    2, 2, 2, 1, 1, 2, 2, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 1, 2,
    2, 1, 2, 2, 1, 2, 2, 2, 1])),
    V=array([[0.42413    , 0.47106    ],
    [0.70577    , 0.397105   ],
    [0.82785    , 0.78025    ],
    [0.48794    , 0.672      ],
    [0.66038    , 0.51128    ],
    [0.29343    , 0.26124    ],
    [0.4434     , 0.53516    ],
    [0.32289    , 0.603435   ],
    [0.63683    , 0.6936     ],
    [0.03       , 0.47757    ],
    [0.54181    , 0.43986    ],
    [0.71785    , 0.21704    ],
    [0...
    [0.30005    , 0.19643    ],
    [0.67906    , 0.83605    ],
    [0.16155    , 0.28498    ],
    [0.52197    , 0.87969    ],
    [0.66037    , 0.57837    ],
    [0.49408    , 0.66846    ],
    [0.80583    , 0.43242    ],
    [0.35813    , 0.58772    ],
    [0.79935    , 0.7757     ],
    [0.79516    , 0.32629    ],
    [0.70743    , 0.50325    ],
    [0.36842    , 0.77238    ],
    [0.72496    , 0.34978    ],
    [0.28822    , 0.62174    ],
    [0.91185    , 0.48697    ],
    [0.29163    , 0.086547   ],
    [0.56487    , 0.17003    ],
    [0.68469    , 0.2221     ],
    [0.55763    , 0.43813    ]]),
    theta=0.1, theta_min=0.1)
```

```
[16]: print("Number of hyperboxes of the GFMM model trained on the data set with missing_
↪feature values = ", onln_gfmm_clf_with_missing.get_n_hyperboxes())
```

```
Number of hyperboxes of the GFMM model trained on the data set with missing feature_
↪ values = 53
```

```
[17]: onln_gfmm_clf_with_missing.draw_hyperbox_and_boundary("The trained GFMM classifier and_
↪ its decision boundaries")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

## Make prediction

```
[18]: y_pred_without_missing = onln_gfmm_clf_without_missing.predict(Xtest)
acc_without_missing = accuracy_score(ytest, y_pred_without_missing)
print(f'Accuracy of the model trained on the data set without missing values = {acc_
↪ without_missing * 100: .2f}%')
```

```
y_pred_with_missing = onln_gfmm_clf_with_missing.predict(Xtest)
acc_with_missing = accuracy_score(ytest, y_pred_with_missing)
print(f'Accuracy of the model trained on the data set with missing values = {acc_with_
↪ missing * 100: .2f}%')
```

```
Accuracy of the model trained on the data set without missing values = 84.50%
Accuracy of the model trained on the data set with missing values = 84.10%
```

```
[19]: Xtest_with_missing = np.array([[0.1, np.nan],
                                     [np.nan, 0.5]])
onln_gfmm_clf_with_missing.predict_with_membership(Xtest_with_missing)
```

```
[19]: array([[0.79877, 0.96097],
             [1.      , 1.      ]])
```

From the illustration of resulting hyperboxes, we can see that for input sample  $X=[0.1, \_]$ , there is no blue (class 1) hyperboxes surrounding the vertical line  $x = 0.1$  and the distance from the green (class 2) hyperboxes to this vertical line ( $x = 0.1$ ) is shorter than the distance from the blue (class 1) hyperboxes to this line. Therefore, the membership value of class 2 (0.96097) is higher than the membership value of class 1 (0.79877) in this case. For input sample  $X = [\_, 0.5]$ , we can see that there are both green hyperboxes and blue hyperboxes crossing out the horizontal line ( $y = 0.5$ ). Therefore, the membership values for both classes in this case are 1.

## 2. Improved online learning algorithm (IOL-GFMM) for General Fuzzy Min-Max Neural Network

```
[20]: from hbbrain.numerical_data.incremental_learner.iol_gfmm import ImprovedOnlineGFMM
```

```
[21]: # Initialise parameters
theta=0.1
gamma=1
is_draw=False
```

## Training a GFMMNN model on the data set without missing feature values

```
[22]: iol_gfmm_clf_without_missing = ImprovedOnlineGFMM(theta=theta, gamma=gamma, is_draw=is_
      ↪ draw)
```

```
[23]: iol_gfmm_clf_without_missing.fit(Xtr_without_missing, ytr_without_missing)
```

```
[23]: ImprovedOnlineGFMM(C=array([[1, 2, 1, 1, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 2, 2, 2, 1,
      ↪ 2, 1,
      2, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 2, 2, 1, 1, 1, 2, 1, 2,
      1, 2, 2, 1, 2, 1, 1, 2, 2, 1, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2,
      2, 2]]),
      N_samples=array([11, 3, 2, 10, 5, 6, 6, 2, 7, 6, 3, 1, 7,
      ↪ 6, 2, 11, 1,
      5, 7, 2, 3, 9, 3, 4, 6, 9, 10, 5, 8, 13, 4, 4, 3, 3,
      6, 4, 2, 1, 2, 1, 1, 1, 2, 1, 2, 1, 1, 2, 1, 1, 1,
      5, 2, 1, 1, 6, 1, 5, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1]),
      V=array([[0.42413, 0.5351...
      [0.52621 , 0.59493 ],
      [0.79935 , 0.7757 ],
      [0.6248 , 0.39922 ],
      [0.79516 , 0.32629 ],
      [0.66562 , 0.36352 ],
      [0.36057 , 0.71561 ],
      [0.72496 , 0.38674 ],
      [0.70743 , 0.50325 ],
      [0.40233 , 0.67232 ],
      [0.28822 , 0.62174 ],
      [0.14737 , 0.28498 ],
      [0.75421 , 0.40498 ],
      [0.59655 , 0.56029 ],
      [0.91185 , 0.48697 ],
      [0.6504 , 0.51624 ],
      [0.68853 , 0.41466 ],
      [0.56487 , 0.17003 ],
      [0.59235 , 0.54123 ],
      [0.68469 , 0.2221 ]]),
      theta=0.1)
```

```
[24]: print("Number of hyperboxes of the GFMM model trained on the data set without missing_
      ↪ feature values = ", iol_gfmm_clf_without_missing.get_n_hyperboxes())
```

```
Number of hyperboxes of the GFMM model trained on the data set without missing feature_
      ↪ values = 68
```

```
[25]: iol_gfmm_clf_without_missing.draw_hyperbox_and_boundary("The trained GFMM classifier and_
      ↪ its decision boundaries")
```

```
<IPython.core.display.Javascript object>
```



```
<IPython.core.display.HTML object>
```

### Training a GFMMN model on the data set WITH missing feature values

```
[26]: iol_gfmm_clf_with_missing = ImprovedOnlineGFMM(theta=theta, gamma=gamma, is_draw=is_draw)
```

```
[27]: iol_gfmm_clf_with_missing.fit(Xtr_with_missing, ytr_with_missing)
```

```
[27]: ImprovedOnlineGFMM(C=array([[1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 1, 2,
↪1, 1,
      2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 2, 1, 1, 2, 2,
      2, 2, 1, 2, 2, 1, 1, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2]]),
      N_samples=array([ 4,  3,  2,  6,  8, 10,  8,  7,  6,  1,  9,  6,  2,
↪3,  5,  7,  2,
      3,  7,  3,  4,  9,  9, 10,  7,  7, 11,  4,  3,  5,  4,  6,  1, 12,
      1,  3,  2,  4,  1,  3,  4,  2,  1,  1,  1,  1,  2,  3,  2,  1,  3,
      5,  1,  3,  3,  1,  2,  1,  1,  1,  1,  1,  1,  1]),
      V=array([[0.39868 , 0.47106 ],
      [0.77074 , 0.48234 ],...
      [0.65327 , 0.51585 ],
      [0.57962 , 0.57837 ],
      [0.79935 , 0.7757  ],
      [0.6248  , 0.39922 ],
      [0.79516 , 0.32629 ],
      [0.66562 , 0.36352 ],
      [0.36842 , 0.77238 ],
      [0.72496 , 0.34978 ],
      [0.70743 , 0.50325 ],
      [0.28822 , 0.62174 ],
      [0.38312 , 0.67232 ],
      [0.75421 , 0.40498 ],
      [0.59655 , 0.56029 ],
      [0.91185 , 0.48697 ],
      [0.6504  , 0.51624 ],
      [0.29163 , 0.086547],
      [0.68853 , 0.41466 ],
      [0.56487 , 0.17003 ],
      [0.68469 , 0.2221  ]]),
      theta=0.1)
```

```
[28]: print("Number of hyperboxes of the GFMM model trained on the data set with missing_
↪feature values = ", iol_gfmm_clf_with_missing.get_n_hyperboxes())
```

```
Number of hyperboxes of the GFMM model trained on the data set with missing feature_
↪values = 63
```

```
[29]: iol_gfmm_clf_with_missing.draw_hyperbox_and_boundary("The trained GFMM classifier and_
↪its decision boundaries")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

### Make prediction

```
[30]: y_pred_without_missing_iol = iol_gfmm_clf_without_missing.predict(Xtest)
      acc_without_missing = accuracy_score(ytest, y_pred_without_missing_iol)
      print(f'Accuracy of the model trained on the data set without missing values = {acc_
      ↪without_missing * 100: .2f}%')

      y_pred_with_missing_iol = iol_gfmm_clf_with_missing.predict(Xtest)
      acc_with_missing = accuracy_score(ytest, y_pred_with_missing_iol)
      print(f'Accuracy of the model trained on the data set with missing values = {acc_with_
      ↪missing * 100: .2f}%')
```

```
Accuracy of the model trained on the data set without missing values = 87.20%
Accuracy of the model trained on the data set with missing values = 86.70%
```

### Using membership value to show the prediction for the testing data with missing features values

```
[31]: Xtest_with_missing = np.array([[0.1, np.nan],
                                     [np.nan, 0.5]])
      iol_gfmm_clf_with_missing.predict_with_membership(Xtest_with_missing)

[31]: array([[0.79877, 0.96097],
            [1.      , 1.      ]])
```

The results and explanation are similar to the case for the original online learning algorithm

### 3. Accelerated agglomerative learning algorithm (AGGLO-2) for General Fuzzy Min-Max Neural Network

```
[32]: from hbbrain.numerical_data.batch_learner.accel_agglo_gfmm import _
      ↪AccelAgglomerativeLearningGFMM

[33]: # Initialise parameters for the accelerated agglomerative learning algorithm
      theta = 0.1
      gamma = 1
      min_simil = 0
      simil_measure = 'mid'
      is_draw = False
```

### Training a GFMMNN model on the data set without missing feature values

```
[34]: agгло2_gfmm_clf_without_missing = AccelAgglomerativeLearningGFMM(theta=theta,
↳ gamma=gamma, min_simil=min_simil, simil_measure=simil_measure, is_draw=is_draw)

[35]: agгло2_gfmm_clf_without_missing.fit(Xtr_without_missing, ytr_without_missing)

[35]: AccelAgglomerativeLearningGFMM(min_simil=0, theta=0.1)

[36]: print("Number of hyperboxes of the GFMM model trained on the data set without missing
↳ feature values = ", agгло2_gfmm_clf_without_missing.get_n_hyperboxes())

Number of hyperboxes of the GFMM model trained on the data set without missing feature
↳ values = 62

[37]: agгло2_gfmm_clf_without_missing.draw_hyperbox_and_boundary("The trained GFMM classifier
↳ and its decision boundaries")

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>
```

### Training a GFMMNN model on the data set WITH missing feature values

```
[38]: agгло2_gfmm_clf_with_missing = AccelAgglomerativeLearningGFMM(theta=theta, gamma=gamma,
↳ min_simil=min_simil, simil_measure=simil_measure, is_draw=is_draw)

[39]: agгло2_gfmm_clf_with_missing.fit(Xtr_with_missing, ytr_with_missing)

[39]: AccelAgglomerativeLearningGFMM(min_simil=0, theta=0.1)

[40]: print("Number of hyperboxes of the GFMM model trained on the data set with missing
↳ feature values = ", agгло2_gfmm_clf_with_missing.get_n_hyperboxes())

Number of hyperboxes of the GFMM model trained on the data set with missing feature
↳ values = 61

[41]: agгло2_gfmm_clf_with_missing.draw_hyperbox_and_boundary("The trained GFMM classifier and
↳ its decision boundaries")

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>
```

### Make prediction

```
[42]: y_pred_without_missing_agglo2 = agglo2_gfmm_clf_without_missing.predict(Xtest)
      acc_without_missing = accuracy_score(ytest, y_pred_without_missing_agglo2)
      print(f'Accuracy of the model trained on the data set without missing values = {acc_
      ↪without_missing * 100: .2f}%')

      y_pred_with_missing_agglo2 = agglo2_gfmm_clf_with_missing.predict(Xtest)
      acc_with_missing = accuracy_score(ytest, y_pred_with_missing_agglo2)
      print(f'Accuracy of the model trained on the data set with missing values = {acc_with_
      ↪missing * 100: .2f}%')

      Accuracy of the model trained on the data set without missing values = 85.00%
      Accuracy of the model trained on the data set with missing values = 85.20%
```

### Using membership value to show the prediction for the testing data with missing features values

```
[43]: Xtest_with_missing = np.array([[0.1, np.nan],
      [np.nan, 0.5]])
      agglo2_gfmm_clf_with_missing.predict_with_membership(Xtest_with_missing)

[43]: array([[0.79877, 0.96097],
      [1.        , 1.        ]])
```

## 4. Agglomerative learning algorithm with full similarity matrix (AGGLO-SM) for General Fuzzy Min-Max Neural Network

```
[44]: from hbbrain.numerical_data.batch_learner.agglo_gfmm import AgglomerativeLearningGFMM

[45]: # Initialise parameters for the accelerated agglomerative learning algorithm
      theta = 0.1
      gamma = 1
      min_simil = 0
      simil_measure = 'mid'
      is_draw = False
```

### Training a GFMMNN model on the data set without missing feature values

```
[46]: agglomsm_gfmm_clf_without_missing = AgglomerativeLearningGFMM(theta=theta, gamma=gamma,
      ↪min_simil=min_simil, simil_measure=simil_measure, is_draw=is_draw)

[47]: agglomsm_gfmm_clf_without_missing.fit(Xtr_without_missing, ytr_without_missing)

[47]: AgglomerativeLearningGFMM(min_simil=0, theta=0.1)

[48]: print("Number of hyperboxes of the GFMM model trained on the data set without missing_
      ↪feature values = ", agglomsm_gfmm_clf_without_missing.get_n_hyperboxes())
```

```
Number of hyperboxes of the GFMM model trained on the data set without missing feature_
↳ values = 62
```

```
[49]: agglomsm_gfmm_clf_without_missing.draw_hyperbox_and_boundary("The trained GFMM classifier_
↳ and its decision boundaries")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

### Training a GFMMNN model on the data set WITH missing feature values

```
[50]: agglomsm_gfmm_clf_with_missing = AgglomerativeLearningGFMM(theta=theta, gamma=gamma, min_
↳ simil=min_simil, simil_measure=simil_measure, is_draw=is_draw)
```

```
[51]: agglomsm_gfmm_clf_with_missing.fit(Xtr_with_missing, ytr_with_missing)
```

```
[51]: AgglomerativeLearningGFMM(min_simil=0, theta=0.1)
```

```
[52]: print("Number of hyperboxes of the GFMM model trained on the data set WITH missing_
↳ feature values = ", agglomsm_gfmm_clf_with_missing.get_n_hyperboxes())
```

```
Number of hyperboxes of the GFMM model trained on the data set WITH missing feature_
↳ values = 60
```

```
[53]: agglomsm_gfmm_clf_with_missing.draw_hyperbox_and_boundary("The trained GFMM classifier_
↳ and its decision boundaries")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

### Make prediction

```
[54]: y_pred_without_missing_agglosm = agglomsm_gfmm_clf_without_missing.predict(Xtest)
acc_without_missing = accuracy_score(ytest, y_pred_without_missing_agglosm)
print(f'Accuracy of the model trained on the data set without missing values = {acc_
↳ without_missing * 100: .2f}%')
```

```
y_pred_with_missing_agglosm = agglomsm_gfmm_clf_with_missing.predict(Xtest)
acc_with_missing = accuracy_score(ytest, y_pred_with_missing_agglosm)
print(f'Accuracy of the model trained on the data set with missing values = {acc_with_
↳ missing * 100: .2f}%')
```

```
Accuracy of the model trained on the data set without missing values = 85.20%
Accuracy of the model trained on the data set with missing values = 84.40%
```

### Using membership value to show the prediction for the testing data with missing features values

```
[55]: Xtest_with_missing = np.array([[0.1, np.nan],
                                     [np.nan, 0.5]])
agglosm_gfmm_clf_with_missing.predict_with_membership(Xtest_with_missing)

[55]: array([[0.79877, 0.96097],
            [1.      , 1.      ]])
```

### Using probability and membership values of classes for prediction

This example shows how to use probability and membership values of class labels for prediction when applying single hyperbox-based models, ensemble models of hyperbox-based classifiers, and multigranular hyperbox-based models. We employ the original online learning algorithm for general fuzzy min-max neural network, accelerated agglomerative learning algorithm for general fuzzy min-max neural network, Simpson's online learning algorithm for fuzzy min-max neural network, bagging of hyperbox-based models, and multigranular hyperbox-based models for demonstration in this tutorial.

```
[1]: %matplotlib notebook
```

```
[2]: import os
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
```

### Load training and testing datasets

```
[3]: # Get the path to the this jupyter notebook file
this_notebook_dir = os.path.dirname(os.path.abspath("__file__"))
this_notebook_dir
```

```
[3]: 'C:\\hyperbox-brain\\examples\\other_learning_ability_gfmm'
```

```
[4]: # Get the home folder of the hyperbox-brain toolbox
from pathlib import Path
project_dir = Path(this_notebook_dir).parent.parent
project_dir
```

```
[4]: WindowsPath('C:/hyperbox-brain')
```

```
[5]: # Create the path to the training and testing files
training_file = os.path.join(project_dir, Path("dataset/syn_num_train.csv"))
testing_file = os.path.join(project_dir, Path("dataset/syn_num_test.csv"))
```

```
[6]: # Create training and testing data sets
df_train = pd.read_csv(training_file, header=None)
df_test = pd.read_csv(testing_file, header=None)

Xy_train = df_train.to_numpy()
```

(continues on next page)

(continued from previous page)

```

Xy_test = df_test.to_numpy()

Xtr = Xy_train[:, :-1]
ytr = Xy_train[:, -1]

Xtest = Xy_test[:, :-1]
ytest = Xy_test[:, -1]

```

## 1. Original online learning algorithm for General fuzzy min-max neural network (OnIn-GFMM)

```
[7]: from hbbrain.numerical_data.incremental_learner.onln_gfmm import OnlineGFMM
```

```
[8]: # Initializing parameters
      theta = 0.1
      theta_min = 0.1
      gamma = 4
      is_draw = False
```

### Train a model

```
[9]: onln_gfmm_clf = OnlineGFMM(theta=theta, theta_min=theta_min, gamma=gamma, is_draw=is_
    ↪ draw)
    onln_gfmm_clf.fit(Xtr, ytr)
```

```
[9]: OnlineGFMM(C=array([1, 2, 1, 1, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 2, 1,
    1, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 2, 2, 1, 1, 2, 1, 2, 1, 2, 1, 1,
    2, 2, 1, 2, 2, 2, 2, 2, 1]),
    V=array([[0.42413    , 0.53516    ],
    [0.70577    , 0.397105   ],
    [0.82785    , 0.78025    ],
    [0.66038    , 0.51128    ],
    [0.48794    , 0.672     ],
    [0.26651    , 0.18424    ],
    [0.32289    , 0.60194    ],
    [0.19944    , 0.03      ],
    [0.29343    , 0.28975    ],
    [0.63683    , 0.6936     ],
    [0.32906    , 0.55512    ],
    [0.03       , 0.47757    ],
    [0.54...
    [0.815     , 0.397095   ],
    [0.67906    , 0.83605    ],
    [0.37033    , 0.26124    ],
    [0.52197    , 0.91371    ],
    [0.66037    , 0.57837    ],
    [0.52621    , 0.66846    ],
    [0.80583    , 0.43242    ],
    [0.79935    , 0.7757     ],
    [0.35813    , 0.58772    ],
```

(continues on next page)

(continued from previous page)

```

[0.79516 , 0.32629 ],
[0.70743 , 0.50325 ],
[0.36057 , 0.71561 ],
[0.72496 , 0.38674 ],
[0.28822 , 0.62174 ],
[0.14737 , 0.28498 ],
[0.56487 , 0.17003 ],
[0.68469 , 0.2221  ],
[0.55763 , 0.43813 ]]),
gamma=4, theta=0.1, theta_min=0.1)

```

## Make prediction

Use probability values and membership values to make prediction for the first ten testing samples. The orders of columns are also the orders of class labels in an ascending order. In this example, the first column contains the predicted values for class 1, and the second column contains the predicted values for class 2.

The predicted class probability is the fraction of the membership value of the representative hyperbox of that class and the sum of all membership values of all representative hyperboxes of all classes. The predicted class membership value is the membership value of the representative hyperbox of that class.

```
[10]: print("Input classes: ", np.unique(ytr))
```

```
Input classes:  [1. 2.]
```

```
[11]: onln_gfmm_clf.predict_proba(Xtest[:10])
```

```
[11]: array([[0.50044451, 0.49955549],
 [0.48259685, 0.51740315],
 [0.42006751, 0.57993249],
 [0.52674382, 0.47325618],
 [0.46011316, 0.53988684],
 [0.50352398, 0.49647602],
 [0.49915114, 0.50084886],
 [0.3190052 , 0.6809948 ],
 [0.50079564, 0.49920436],
 [0.44152243, 0.55847757]])
```

```
[12]: onln_gfmm_clf.predict_with_membership(Xtest[:10])
```

```
[12]: array([[0.95696 , 0.95526 ],
 [0.76536 , 0.82056 ],
 [0.72176 , 0.99644 ],
 [0.94304 , 0.84728 ],
 [0.85224 , 1.         ],
 [0.96876 , 0.9552  ],
 [0.92908 , 0.93224 ],
 [0.46844 , 1.         ],
 [1.       , 0.9968225],
 [0.78976 , 0.99896 ]])
```



## 2. Accelerated agglomerative learning algorithm for General fuzzy min-max neural network (AGGLO-2)

```
[13]: from hbbrain.numerical_data.batch_learner.accel_agglo_gfmm import _
      ↪ AccelAgglomerativeLearningGFMM
```

```
[14]: # Initializing parameters
      theta=0.1
      gamma=4
      min_simil=0
      simil_measure='long'
      is_draw=False
```

### Train a model

```
[15]: aggro2_gfmm_clf = AccelAgglomerativeLearningGFMM(theta=theta, gamma=gamma, min_simil=min_
      ↪ simil, simil_measure=simil_measure, is_draw=is_draw)
      aggro2_gfmm_clf.fit(Xtr, ytr)

[15]: AccelAgglomerativeLearningGFMM(gamma=4, min_simil=0, simil_measure='long',
      theta=0.1)
```

### Make prediction

```
[16]: aggro2_gfmm_clf.predict_proba(Xtest[:10])
```

```
[16]: array([[0.50882641, 0.49117359],
             [0.47279466, 0.52720534],
             [0.42148046, 0.57851954],
             [0.56160076, 0.43839924],
             [0.46758668, 0.53241332],
             [0.50352398, 0.49647602],
             [0.49915114, 0.50084886],
             [0.20099716, 0.79900284],
             [0.49770692, 0.50229308],
             [0.44615176, 0.55384824]])
```

```
[17]: aggro2_gfmm_clf.predict_with_membership(Xtest[:10])
```

```
[17]: array([[0.95696, 0.92376],
             [0.76536, 0.85344],
             [0.72176, 0.99068],
             [0.94304, 0.73616],
             [0.87824, 1.      ],
             [0.96876, 0.9552 ],
             [0.92908, 0.93224],
             [0.25156, 1.      ],
             [0.9116 , 0.92   ],
             [0.78976, 0.9804 ]])
```

### 3. Original online learning algorithm for Simpson's Fuzzy min-max neural network (FMNN)

```
[18]: from hbbrain.numerical_data.incremental_learner.fmn import FMNNClassifier
```

```
[19]: # Initializing parameters
      theta = 0.1
      gamma = 4
      is_draw = False
```

#### Train a model

```
[20]: fmnn_clf = FMNNClassifier(theta=theta, gamma=gamma, is_draw=is_draw)
      fmnn_clf.fit(Xtr, ytr)
```

```
[20]: FMNNClassifier(C=array([[1, 2, 1, 1, 1, 2, 1, 2, 1, 2, 2, 2, 1, 1, 2, 1, 2, 1, 2, 2, 2, 2,
    2, 2, 1, 2, 2, 1, 1, 2, 2, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 1,
    2, 1, 1, 1, 2, 2, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 2,
    2, 1, 1, 2, 2, 1, 2, 1, 1, 2, 1, 1, 2, 1, 2, 2, 2, 1, 1, 1, 1, 2,
    1, 1, 1]]),
    V=array([[0.36239, 0.55942],
    [0.64082, 0.43016875],
    [0.91059, 0.82085],
    [0.65328, 0.50326],
    [0.46107, 0.68306],
    [0.29812, 0.18424],
    [0.33593, 0.68775],
    [0.1...,
    [0.25621, 0.62174],
    [0.42403, 0.7592],
    [0.79157, 0.59996],
    [0.72496, 0.34978],
    [0.36842, 0.76576],
    [0.73681, 0.71261],
    [0.66773, 0.31155],
    [0.32289, 0.6747],
    [0.28077, 0.27116],
    [0.61106, 0.28476],
    [0.75421, 0.40498],
    [0.38038, 0.67232],
    [0.36745, 0.52006],
    [0.91185, 0.48697],
    [0.35813, 0.58584],
    [0.25924, 0.42696],
    [0.70685, 0.64383],
    [0.75047, 0.6092],
    [0.72842, 0.61048]]),
    gamma=4, theta=0.1)
```

**Make prediction**

```
[21]: fmnn_clf.predict_proba(Xtest[:10])
```

```
[21]: array([[0.5001282 , 0.4998718 ],
          [0.49113606, 0.50886394],
          [0.47048553, 0.52951447],
          [0.51190571, 0.48809429],
          [0.49033913, 0.50966087],
          [0.5028714 , 0.4971286 ],
          [0.49750079, 0.50249921],
          [0.41923606, 0.58076394],
          [0.49781361, 0.50218639],
          [0.47269621, 0.52730379]])
```

```
[22]: fmnn_clf.predict_with_membership(Xtest[:10])
```

```
[22]: array([[0.9801625 , 0.97966   ],
          [0.92338   , 0.95671   ],
          [0.8885225 , 1.         ],
          [0.96807   , 0.92304   ],
          [0.96158875, 0.99948   ],
          [1.         , 0.98858   ],
          [0.97989   , 0.989735  ],
          [0.72187   , 1.         ],
          [0.9912925 , 1.         ],
          [0.89644   , 1.         ]])
```

**4. Bagging of base general fuzzy min-max neural networks trained by the original online learning algorithm**

```
[23]: from hbbrain.numerical_data.ensemble_learner.decision_comb_bagging import_
      ↪ DecisionCombinationBagging
```

```
[24]: # Initialise parameters
n_estimators = 20 # number of base learners
max_samples = 0.5 # sampling rate for samples
bootstrap = False # random subsampling without replacement
class_balanced = False # do not use the class-balanced sampling mode
n_jobs = 4 # number of processes is used to build base learners
```

**Train a bagging model**

```
[25]: # Init a hyperbox-based model used to train base learners
      # Using the GFMM classifier with the original online learning algorithm with the maximum_
      ↪ hyperbox size 0.1
base_estimator = OnlineGFMM(theta=theta, gamma=gamma)
```

```
[26]: dc_bagging_subsampling = DecisionCombinationBagging(base_estimator=base_estimator, n_
↳ estimators=n_estimators, max_samples=max_samples, bootstrap=bootstrap, class_
↳ balanced=class_balanced, n_jobs=n_jobs, random_state=0)
```

```
[27]: dc_bagging_subsampling.fit(Xtr, ytr)
```

```
[27]: DecisionCombinationBagging(base_estimator=OnlineGFMM(C=array([], dtype=float64),
V=array([], dtype=float64),
W=array([], dtype=float64),
gamma=4, theta=0.1),
n_estimators=20, n_jobs=4, random_state=0)
```

## Make prediction

This example shows how to use `predict_proba` and `predict_with_membership` functions to make prediction. The predicted class probabilities of an input sample with respect to an ensemble model are computed as the mean predicted class probabilities of the hyperbox-based learners in the ensemble model. The class probability of a single hyperbox-based learner is the fraction of the membership value of the representative hyperbox of that class and the sum of all membership values of all representative hyperboxes of all classes.

The predicted class memberships of an input sample are computed as the mean predicted class memberships of the hyperbox-based learners in the ensemble model. The class membership of a single hyperbox-based learner is the membership from the input X to the representative hyperbox of that class to join the prediction procedure.

```
[28]: dc_bagging_subsampling.predict_proba(Xtest[:10])
```

```
[28]: array([[0.47815396, 0.52184604],
[0.5012498 , 0.4987502 ],
[0.3955224 , 0.6044776 ],
[0.54362581, 0.45637419],
[0.47159695, 0.52840305],
[0.52292342, 0.47707658],
[0.49756119, 0.50243881],
[0.1964586 , 0.8035414 ],
[0.47362699, 0.52637301],
[0.39199205, 0.60800795]])
```

```
[29]: dc_bagging_subsampling.predict_with_membership(Xtest[:10])
```

```
[29]: array([[0.843956 , 0.91573 ],
[0.818432 , 0.810635 ],
[0.639162 , 0.963144 ],
[0.87822 , 0.7370675 ],
[0.858645 , 0.958337 ],
[0.98751575, 0.90252725],
[0.916385 , 0.927697 ],
[0.232785 , 0.932006 ],
[0.86215975, 0.953138 ],
[0.6373855 , 0.977412 ]])
```

## 5. Multi-resolution Hierarchical Granular Representation based Classifier using GFMM

```
[30]: from hbbrain.numerical_data.multigranular_learner.multi_resolution_gfmm import
      ↪ MultiGranularGFMM

[31]: # Initializing parameters
      # number of disjoint partitions to build base learners
      n_partitions = 4
      # a list of maximum hyperbox sizes for granularity levels
      granular_theta = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
      # minimum membership values between two hyperboxes aggregated at higher abstraction
      ↪ levels
      min_membership_aggregation = 0.1
      # the speed of decreasing of membership values
      gamma = 4
```

### Training a multigranular model

```
[32]: from hbbrain.constants import HETEROGENEOUS_CLASS_LEARNING
      multi_granular_gfmm_clf = MultiGranularGFMM(n_partitions=n_partitions, granular_
      ↪ theta=granular_theta, gamma=gamma, min_membership_aggregation=min_membership_
      ↪ aggregation)
      # Training using the heterogeneous model for class labels.
      multi_granular_gfmm_clf.fit(Xtr, ytr, learning_type=HETEROGENEOUS_CLASS_LEARNING)

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done 2 out of 4 | elapsed: 2.7s remaining: 2.7s
[Parallel(n_jobs=4)]: Done 4 out of 4 | elapsed: 2.7s finished

[32]: MultiGranularGFMM(gamma=4, granular_theta=[0.1, 0.2, 0.3, 0.4, 0.5, 0.6],
      min_membership_aggregation=0.1)
```

### Make prediction

The predicted class probability at a given granularity level is the fraction of the membership value of the representative hyperbox of that class at the given granularity level and the sum of all membership values of all representative hyperboxes of all classes joining the prediction procedure. If the given granularity level gets the values of -1, then the predicted class probability value for each sample is the average of probability values at all available granularity levels. Similar meaning is applied for the predicted class membership values.

```
[33]: # Predicted class probability values for the first ten samples based on the average
      ↪ values of all available granularity levels
      multi_granular_gfmm_clf.predict_proba(Xtest[:10], level=-1)

[33]: array([[0.51102808, 0.48897192],
      [0.54380893, 0.45619107],
      [0.39490266, 0.60509734],
      [0.54865927, 0.45134073],
      [0.46152604, 0.53847396],
      [0.52639798, 0.47360202],
      [0.48346396, 0.51653604],
```

(continues on next page)

(continued from previous page)

```
[0.2453813 , 0.7546187 ],
[0.45120077, 0.54879923],
[0.44126587, 0.55873413]])
```

```
[34]: # Predicted class probability values for the first ten samples at the second granularity_
      ↪ level
      multi_granular_gfmm_clf.predict_proba(Xtest[:10], level=1)
```

```
[34]: array([[0.51670508, 0.48329492],
            [0.54928154, 0.45071846],
            [0.38979741, 0.61020259],
            [0.54133645, 0.45866355],
            [0.45073053, 0.54926947],
            [0.51150372, 0.48849628],
            [0.47904728, 0.52095272],
            [0.20099716, 0.79900284],
            [0.43810123, 0.56189877],
            [0.44126587, 0.55873413]])
```

```
[35]: # Predicted class membership values for the first ten samples based on the average_
      ↪ values of all available granularity levels
      multi_granular_gfmm_clf.predict_with_membership(Xtest[:10], level=-1)
```

```
[35]: array([[0.98879333, 0.94611667],
            [0.97816   , 0.82056   ],
            [0.65262667, 1.         ],
            [0.9936    , 0.81736   ],
            [0.8571    , 1.         ],
            [1.         , 0.89970333],
            [0.93597333, 1.         ],
            [0.32505333, 0.99963333],
            [0.82216   , 1.         ],
            [0.78976   , 1.         ]])
```

```
[36]: # Predicted class membership values for the first ten samples at the second granularity_
      ↪ level
      multi_granular_gfmm_clf.predict_with_membership(Xtest[:10], level=1)
```

```
[36]: array([[1.         , 0.93534],
            [1.         , 0.82056],
            [0.6388   , 1.         ],
            [1.         , 0.84728],
            [0.8206   , 1.         ],
            [1.         , 0.95502],
            [0.91956   , 1.         ],
            [0.25156   , 1.         ],
            [0.77968   , 1.         ],
            [0.78976   , 1.         ]])
```

## Data editing using general fuzzy min-max neural network

This example shows how to using the general fuzzy min-max neural network (GFMMNN) to edit the training data and remove outliers. There are three different data editing approaches which are presented in [1]. This example uses a GFMMNN trained by an accelerated agglomerative learning algorithm for demonstration, but any other learning algorithms of the GFMMNN can be used within data editing functions.

[1] Gabrys, B. (2001). Data editing for neuro-fuzzy classifiers. In Proceedings of the Fourth International ICSC Symposia on Soft Computing and Intelligent Systems for Industry.

```
[1]: %matplotlib notebook
```

```
[2]: import os
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
```

## Load training and testing datasets

```
[3]: # Get the path to the this jupyter notebook file
this_notebook_dir = os.path.dirname(os.path.abspath("__file__"))
this_notebook_dir
```

```
[3]: 'C:\\hyperbox-brain\\examples\\other_learning_ability_gfmm'
```

```
[4]: # Get the home folder of the hyperbox-brain toolbox
from pathlib import Path
project_dir = Path(this_notebook_dir).parent.parent
project_dir
```

```
[4]: WindowsPath('C:/hyperbox-brain')
```

```
[5]: # Create the path to the training and testing files
training_file = os.path.join(project_dir, Path("dataset/syn_num_train.csv"))
testing_file = os.path.join(project_dir, Path("dataset/syn_num_test.csv"))
```

```
[6]: # Create training and testing data sets
df_train = pd.read_csv(training_file, header=None)
df_test = pd.read_csv(testing_file, header=None)

Xy_train = df_train.to_numpy()
Xy_test = df_test.to_numpy()

Xtr = Xy_train[:, :-1]
ytr = Xy_train[:, -1]

Xtest = Xy_test[:, :-1]
ytest = Xy_test[:, -1]
```

```
[7]: from hbbrain.numerical_data.batch_learner.accel_agglo_gfmm import _
↪ AccelAgglomerativeLearningGFMM
```

```
[8]: # Initialise a gfm model by the accelerated agglomerative learning algorithm
theta = 0.2
gamma = 1
min_simil = 0
simil_measure = 'mid'
is_draw = False
```

```
[9]: import matplotlib.pyplot as plt
```

```
[10]: # Create a function to show input samples (in this case only two classes)
def show_2d_samples(X, y, title="Input sample"):
    colours = ['blue', 'green', 'red', 'black']
    labels = ['class 1', 'class 2', 'class 3', 'class 4']
    fig1, ax1 = plt.subplots()
    classes = np.unique(y)
    for index, c in enumerate(classes):
        data = X[y == c]
        color = colours[index]
        label = labels[index]
        ax1.scatter(
            data[:, 0],
            data[:, 1],
            color=color,
            label=label
        )

    ax1.set_xlabel('X1')
    ax1.set_ylabel('X2')
    ax1.title.set_text(title)
    ax1.legend(loc='lower right')

    fig1.show()
```

### Verify the input data before doing data editing

```
[11]: # Show the existing training set
show_2d_samples(Xtr, ytr, "Input data set before editing")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[12]: print("Number of input samples before editing = ", Xtr.shape[0])
```

```
Number of input samples before editing = 250
```



**Build a GFMMNN model from the original input training data**

```
[13]: agгло2_clf_original_data = AccelAgglomerativeLearningGFMM(theta=theta, gamma=gamma, min_
      ↪simil=min_simil, simil_measure=simil_measure, is_draw=is_draw)

[14]: agгло2_clf_original_data.fit(Xtr, ytr)

[14]: AccelAgglomerativeLearningGFMM(min_simil=0, theta=0.2)
```

**Display decision boundaries among classes if input data are 2-dimensional**

```
[15]: agгло2_clf_original_data.draw_hyperbox_and_boundary("The trained GFMM classifier and its_
      ↪decision boundaries")

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

[16]: print("Number of hyperboxes = ", agгло2_clf_original_data.get_n_hyperboxes())

Number of hyperboxes = 43
```

**Make prediction**

```
[17]: from sklearn.metrics import accuracy_score

[18]: y_pred_original_data = agгло2_clf_original_data.predict(Xtest)
      acc = accuracy_score(ytest, y_pred_original_data)
      print(f'Accuracy = {acc * 100: .2f}%')

Accuracy = 87.00%
```

**1. Method 1 of Data editing: Using k-nearest neighbors and a leave-one-out scheme.**

The first of the data editing procedures examined in this paper follows the general data editing method presented in [2]. It is based on an application of the k-nearest neighbor classification rules within a leave-one-out scheme. To make it suitable for the GFMM algorithm, the membership function of the GFMMNN acts as the similarity measure utilised for finding k-nearest neighbours. In this case, the hyperboxes represent individual training data points and the outputs of the hyperbox layer nodes can be treated as the classification values. The k maximum values from the hyperbox layer outputs are used together with the above classification rules to decide whether the input is classified correctly or not.

[2] Webb, A. R. (2003). Statistical pattern recognition. John Wiley & Sons.

```
[19]: # Import the data editing function
      from hbbbrain.utils.data_editing import data_editing_leave_one_out

[20]: # Initialise a GFMMNN model using the accelerated agglomerative learning algorithm
      agгло2_clf = AccelAgglomerativeLearningGFMM(theta=theta, gamma=gamma, min_simil=min_
      ↪simil, simil_measure=simil_measure, is_draw=is_draw)
```

```
[21]: # Initialise input paramters of the data editing method
      k_neighbors=5 # Using the 5-NN rule for classification
      n_iters=100 # Repeat 100 times
      n_last_iters=5 # If 5 consecutive iterations not remove any sample, stop the loop
      seed=0 # kernel seed for random split data
```

```
[22]: # Doing data editing and get the output dataset
      Xtr_1, ytr_1 = data_editing_leave_one_out(Xtr, ytr, gfm_estimator=agglo2_clf, k_
      ↪ neighbors=k_neighbors, n_iters=n_iters, n_last_iters=n_last_iters, seed=seed)
```

```
[23]: print("Number of remaining samples after editing = ", Xtr_1.shape[0])
```

```
Number of remaining samples after editing = 183
```

```
[24]: # Show the remaining training set
      show_2d_samples(Xtr_1, ytr_1, "Input data set after editing by method 1")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

### Build a GFMMNN model from the remaining training data after editing

```
[25]: agglo2_clf_editing_method_1 = AccelAgglomerativeLearningGFMM(theta=theta, gamma=gamma,
      ↪ min_simil=min_simil, simil_measure=simil_measure, is_draw=is_draw)
```

```
[26]: agglo2_clf_editing_method_1.fit(Xtr_1, ytr_1)
```

```
[26]: AccelAgglomerativeLearningGFMM(min_simil=0, theta=0.2)
```

### Display decision boundaries among classes if input data are 2-dimensional

```
[27]: agglo2_clf_editing_method_1.draw_hyperbox_and_boundary("The trained GFMM classifier and
      ↪ its decision boundaries")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[28]: print("Number of hyperboxes = ", agglo2_clf_editing_method_1.get_n_hyperboxes())
```

```
Number of hyperboxes = 16
```

## Make prediction

```
[29]: y_pred_editing_method_1 = agglo2_clf_editing_method_1.predict(Xtest)
      acc = accuracy_score(ytest, y_pred_editing_method_1)
      print(f'Accuracy = {acc * 100: .2f}%')
```

```
Accuracy = 89.60%
```

## 2. Method 2 of Data editing: Using repeated two fold cross-validation scheme.

The second data editing procedure is based on the repeated two-fold cross-validation. At each iteration, input data are randomly split into two separate sets. Next, a fold is used to build a GFMMNN model and the second fold is used to make prediction. This process is iterated for each fold so that each fold is used to train and validate the model in turn. After that, the misclassified samples in the validation fold are marked and the random splitting into two separate sets continues for the original training data set. The process can be repeated a fixed number of times (i.e. 100 times) or stopped if the last n iterations have not resulted in new samples being marked as misclassified. Only after the multiple cross-validation process is completed, all the marked samples are removed and the GFMM trained for the edited training set.

```
[30]: # Import the data editing function
      from hbbbrain.utils.data_editing import data_editing_two_fold_cv
```

```
[31]: # Initialise a GFMMNN model using the accelerated agglomerative learning algorithm
      agglo2_clf = AccelAgglomerativeLearningGFMM(theta=theta, gamma=gamma, min_simil=min_
      ↪simil, simil_measure=simil_measure, is_draw=is_draw)
```

```
[32]: # Initialise input paramters of the data editing method
      n_iters=100 # Repeat 100 times
      n_last_iters=5 # If 5 consecutive iterations not remove any sample, stop the loop
      seed=0 # kernel seed for random split data
```

```
[33]: # Doing data editing and get the output dataset
      Xtr_2, ytr_2 = data_editing_two_fold_cv(Xtr, ytr, gfmm_estimator=agglo2_clf, n_iters=n_
      ↪iters, n_last_iters=n_last_iters, seed=seed)
```

```
[34]: print("Number of remaining samples after editing = ", Xtr_2.shape[0])

      Number of remaining samples after editing = 133
```

```
[35]: # Show the remaining training set
      show_2d_samples(Xtr_2, ytr_2, "Input data set after editing by method 2")

      <IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

### Build a GFMMNN model from the remaining training data after editing

```
[36]: agгло2_clf_editing_method_2 = AccelAgglomerativeLearningGFMM(theta=theta, gamma=gamma,
↳ min_simil=min_simil, simil_measure=simil_measure, is_draw=is_draw)
```

```
[37]: agгло2_clf_editing_method_2.fit(Xtr_2, ytr_2)
```

```
[37]: AccelAgglomerativeLearningGFMM(min_simil=0, theta=0.2)
```

### Display decision boundaries among classes if input data are 2-dimensional

```
[38]: agгло2_clf_editing_method_2.draw_hyperbox_and_boundary("The trained GFMM classifier and
↳ its decision boundaries")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[39]: print("Number of hyperboxes = ", agгло2_clf_editing_method_2.get_n_hyperboxes())
```

```
Number of hyperboxes = 13
```

### Make prediction

```
[40]: y_pred_editing_method_2 = agгло2_clf_editing_method_2.predict(Xtest)
acc = accuracy_score(ytest, y_pred_editing_method_2)
print(f'Accuracy = {acc * 100: .2f}%')
```

```
Accuracy = 89.80%
```

## 3. Method 3 of Data editing: Using repeated two fold cross-validation and a probability of misclassification for each sample.

In the second data editing method, all misclassified samples are removed. However, as it has also been indicated in the literature, removing all the misclassified samples can sometimes result in dropping whole regions of a class and essentially overediting the training set. In terms of the GFMM classifier generated for such edited data set, due to overediting, the classification models have turned out to be too simple to accurately capture the decision boundaries. Therefore, the third data editing procedure has been designed using an observation that some input data samples are more consistently used for generation of classifiers during the cross-validation process than others. It also applies to the misclassified samples, whereas some of the samples will be misclassified consistently while others only occasionally. The approach based on the rejection of all the samples which have been misclassified at least ones during the multiple cross-validation does not attempt to take this fact into account in any way. In order to rectify this problem, an approach which will estimate the probability of every single point in the original training data set to be kept during the multiple cross-validation. This probability is simply calculated as the ratio of the number of times an input X has been retained in the validation process of the trained classifier to the total number of repetitions of the two-fold cross validation.

```
[41]: # Import the data editing function
      from hbbbrain.utils.data_editing import data_editing_two_fold_cv_with_probability

[42]: # Initialise a GFMMNN model using the accelerated agglomerative learning algorithm
      agгло2_clf = AccelAgglomerativeLearningGFMM(theta=theta, gamma=gamma, min_simil=min_
      ↪simil, simil_measure=simil_measure, is_draw=is_draw)

[43]: # Initialise input paramters of the data editing method
      n_iters=100 # Repeat 100 times
      min_remained_prob=0.5 # Minimum probability value so that a sample is retained (50% in_
      ↪this case)
      seed=0 # kernel seed for random split data

[44]: # Doing data editing and get the output dataset
      Xtr_3, ytr_3 = data_editing_two_fold_cv_with_probability(Xtr, ytr, gfmm_estimator=agгло2_
      ↪clf, n_iters=n_iters, min_remained_prob=min_remained_prob, seed=seed)

[45]: print("Number of remaining samples after editing = ", Xtr_3.shape[0])
      Number of remaining samples after editing = 217

[46]: # Show the remaining training set
      show_2d_samples(Xtr_3, ytr_3, "Input data set after editing by method 3")

      <IPython.core.display.Javascript object>

      <IPython.core.display.HTML object>
```

### Build a GFMMNN model from the remaining training data after editing

```
[47]: agгло2_clf_editing_method_3 = AccelAgglomerativeLearningGFMM(theta=theta, gamma=gamma,
      ↪min_simil=min_simil, simil_measure=simil_measure, is_draw=is_draw)

[48]: agгло2_clf_editing_method_3.fit(Xtr_3, ytr_3)

[48]: AccelAgglomerativeLearningGFMM(min_simil=0, theta=0.2)
```

### Display decision boundaries among classes if input data are 2-dimensional

```
[49]: agгло2_clf_editing_method_3.draw_hyperbox_and_boundary("The trained GFMM classifier and_
      ↪its decision boundaries")

      <IPython.core.display.Javascript object>

      <IPython.core.display.HTML object>

[50]: print("Number of hyperboxes = ", agгло2_clf_editing_method_3.get_n_hyperboxes())
      Number of hyperboxes = 20
```

### Make prediction

```
[51]: y_pred_editing_method_3 = agglo2_clf_editing_method_3.predict(Xtest)
      acc = accuracy_score(ytest, y_pred_editing_method_3)
      print(f'Accuracy = {acc * 100: .2f}%')
```

```
Accuracy = 89.60%
```

## 2.8.9 Store and load the trained models

### Store and Reload the Trained Models

This example shows how to store a trained hyperbox-based model and reload it to make prediction. This example will use a random hyperboxes model for illustration.

```
[1]: from sklearn.datasets import make_classification
      from hbbbrain.numerical_data.incremental_learner.iol_gfmm import ImprovedOnlineGFMM
      from hbbbrain.numerical_data.ensemble_learner.random_hyperboxes import
      ↪ RandomHyperboxesClassifier
      from hbbbrain.utils.model_storage import store_model, load_model
```

### Generate training data

```
[2]: X, y = make_classification(n_samples=100, n_features=4, n_informative=2, n_redundant=0,
      ↪ random_state=0, shuffle=False)
```

```
[3]: # Normalise data into the range of [0, 1]
      from sklearn.preprocessing import MinMaxScaler
      scaler = MinMaxScaler()
      scaler.fit(X)
      X = scaler.transform(X)
```

### Training a random hyperboxes model

```
[4]: clf = RandomHyperboxesClassifier(base_estimator=ImprovedOnlineGFMM(0.1), n_estimators=10,
      ↪ random_state=0).fit(X, y)
```

### Make prediction

```
[6]: y_pred = clf.predict([[1, 0.6, 0.5, 0.2]])
      print("Predicted class for the input patter [1, 0.6, 0.5, 0.2] is %d"%y_pred[0])
```

```
Predicted class for the input patter [1, 0.6, 0.5, 0.2] is 1
```

**Store the trained model**

```
[7]: store_model(clf, "store_example_model.dummy")
```

**Reload the trained model and make prediction**

```
[8]: clf_load = load_model("store_example_model.dummy")
```

```
[9]: y_pred = clf_load.predict([[1, 0.6, 0.5, 0.2]])  
print("Predicted class for the input patter [1, 0.6, 0.5, 0.2] is %d"%y_pred[0])
```

```
Predicted class for the input patter [1, 0.6, 0.5, 0.2] is 1
```





## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## BIBLIOGRAPHY

- [1] Simpson, P. (1992). Fuzzy min—max neural networks—Part 1: Classification. *IEEE transactions on neural networks*, 3(5), 776-786.
- [1] T.T. Khuat, B. Gabrys, “An in-depth comparison of methods handling mixed-attribute data for general fuzzy min—max neural network”, *Neurocomputing*, vol. 464, pp. 175-202, 2021.
- [2] P.R. Castillo, J. Cardenosa, “Fuzzy min—max neural networks for categorical data: application to missing data imputation”, *Neural Computing and Applications*, vol. 21, pp. 1349–1362, 2012.
- [1] Gabrys, B., & Bargiela, A. (2000). General fuzzy min-max neural network for clustering and classification. *IEEE transactions on neural networks*, 11(3), 769-783.
- [2] Khuat, T. T., & Gabrys, B. (2021). Accelerated learning algorithms of general fuzzy min-max neural network using a novel hyperbox selection rule. *Information Sciences*, 547, 887-909.
- [1] O. N. Al-Sayaydeh, M. F. Mohammed, E. Alhroob, H. Tao, and C. P. Lim, “A refined fuzzy min—max neural network with new learning procedures for pattern classification,” *IEEE Transactions on Fuzzy Systems*, vol. 28, no. 10, pp. 2480-2494, 2019.
- [1] O. N. Al-Sayaydeh, M. F. Mohammed, E. Alhroob, H. Tao, and C. P. Lim, “A refined fuzzy min—max neural network with new learning procedures for pattern classification,” *IEEE Transactions on Fuzzy Systems*, vol. 28, no. 10, pp. 2480-2494, 2019.
- [1] B. Gabrys, “Agglomerative learning algorithms for general fuzzy min-max neural network,” *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 32, no. 1, pp. 67-82, 2002.
- [2] T.T. Khuat and B. Gabrys, “An Online Learning Algorithm for a Neuro-Fuzzy Classifier with Mixed-Attribute Data,” *ArXiv Preprint*, no. arXiv:2009.14670, 2020.
- [1] T.T. Khuat and B. Gabrys, “Accelerated learning algorithms of general fuzzy min-max neural network using a novel hyperbox selection rule,” *Information Sciences*, vol. 547, pp. 887-909, 2021.
- [1] T.T. Khuat and B. Gabrys, “Accelerated learning algorithms of general fuzzy min-max neural network using a novel hyperbox selection rule,” *Information Sciences*, vol. 547, pp. 887-909, 2021.
- [1] T. T. Khuat and B. Gabrys “An Online Learning Algorithm for a Neuro-Fuzzy Classifier with Mixed-Attribute Data”, *ArXiv preprint arXiv:2009.14670*, 2020.
- [1] T. T. Khuat and B. Gabrys “An in-depth comparison of methods handling mixed-attribute data for general fuzzy min—max neural network”, *Neurocomputing*, vol 464, pp. 175-202, 2021.
- [1] T. T. Khuat and B. Gabrys “An in-depth comparison of methods handling mixed-attribute data for general fuzzy min—max neural network”, *Neurocomputing*, vol 464, pp. 175-202, 2021.
- [1] B. Gabrys, “Agglomerative learning algorithms for general fuzzy min-max neural network”, *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 32, no. 1, pp. 67-82, 2002.

- [2] T.T. Khuat and B. Gabrys, "Accelerated learning algorithms of general fuzzy min-max neural network using a novel hyperbox selection rule," *Information Sciences*, vol. 547, pp. 887-909, 2021.
- [1] B. Gabrys, "Agglomerative learning algorithms for general fuzzy min-max neural network", *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 32, no. 1, pp. 67-82, 2002.
- [2] T.T. Khuat and B. Gabrys, "Accelerated learning algorithms of general fuzzy min-max neural network using a novel hyperbox selection rule," *Information Sciences*, vol. 547, pp. 887-909, 2021.
- [1] L. Breiman, "Pasting small votes for classification in large databases and on-line", *Machine Learning*, vol. 36, no. 1, pp. 85-103, 1999.
- [2] L. Breiman, "Bagging predictors", *Machine Learning*, vol. 24, no. 2, pp. 123-140, 1996.
- [3] B. Gabrys, "Combining neuro-fuzzy classifiers for improved generalisation and reliability", in *Proceedings of the 2002 International Joint Conference on Neural Networks*, vol. 3, pp. 2410-2415, 2002.
- [1] L. Breiman, "Pasting small votes for classification in large databases and on-line", *Machine Learning*, vol. 36, no. 1, pp. 85-103, 1999.
- [2] L. Breiman, "Bagging predictors", *Machine Learning*, vol. 24, no. 2, pp. 123-140, 1996.
- [3] B. Gabrys, "Combining neuro-fuzzy classifiers for improved generalisation and reliability", in *Proceedings of the 2002 International Joint Conference on Neural Networks*, vol. 3, pp. 2410-2415, 2002.
- [1] L. Breiman, "Pasting small votes for classification in large databases and on-line", *Machine Learning*, vol. 36, no. 1, pp. 85-103, 1999.
- [2] L. Breiman, "Bagging predictors", *Machine Learning*, vol. 24, no. 2, pp. 123-140, 1996.
- [3] B. Gabrys, "Combining neuro-fuzzy classifiers for improved generalisation and reliability", in *Proceedings of the 2002 International Joint Conference on Neural Networks*, vol. 3, pp. 2410-2415, 2002.
- [1] L. Breiman, "Pasting small votes for classification in large databases and on-line", *Machine Learning*, vol. 36, no. 1, pp. 85-103, 1999.
- [2] L. Breiman, "Bagging predictors", *Machine Learning*, vol. 24, no. 2, pp. 123-140, 1996.
- [3] B. Gabrys, "Combining neuro-fuzzy classifiers for improved generalisation and reliability", in *Proceedings of the 2002 International Joint Conference on Neural Networks*, vol. 3, pp. 2410-2415, 2002.
- [1] T. T. Khuat and B. Gabrys, "Random Hyperboxes", *IEEE Transactions on Neural Networks and Learning Systems*, 2021.
- [1] T. T. Khuat and B. Gabrys "Random Hyperboxes", *IEEE Transactions on Neural Networks and Learning Systems*, 2021.
- [1] B. Gabrys and A. Bargiela, "General fuzzy min-max neural network for clustering and classification," *IEEE Transactions on Neural Networks*, vol. 11, no. 3, pp. 769-783, 2000.
- [2] T.T. Khuat and B. Gabrys, "Accelerated learning algorithms of general fuzzy min-max neural network using a novel hyperbox selection rule," *Information Sciences*, vol. 547, pp. 887-909, 2021.
- [3] B. Gabrys, "Agglomerative learning algorithms for general fuzzy min-max neural network", *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology*, vol. 32, no. 1, pp. 67-82, 2002.
- [1] T.T. Khuat, F. Chen, and B. Gabrys, "An improved online learning algorithm for general fuzzy min-max neural network," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pp. 1-9, 2020.
- [2] T.T. Khuat and B. Gabrys, "Accelerated learning algorithms of general fuzzy min-max neural network using a novel hyperbox selection rule," *Information Sciences*, vol. 547, pp. 887-909, 2021.
- [1] P. Simpson, "Fuzzy min—max neural networks—Part 1: Classification," *IEEE Transactions on Neural Networks*, vol. 3, no. 5, pp. 776-786, 1992.

- [1] M. Mohammed and C. P. Lim, "An enhanced fuzzy min-max neural network for pattern classification," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 3, pp. 417-429, 2014.
- [1] M. Mohammed and C. P. Lim, "Improving the Fuzzy Min-Max neural network with a k-nearest hyperbox expansion rule for pattern classification," *Applied Soft Computing*, vol. 52, pp. 135-145, 2017.
- [1] O. N. Al-Sayaydeh, M. F. Mohammed, E. Alhroob, H. Tao, and C. P. Lim, "A refined fuzzy min-max neural network with new learning procedures for pattern classification," *IEEE Transactions on Fuzzy Systems*, vol. 28, no. 10, pp. 2480-2494, 2019.
- [1] T.T. Khuat, F. Chen, and B. Gabrys, "An Effective Multiresolution Hierarchical Granular Representation Based Classifier Using General Fuzzy Min-Max Neural Network," *IEEE Transactions on Fuzzy Systems*, vol. 29, no. 2, pp. 427-441, 2021.



## PYTHON MODULE INDEX

### h

hbbrain.base.base\_ensemble, 67  
hbbrain.base.base\_estimator, 55  
hbbrain.base.base\_fmnn\_estimator, 63  
hbbrain.base.base\_gfmm\_estimator, 58  
hbbrain.mixed\_data.eiol\_gfmm, 68  
hbbrain.mixed\_data.freq\_cat\_onln\_gfmm, 76  
hbbrain.mixed\_data.onehot\_onln\_gfmm, 84  
hbbrain.numerical\_data.batch\_learner.accel\_agglo\_gfmm, 95  
hbbrain.numerical\_data.batch\_learner.agglo\_gfmm, 91  
hbbrain.numerical\_data.ensemble\_learner.base\_bagging, 99  
hbbrain.numerical\_data.ensemble\_learner.base\_cross\_val\_bagging, 101  
hbbrain.numerical\_data.ensemble\_learner.cross\_val\_random\_hyperboxes, 126  
hbbrain.numerical\_data.ensemble\_learner.decision\_comb\_bagging, 103  
hbbrain.numerical\_data.ensemble\_learner.decision\_comb\_cross\_val\_bagging, 106  
hbbrain.numerical\_data.ensemble\_learner.model\_comb\_bagging, 110  
hbbrain.numerical\_data.ensemble\_learner.model\_comb\_cross\_val\_bagging, 116  
hbbrain.numerical\_data.ensemble\_learner.random\_hyperboxes, 122  
hbbrain.numerical\_data.incremental\_learner.efmnn, 142  
hbbrain.numerical\_data.incremental\_learner.fmnn, 140  
hbbrain.numerical\_data.incremental\_learner.iol\_gfmm, 135  
hbbrain.numerical\_data.incremental\_learner.knefmnn, 144  
hbbrain.numerical\_data.incremental\_learner.onln\_gfmm, 131  
hbbrain.numerical\_data.incremental\_learner.rfmnn, 147  
hbbrain.numerical\_data.multigranular\_learner.multi\_resolution\_gfmm, 151  
hbbrain.utils.adjust\_hyperbox, 37  
hbbrain.utils.dist\_metrics, 52  
hbbrain.utils.drawing\_func, 49  
hbbrain.utils.matrix\_transformation, 48  
hbbrain.utils.membership\_calc, 25  
hbbrain.utils.model\_storage, 54





## INDEX

### A

**AccelAgglomerativeLearningGFMM** (class in *hb-brain.numerical\_data.batch\_learner.accel\_agglo\_gfmm*), 95

**AgglomerativeLearningGFMM** (class in *hb-brain.numerical\_data.batch\_learner.agglo\_gfmm*), 91

**asym\_similarity\_val\_one\_many\_hyperboxes()** (in module *hbbrain.utils.membership\_calc*), 25

### B

**BaseBagging** (class in *hb-brain.numerical\_data.ensemble\_learner.base\_bagging*), 99

**BaseCrossValBagging** (class in *hb-brain.numerical\_data.ensemble\_learner.base\_cross\_val\_bagging*), 101

**BaseEnsemble** (class in *hbbrain.base.base\_ensemble*), 67

**BaseFMNNClassifier** (class in *hb-brain.base.base\_fmnn\_estimator*), 63

**BaseGFMMClassifier** (class in *hb-brain.base.base\_gfmm\_estimator*), 58

**BaseHyperboxClassifier** (class in *hb-brain.base.base\_estimator*), 55

**bitwise\_membership()** (in module *hb-brain.utils.membership\_calc*), 25

### C

**compute\_increasing\_entropy()** (*hb-brain.mixed\_data.eiol\_gfmm.ExtendedImprovedOnlineGFMM* method), 70

**compute\_similarity\_among\_categorical\_values()** (in module *hb-brain.mixed\_data.freq\_cat\_onln\_gfmm*), 81

**convert\_format\_missing\_input\_zero\_one()** (in module *hbbrain.base.base\_gfmm\_estimator*), 61

**convert\_granular\_theta\_to\_level()** (in module *hb-*

*brain.numerical\_data.multigranular\_learner.multi\_resolution\_gfmm*), 160

**CrossValRandomHyperboxesClassifier** (class in *hb-brain.numerical\_data.ensemble\_learner.cross\_val\_random\_hyperboxes*), 126

### D

**DecisionCombinationBagging** (class in *hb-brain.numerical\_data.ensemble\_learner.decision\_comb\_bagging*), 103

**DecisionCombinationCrossValBagging** (class in *hb-brain.numerical\_data.ensemble\_learner.decision\_comb\_cross\_val\_bagging*), 106

**delay()** (*hbbrain.base.base\_estimator.BaseHyperboxClassifier* method), 56

**draw\_2D\_hyperbox\_and\_boundary\_granular\_level()** (*hbbrain.numerical\_data.multigranular\_learner.multi\_resolution\_gfmm* method), 153

**draw\_2D\_hyperbox\_and\_boundary\_partitions()** (*hbbrain.numerical\_data.multigranular\_learner.multi\_resolution\_gfmm* method), 154

**draw\_box()** (in module *hbbrain.utils.drawing\_func*), 49

**draw\_box\_parallel\_coordinate()** (in module *hb-brain.utils.drawing\_func*), 50

**draw\_decision\_boundary\_2D()** (in module *hb-brain.utils.drawing\_func*), 50

**draw\_hyperbox\_and\_boundary()** (*hb-brain.base.base\_estimator.BaseHyperboxClassifier* method), 56

### E

**EFMNNClassifier** (class in *hb-brain.numerical\_data.incremental\_learner.efmnn*), 142

**estimators\_samples\_** (*hb-brain.numerical\_data.ensemble\_learner.base\_bagging.BaseBagging* property), 100

**estimators\_samples\_** (*hb-brain.numerical\_data.ensemble\_learner.base\_cross\_val\_bagging* property), 101

**estimators\_samples\_** (*hb-brain.numerical\_data.ensemble\_learner.cross\_val\_random\_hyperboxes* property), 126

property), 129

estimators\_samples\_ (hb- FreqCatOnlineGfmm (class in hb-  
brain.numerical\_data.ensemble\_learner.random\_hyperboxes.RandomHyperboxesClassifier),  
property), 124

ExtendedImprovedOnlineGfmm (class in hb-  
brain.mixed\_data.eiol\_gfmm), 68

## F

f\_sim\_freq\_cat\_features() (in module hb-  
brain.utils.membership\_calc), 26

fit() (hbbrain.base.base\_estimator.BaseHyperboxClassifier  
method), 56

fit() (hbbrain.mixed\_data.eiol\_gfmm.ExtendedImprovedOnlineGfmm  
method), 71

fit() (hbbrain.mixed\_data.freq\_cat\_onln\_gfmm.FreqCatOnlineGfmm  
method), 78

fit() (hbbrain.mixed\_data.onehot\_onln\_gfmm.OneHotOnlineGfmm  
method), 86

fit() (hbbrain.numerical\_data.batch\_learner.accel\_agglo\_gfmm.AccelAgglomerativeGfmm  
method), 97

fit() (hbbrain.numerical\_data.batch\_learner.agglo\_gfmm.AggglomerativeGfmm  
method), 93

fit() (hbbrain.numerical\_data.ensemble\_learner.base\_bagging.BaseBagging  
method), 100

fit() (hbbrain.numerical\_data.ensemble\_learner.base\_cross\_val\_bagging.BaseCrossValBagging  
method), 102

fit() (hbbrain.numerical\_data.ensemble\_learner.cross\_val\_random\_hyperboxes.CrossValRandomHyperboxesClassifier  
method), 129

fit() (hbbrain.numerical\_data.ensemble\_learner.decision\_comb\_bagging.DecisionCombinationBagging  
method), 105

fit() (hbbrain.numerical\_data.ensemble\_learner.decision\_comb\_cross\_val\_bagging.DecisionCombinationCrossValBagging  
method), 109

fit() (hbbrain.numerical\_data.ensemble\_learner.model\_comb\_bagging.ModelCombinationBagging  
method), 112

fit() (hbbrain.numerical\_data.ensemble\_learner.model\_comb\_cross\_val\_bagging.ModelCombinationCrossValBagging  
method), 119

fit() (hbbrain.numerical\_data.ensemble\_learner.random\_hyperboxes.RandomHyperboxesClassifier  
method), 125

fit() (hbbrain.numerical\_data.incremental\_learner.efmnn.EFMNNClassifier  
method), 143

fit() (hbbrain.numerical\_data.incremental\_learner.fmn.FMNNClassifier  
method), 141

fit() (hbbrain.numerical\_data.incremental\_learner.iol\_gfmm.IOLGfmm  
method), 137

fit() (hbbrain.numerical\_data.incremental\_learner.knefmnn.KNEFMNNClassifier  
method), 146

fit() (hbbrain.numerical\_data.incremental\_learner.onln\_gfmm.OnlineGfmm  
method), 134

fit() (hbbrain.numerical\_data.incremental\_learner.rfmnn.RFMNNClassifier  
method), 148

fit() (hbbrain.numerical\_data.multigranular\_learner.multi\_resolution\_gfmm.MultiResolutionGranularGfmm  
method), 154

FMNNClassifier (class in hb-  
brain.numerical\_data.incremental\_learner.fmn),

## G

generate\_grid\_decision\_boundary\_2D() (in mod-  
ule hbbrain.utils.drawing\_func), 51

get\_cmap() (in module hbbrain.utils.drawing\_func), 51

get\_membership\_extended\_iol\_gfmm\_all\_classes()  
(in module hbbrain.utils.membership\_calc), 26

get\_membership\_fmn\_all\_classes() (in module  
hbbrain.utils.membership\_calc), 27

get\_membership\_free\_range\_gfmm\_all\_classes()  
(in module hbbrain.utils.membership\_calc), 28

get\_membership\_freq\_cat\_gfmm\_all\_classes()  
(in module hbbrain.utils.membership\_calc), 29

get\_membership\_gfmm\_all\_classes() (in module  
hbbrain.utils.membership\_calc), 30

get\_membership\_onehot\_gfmm\_all\_classes() (in  
module hbbrain.utils.membership\_calc), 30

get\_n\_hyperboxes() (hb-  
brain.base.base\_estimator.BaseHyperboxClassifier  
method), 57

get\_n\_hyperboxes() (hb-  
brain.mixed\_data.eiol\_gfmm.ExtendedImprovedOnlineGfmm  
method), 71

get\_n\_hyperboxes() (hb-  
brain.mixed\_data.freq\_cat\_onln\_gfmm.FreqCatOnlineGfmm  
method), 79

get\_n\_hyperboxes() (hb-  
brain.mixed\_data.onehot\_onln\_gfmm.OneHotOnlineGfmm  
method), 87

get\_n\_hyperboxes() (hb-  
brain.numerical\_data.ensemble\_learner.base\_bagging.BaseBagging  
method), 100

get\_n\_hyperboxes() (hb-  
brain.numerical\_data.ensemble\_learner.base\_cross\_val\_bagging.BaseCrossValBagging  
method), 102

get\_n\_hyperboxes() (hb-  
brain.numerical\_data.ensemble\_learner.cross\_val\_random\_hyperboxes.CrossValRandomHyperboxesClassifier  
method), 129

get\_n\_hyperboxes() (hb-  
brain.numerical\_data.ensemble\_learner.decision\_comb\_bagging.DecisionCombinationBagging  
method), 105

get\_n\_hyperboxes() (hb-  
brain.numerical\_data.ensemble\_learner.decision\_comb\_cross\_val\_bagging.DecisionCombinationCrossValBagging  
method), 109

get\_n\_hyperboxes() (hb-  
brain.numerical\_data.ensemble\_learner.model\_comb\_bagging.ModelCombinationBagging  
method), 112

get\_n\_hyperboxes() (hb-  
brain.numerical\_data.ensemble\_learner.model\_comb\_cross\_val\_bagging.ModelCombinationCrossValBagging  
method), 119

get\_n\_hyperboxes() (hb-  
brain.numerical\_data.ensemble\_learner.random\_hyperboxes.RandomHyperboxesClassifier  
method), 125

get\_n\_hyperboxes() (hb-  
brain.numerical\_data.ensemble\_learner.cross\_val\_random\_hyperboxes.CrossValRandomHyperboxesClassifier  
method), 130

get\_n\_hyperboxes() (hb-  
brain.numerical\_data.ensemble\_learner.random\_hyperboxes.RandomHyperboxesClassifier  
method), 125

get\_n\_hyperboxes\_at\_partition() (hb-  
brain.numerical\_data.multigranular\_learner.multi\_resolution\_gfmm.MultiResolutionGranularGfmm  
method), 155

get\_n\_hyperboxes\_comb\_model() (hb-  
brain.numerical\_data.ensemble\_learner.model\_comb\_bagging.ModelCombinationBagging  
method), 113

`get_n_hyperboxes_comb_model()` (hb- module, 68  
`brain.numerical_data.ensemble_learner.model_comb_bagging.ModelCombBagging`  
`method`), 120 module, 76  
`get_sample_explanation()` (hb- hbbbrain.mixed\_data.onehot\_onln\_gfmm  
`brain.base.base_fmnn_estimator.BaseFMNNClassifier` module, 84  
`method`), 64 hbbbrain.numerical\_data.batch\_learner.accel\_agglo\_gfmm  
`get_sample_explanation()` (hb- module, 95  
`brain.mixed_data.eiol_gfmm.ExtendedImprovedOnlineGfmm` hbbbrain.numerical\_data.batch\_learner.agglo\_gfmm  
`method`), 71 module, 91  
`get_sample_explanation()` (hb- hbbbrain.numerical\_data.ensemble\_learner.base\_bagging  
`brain.mixed_data.freq_cat_onln_gfmm.FreqCatOnlineGfmm` module, 99  
`method`), 79 hbbbrain.numerical\_data.ensemble\_learner.base\_cross\_val\_bag  
`get_sample_explanation()` (hb- module, 101  
`brain.mixed_data.onehot_onln_gfmm.OneHotOnlineGfmm` hbbbrain.numerical\_data.ensemble\_learner.cross\_val\_random\_h  
`method`), 87 module, 126  
`get_sample_explanation()` (hb- hbbbrain.numerical\_data.ensemble\_learner.decision\_comb\_bagging  
`brain.numerical_data.batch_learner.accel_agglo_gfmm.AccelerativeLearningGfmm` module, 103  
`method`), 97 hbbbrain.numerical\_data.ensemble\_learner.decision\_comb\_cross  
`get_sample_explanation()` (hb- module, 106  
`brain.numerical_data.batch_learner.agglo_gfmm.AgglomerativeLearningGfmm` hbbbrain.numerical\_data.ensemble\_learner.model\_comb\_bagging  
`method`), 93 module, 110  
`get_sample_explanation()` (hb- hbbbrain.numerical\_data.ensemble\_learner.model\_comb\_cross\_val  
`brain.numerical_data.incremental_learner.iol_gfmm.ImprovedOnlineGfmm` module, 110  
`method`), 138 hbbbrain.numerical\_data.ensemble\_learner.random\_hyperboxes  
`get_sample_explanation()` (hb- module, 122  
`brain.numerical_data.incremental_learner.onln_gfmm.OnlineGfmm` module, 142  
`method`), 134 hbbbrain.numerical\_data.incremental\_learner.efmnn  
`get_sample_explanation()` (hb- hbbbrain.numerical\_data.incremental\_learner.fmnn  
`brain.numerical_data.incremental_learner.rfmnn.RFMN` module, 140  
`method`), 149 hbbbrain.numerical\_data.incremental\_learner.iol\_gfmm  
`get_sample_explanation_granular_level()` (hb- module, 135  
`brain.numerical_data.multigranular_learner.multigranular_fmnn.MultigranularFMN` hbbbrain.numerical\_data.incremental\_learner.knefmnn  
`method`), 155 module, 144  
`granular_learning_phase_1()` (hb- hbbbrain.numerical\_data.incremental\_learner.onln\_gfmm  
`brain.numerical_data.multigranular_learner.multi_resolution_fmnn.MultiGranularGfmm` module, 146  
`method`), 156 hbbbrain.numerical\_data.incremental\_learner.rfmnn  
`granular_learning_phase_2()` (hb- module, 147  
`brain.numerical_data.multigranular_learner.multigranular_fmnn.MultigranularFMN` hbbbrain.numerical\_data.incremental\_learner.multi\_resolution\_fmnn  
`method`), 157 module, 151  
  
H hbbbrain.utils.adjust\_hyperbox  
module, 37  
`hashing()` (in module hb- hbbbrain.utils.dist\_metrics  
`brain.utils.matrix_transformation`), 48 module, 52  
`hashing_mat()` (in module hb- hbbbrain.utils.drawing\_func  
`brain.utils.matrix_transformation`), 48 module, 49  
hbbbrain.base.base\_ensemble hbbbrain.utils.matrix\_transformation  
module, 67 module, 48  
hbbbrain.base.base\_estimator hbbbrain.utils.membership\_calc  
module, 55 module, 25  
hbbbrain.base.base\_fmnn\_estimator hbbbrain.utils.model\_storage  
module, 63 module, 54  
hbbbrain.base.base\_gfmm\_estimator hyperbox\_contraction\_efmnn() (in module hb-  
module, 58 `brain.utils.adjust_hyperbox`), 37  
hbbbrain.mixed\_data.eiol\_gfmm

hyperbox\_contraction\_fmnn() (in module hb-  
brain.utils.adjust\_hyperbox), 38  
hyperbox\_contraction\_freq\_cat\_gfmm() (in mod-  
ule hbbrain.utils.adjust\_hyperbox), 39  
hyperbox\_contraction\_rfmnn() (in module hb-  
brain.utils.adjust\_hyperbox), 39  
hyperbox\_overlap\_test\_efmnn() (in module hb-  
brain.utils.adjust\_hyperbox), 40  
hyperbox\_overlap\_test\_fmnn() (in module hb-  
brain.utils.adjust\_hyperbox), 40  
hyperbox\_overlap\_test\_freq\_cat\_gfmm() (in mod-  
ule hbbrain.utils.adjust\_hyperbox), 41

## I

ImprovedOnlineGFMM (class in hb-  
brain.numerical\_data.incremental\_learner.iol\_gfmm),  
135  
impute\_missing\_categorical\_features() (in mod-  
ule hbbrain.mixed\_data.eiol\_gfmm), 73  
impute\_missing\_value\_cat\_feature() (in module  
hbbrain.mixed\_data.onehot\_onln\_gfmm), 89  
initialise\_canvas\_graph() (hb-  
brain.base.base\_estimator.BaseHyperboxClassifier  
method), 57  
is\_contain\_missing\_value() (in module hb-  
brain.base.base\_gfmm\_estimator), 61  
is\_overlap\_cat\_features\_one\_by\_one() (in mod-  
ule hbbrain.utils.adjust\_hyperbox), 42  
is\_overlap\_cat\_features\_one\_vs\_many() (in mod-  
ule hbbrain.utils.adjust\_hyperbox), 42  
is\_overlap\_diff\_labels\_num\_data\_rfmnn() (in  
module hbbrain.utils.adjust\_hyperbox), 43  
is\_overlap\_one\_many\_diff\_label\_hyperboxes\_mixed\_data\_general() (in module hbbrain.utils.adjust\_hyperbox), 44  
is\_overlap\_one\_many\_diff\_label\_hyperboxes\_num\_data\_general() (in module hbbrain.utils.adjust\_hyperbox), 44  
is\_overlap\_one\_many\_hyperboxes\_num\_data\_general() (in module hbbrain.utils.adjust\_hyperbox), 45  
is\_satisfied\_cat\_expansion\_conds() (hb-  
brain.mixed\_data.freq\_cat\_onln\_gfmm.FreqCatOnlineGFMM  
method), 80  
is\_satisfied\_cat\_expansion\_conds() (hb-  
brain.mixed\_data.onehot\_onln\_gfmm.OneHotOnlineGFMM  
method), 87  
is\_two\_hyperboxes\_overlap\_num\_data\_free\_range\_general() (in module hbbrain.utils.adjust\_hyperbox), 45  
is\_two\_hyperboxes\_overlap\_num\_data\_general() (in module hbbrain.utils.adjust\_hyperbox), 46

## K

KNEFMNNClassifier (class in hb-  
brain.numerical\_data.incremental\_learner.knefmnn),  
144

## L

load\_model() (in module hbbrain.utils.model\_storage),  
54  
load\_multi\_models() (in module hb-  
brain.utils.model\_storage), 54

## M

manhattan\_distance() (in module hb-  
brain.utils.dist\_metrics), 52  
manhattan\_distance\_with\_missing\_val() (in mod-  
ule hbbrain.utils.dist\_metrics), 52  
manhattan\_distance\_with\_missing\_val\_free\_range() (in module hbbrain.utils.dist\_metrics), 53  
membership\_cat\_feature\_eiol\_gfmm() (in module  
hbbrain.utils.membership\_calc), 31  
membership\_func\_extended\_iol\_gfmm() (in module  
hbbrain.utils.membership\_calc), 32  
membership\_func\_fmnn() (in module hb-  
brain.utils.membership\_calc), 33  
membership\_func\_free\_range\_gfmm() (in module  
hbbrain.utils.membership\_calc), 34  
membership\_func\_freq\_cat\_gfmm() (in module hb-  
brain.utils.membership\_calc), 34  
membership\_func\_gfmm() (in module hb-  
brain.utils.membership\_calc), 35  
membership\_func\_onehot\_gfmm() (in module hb-  
brain.utils.membership\_calc), 36  
membership\_function\_freq\_cat() (in module hb-  
brain.utils.membership\_calc), 37  
ModelCombinationBagging (class in hb-  
brain.numerical\_data.ensemble\_learner.model\_comb\_bagging),  
110  
ModelCombinationCrossValBagging (class in hb-  
brain.numerical\_data.ensemble\_learner.model\_comb\_cross\_val-  
bagging), 116  
module  
hbbrain.base.base\_ensemble, 67  
hbbrain.base.base\_estimator, 55  
hbbrain.base.base\_fmnn\_estimator, 63  
hbbrain.base.base\_gfmm\_estimator, 58  
hbbrain.mixed\_data.eiol\_gfmm, 68  
hbbrain.mixed\_data.freq\_cat\_onln\_gfmm, 76  
hbbrain.mixed\_data.onehot\_onln\_gfmm, 84  
hbbrain.numerical\_data.batch\_learner.accel\_agglo\_gfmm,  
95  
hbbrain.numerical\_data.batch\_learner.agglo\_gfmm,  
91  
hbbrain.numerical\_data.ensemble\_learner.base\_bagging,  
99  
hbbrain.numerical\_data.ensemble\_learner.base\_cross\_val-  
bagging, 101  
hbbrain.numerical\_data.ensemble\_learner.cross\_val\_rand-  
om, 126

hbbrain.numerical\_data.ensemble\_learner.decision\_comb\_bagging,  
 103 `predict()` (hbbrain.base.base\_fmnn\_estimator.BaseFMNNClassifier  
 hbbrain.numerical\_data.ensemble\_learner.decision\_comb\_cross\_val\_bagging,  
 106 `method`), 65  
 hbbrain.numerical\_data.ensemble\_learner.model\_comb\_bagging,  
 110 `predict()` (hbbrain.base.base\_gfmm\_estimator.BaseGFMMClassifier  
 hbbrain.numerical\_data.ensemble\_learner.model\_comb\_cross\_val\_bagging,  
 116 `method`), 72  
 hbbrain.numerical\_data.ensemble\_learner.random\_hyperboxes,  
 122 `predict()` (hbbrain.mixed\_data.freq\_cat\_onln\_gfmm.FreqCatOnlineGFMM  
 hbbrain.numerical\_data.incremental\_learner.efmnn, 88  
 142 `predict()` (hbbrain.numerical\_data.batch\_learner.accel\_agglo\_gfmm.AccelAgglo  
 hbbrain.numerical\_data.incremental\_learner.fmnn, `method`), 98  
 140 `predict()` (hbbrain.numerical\_data.batch\_learner.agglo\_gfmm.Agglo  
 hbbrain.numerical\_data.incremental\_learner.iol\_gfmm, `method`), 94  
 135 `predict()` (hbbrain.numerical\_data.ensemble\_learner.cross\_val\_random  
 hbbrain.numerical\_data.incremental\_learner.knefmnn, `method`), 130  
 144 `predict()` (hbbrain.numerical\_data.ensemble\_learner.decision\_comb\_bag  
 hbbrain.numerical\_data.incremental\_learner.onln\_gfmm, `method`), 105  
 131 `predict()` (hbbrain.numerical\_data.ensemble\_learner.decision\_comb\_cro  
 hbbrain.numerical\_data.incremental\_learner.rfmnn, `method`), 109  
 147 `predict()` (hbbrain.numerical\_data.ensemble\_learner.model\_comb\_bagg  
 hbbrain.numerical\_data.multigranular\_learner.multi\_resolution\_gfmm,  
 151 `method`), 115  
 hbbrain.utils.adjust\_hyperbox, 37  
 hbbrain.utils.dist\_metrics, 52  
 hbbrain.utils.drawing\_func, 49  
 hbbrain.utils.matrix\_transformation, 48  
 hbbrain.utils.membership\_calc, 25  
 hbbrain.utils.model\_storage, 54  
 MultiGranularGFMM (class in hb-  
 brain.numerical\_data.multigranular\_learner.multi\_resolution\_gfmm),  
 151 `predict()` (hbbrain.numerical\_data.multigranular\_learner.multi\_resolutio  
 N  
 n\_cat\_features\_containing\_bit\_one() (in module  
 hbbrain.utils.membership\_calc), 37  
 O  
 one\_hot\_encoding\_cat\_feature() (in module hb-  
 brain.mixed\_data.onehot\_onln\_gfmm), 89  
 OneHotOnlineGFMM (class in hb-  
 brain.mixed\_data.onehot\_onln\_gfmm), 84  
 OnlineGFMM (class in hb-  
 brain.numerical\_data.incremental\_learner.onln\_gfmm),  
 131 `predict_at_partitions()` (hb-  
 brain.numerical\_data.multigranular\_learner.multi\_resolution\_gfmm,  
`method`), 157  
 ordinal\_encode\_categorical\_features() (in mod-  
 ule hbbrain.mixed\_data.freq\_cat\_onln\_gfmm),  
 82 `predict_freq_cat_feature_manhanttan()` (in mod-  
 ule hbbrain.mixed\_data.freq\_cat\_onln\_gfmm),  
 82  
 overlap\_resolving\_num\_data() (in module hb-  
 brain.utils.adjust\_hyperbox), 47  
 overlap\_resolving\_num\_data\_free\_range() (in  
 module hbbrain.utils.adjust\_hyperbox), 47 `predict_onehot_cat_feature_manhanttan()`  
 (in module hb-  
 brain.mixed\_data.onehot\_onln\_gfmm), 90  
`predict_proba()` (hb-  
 brain.base.base\_fmnn\_estimator.BaseFMNNClassifier  
`method`), 65  
`predict_proba()` (hb-  
 brain.base.base\_gfmm\_estimator.BaseGFMMClassifier  
`method`), 60  
`predict_proba()` (hb-  
 brain.mixed\_data.eiol\_gfmm.ExtendedImprovedOnlineGFMM  
`method`), 72  
`predict_proba()` (hb-  
 brain.mixed\_data.freq\_cat\_onln\_gfmm.FreqCatOnlineGFMM  
`method`), 80



`predict_proba()` (hb- `predict_with_membership()` (hb-  
`brain.mixed_data.onehot_onln_gfmm.OneHotOnlineGFMM`  
`method`), 88 `method`), 73  
`predict_proba()` (hb- `predict_with_membership()` (hb-  
`brain.numerical_data.ensemble_learner.cross_val_random_hyperbox`  
`method`), 130 `method`), 81  
`predict_proba()` (hb- `predict_with_membership()` (hb-  
`brain.numerical_data.ensemble_learner.decision_comb_bagging`  
`method`), 105 `method`), 88  
`predict_proba()` (hb- `predict_with_membership()` (hb-  
`brain.numerical_data.ensemble_learner.decision_comb_cross_val`  
`method`), 109 `method`), 130  
`predict_proba()` (hb- `predict_with_membership()` (hb-  
`brain.numerical_data.ensemble_learner.model_comb_bagging`  
`method`), 114 `method`), 106  
`predict_proba()` (hb- `predict_with_membership()` (hb-  
`brain.numerical_data.ensemble_learner.model_comb_cross_val`  
`method`), 120 `method`), 110  
`predict_proba()` (hb- `predict_with_membership()` (hb-  
`brain.numerical_data.ensemble_learner.random_hyperboxes`  
`method`), 125 `method`), 114  
`predict_proba()` (hb- `predict_with_membership()` (hb-  
`brain.numerical_data.multigranular_learner.multi_resolution`  
`method`), 158 `method`), 121  
`predict_proba_all_base_learners()` (hb- `predict_with_membership()` (hb-  
`brain.numerical_data.ensemble_learner.model_comb_bagging`  
`method`), 114 `method`), 125  
`predict_proba_all_base_learners()` (hb- `predict_with_membership()` (hb-  
`brain.numerical_data.ensemble_learner.model_comb_cross_val`  
`method`), 120 `method`), 158  
`predict_rfmmn()` (in module hb- `predict_with_membership()` (in module hb-  
`brain.numerical_data.incremental_learner.rfmmn`), `brain.numerical_data.multigranular_learner.multi_resolution_gfmm`  
150 161  
`predict_voting()` (hb- `predict_with_membership_all_base_learners()`  
`brain.numerical_data.ensemble_learner.model_comb_bagging`  
`method`), 114 `method`), 115  
`predict_voting()` (hb- `predict_with_membership_all_base_learners()`  
`brain.numerical_data.ensemble_learner.model_comb_cross_val`  
`method`), 121 `method`), 121  
`predict_with_centroids()` (in module hb- `predict_with_probability()` (in module hb-  
`brain.numerical_data.multigranular_learner.multi_resolution_gfmm`  
160 `brain.base.base_gfmm_estimator`), 62  
`predict_with_manhattan()` (in module hb- `predict_with_probability_mixed_data()` (in mod-  
`brain.base.base_gfmm_estimator`), 61 `ule hb`  
`brain.numerical_data.eiol_gfmm`), 74  
`predict_with_manhattan_fmnn()` (in module hb- **R**  
`brain.base.base_fmnn_estimator`), 66  
`predict_with_manhattan_mixed_data()` (in module `RandomHyperboxesClassifier` (class in hb-  
`hb`  
`brain.mixed_data.eiol_gfmm`), 74 `brain.numerical_data.ensemble_learner.random_hyperboxes`),  
122  
`predict_with_membership()` (hb- `remove_contained_hyperboxes()` (in module hb-  
`brain.base.base_fmnn_estimator.BaseFMNNClassifier` `brain.numerical_data.multigranular_learner.multi_resolution_gfmm`  
`method`), 65 162  
`predict_with_membership()` (hb- `rfmmn_distance()` (in module hb-  
`brain.base.base_gfmm_estimator.BaseGFMMClassifier` `brain.utils.dist_metrics`), 54  
`method`), 61

RFMNNClassifier (class in hb- brain.numerical\_data.ensemble\_learner.random\_hyperboxes.Ran  
 brain.numerical\_data.incremental\_learner.rfmnn), method), 126  
 147 split\_matrix() (in module hb-  
 brain.utils.matrix\_transformation), 49

**S** store\_model() (in module hb-  
 brain.utils.model\_storage), 54

show\_sample\_explanation() (hb-  
 brain.base.base\_estimator.BaseHyperboxClassifier  
 method), 57

simple\_pruning() (hb-  
 brain.base.base\_fmnn\_estimator.BaseFMNNClassifier  
 method), 66

simple\_pruning() (hb-  
 brain.mixed\_data.eiol\_gfmm.ExtendedImprovedOnlineGFMM  
 method), 73

simple\_pruning() (hb-  
 brain.mixed\_data.freq\_cat\_onln\_gfmm.FreqCatOnlineGFMM  
 method), 81

simple\_pruning() (hb-  
 brain.mixed\_data.onehot\_onln\_gfmm.OneHotOnlineGFMM  
 method), 89

simple\_pruning() (hb-  
 brain.numerical\_data.batch\_learner.accel\_agglo\_gfmm.AccelAgglomerativeLearningGFMM  
 method), 98

simple\_pruning() (hb-  
 brain.numerical\_data.batch\_learner.agglo\_gfmm.AgglomerativeLearningGFMM  
 method), 94

simple\_pruning() (hb-  
 brain.numerical\_data.ensemble\_learner.model\_comb\_bagging.ModelCombinationBagging  
 method), 115

simple\_pruning() (hb-  
 brain.numerical\_data.ensemble\_learner.model\_comb\_cross\_val\_bagging.ModelCombinationCrossValBagging  
 method), 121

simple\_pruning() (hb-  
 brain.numerical\_data.incremental\_learner.iol\_gfmm.ImprovedOnlineGFMM  
 method), 139

simple\_pruning() (hb-  
 brain.numerical\_data.incremental\_learner.onln\_gfmm.OnlineGFMM  
 method), 135

simple\_pruning() (hb-  
 brain.numerical\_data.incremental\_learner.rfmnn.RFMNNClassifier  
 method), 149

simple\_pruning() (hb-  
 brain.numerical\_data.multigranular\_learner.multi\_resolution\_gfmm.MultiGranularGFMM  
 method), 159

simple\_pruning\_base\_estimators() (hb-  
 brain.numerical\_data.ensemble\_learner.base\_bagging.BaseBagging  
 method), 100

simple\_pruning\_base\_estimators() (hb-  
 brain.numerical\_data.ensemble\_learner.base\_cross\_val\_bagging.BaseCrossValBagging  
 method), 102

simple\_pruning\_base\_estimators() (hb-  
 brain.numerical\_data.ensemble\_learner.cross\_val\_random\_hyperboxes.CrossValRandomHyperboxesClassifier  
 method), 131

simple\_pruning\_base\_estimators() (hb-