
deslib Documentation

Release 0.2

Rafael Cruz

Apr 24, 2018

Contents

| | | |
|----------|----------------------------|-----------|
| 1 | Introduction | 3 |
| 2 | Philosophy | 5 |
| 3 | API Reference | 7 |
| 3.1 | User guide | 7 |
| 3.2 | API Reference | 13 |
| 3.3 | General examples | 71 |
| 3.4 | Release history | 77 |
| 4 | Example | 81 |
| 5 | Citation | 83 |
| 5.1 | References | 83 |
| | Python Module Index | 85 |

DESlib is an ensemble learning library focusing the implementation of the state-of-the-art techniques for dynamic classifier and ensemble selection.

DESlib is a work in progress. Contributions are welcomed through its GitHub page: <https://github.com/Menelau/DESlib>.

Dynamic Selection (DS) refers to techniques in which the base classifiers are selected on the fly, according to each new sample to be classified. Only the most competent, or an ensemble containing the most competent classifiers is selected to predict the label of a specific test sample. The rationale for such techniques is that not every classifier in the pool is an expert in classifying all unknown samples; rather, each base classifier is an expert in a different local region of the feature space.

DS is one of the most promising MCS approaches due to the fact that more and more works are reporting the superior performance of such techniques over static combination methods. Such techniques have achieved better classification performance especially when dealing with small-sized and imbalanced datasets. A comprehensive review of dynamic selection can be found in the following papers¹²

¹ : R. M. O. Cruz, R. Sabourin, and G. D. Cavalcanti, "Dynamic classifier selection: Recent advances and perspectives," *Information Fusion*, vol. 41, pp. 195 – 216, 2018.

² : A. S. Britto, R. Sabourin, L. E. S. de Oliveira, Dynamic selection of classifiers - A comprehensive review, *Pattern Recognition* 47 (11) (2014) 3665–3680.

DESlb was developed with two objectives in mind: to make it easy to integrate Dynamic Selection algorithms to machine learning projects, and to facilitate research on this topic, by providing implementations of the main DES and DCS methods, as well as the commonly used baseline methods. Each algorithm implements the main methods in the [scikit-learn](#) API **scikit-learn**: **fit(X, y)**, **predict(X)**, **predict_proba(X)** and **score(X, y)**.

The implementation of the DS methods is modular, following a taxonomy defined in¹. This taxonomy considers the main characteristics of DS methods, that are centered in three components:

1. the methodology used to define the local region, in which the competence level of the base classifiers are estimated (region of competence);
2. the source of information used to estimate the competence level of the base classifiers.
3. the selection approach to define the best classifier (for DCS) or the best set of classifiers (for DES).

This modular approach makes it easy for researchers to implement new DS methods, in many cases requiring only the implementation of the method **estimate_competence**, that is, how the local competence of the base classifier is measured.

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

3.1 User guide

This user guide explains how to install DESlib, how to contribute to the library and presents a step-by-step tutorial to fit and predict new instances using several dynamic selection techniques.

3.1.1 Installation

The library can be installed using pip:

Stable version:

```
pip install deslib
```

Latest version (under development):

```
pip install git+https://github.com/Menelau/DESlib
```

DESlib is tested to work with Python 3.5, and 3.6. The dependency requirements are:

- `scipy(>=0.13.3)`
- `numpy(>=1.10.4)`
- `scikit-learn(>=0.19.0)`

These dependencies are automatically installed using the pip commands above.

3.1.2 Development

DESlib was started by Rafael M. O. Cruz as a way to facilitate research in this topic by providing other researchers a toolbox with everything that is required to easily develop and compare different dynamic ensemble techniques.

The library is a work in progress. As an open-source project, any type of contribution is welcomed and encouraged!

Contributing to DESlib

You can contribute to the project in several ways:

- Reporting bugs
- Requesting features
- Improving the documentation
- Adding examples to use the library
- Implementing new features and fixing bugs

Reporting Bugs and requesting features

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 new feature implemented. Before opening a new issue, please check if the issue is not being currently addressed: [Issues](<https://github.com/Menelau/DESlib/issues>)

For reporting bugs:

- Include information of your working environment. This information can be found by running the following code snippet:

```
import platform; print(platform.platform())
import sys; print("Python", sys.version)
import numpy; print("NumPy", numpy.__version__)
import scipy; print("SciPy", scipy.__version__)
import sklearn; print("Scikit-Learn", sklearn.__version__)
```

- Include a [reproducible](<https://stackoverflow.com/help/mcve>) code snippet or link to a [gist](<https://gist.github.com>). If an exception is raised, please provide the traceback.

Documentation

We are glad 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 doc/ directory.

You can edit the documentation using any text editor and then generate the HTML output by typing `make html` from the doc/ directory. Alternatively, `make` can be used to quickly generate the documentation without the example gallery. The resulting HTML files will be placed in `_build/html/` and are viewable in a web browser. See the README file in the doc/ directory for more information.

For building the documentation, you will need to install `sphinx` and `sphinx_rtd_theme`. This can be easily done by installing the requirements for development using the following command:

```
pip install -r requirements-dev.txt
```

Contributing with code

The preferred way to contribute is to fork the main repository to your account:

1. Fork the [project repository](<https://github.com/Menelau/DESlib>): click on the ‘Fork’ button near the top of the page. This creates a copy of the code under your account on the GitHub server.

2. Clone this copy to your local disk:

```
$ git clone git@github.com:YourLogin/DESlib.git $ cd DESlib
```

3. Install all requirements for development:

```
$ pip install -r requirements-dev.txt $ pip install -e .
```

4. Create a branch to hold your changes:

```
$ git checkout -b branch_name
```

Where `branch_name` is the new feature or bug to be fixed. Do not work directly on the `master` branch.

5. Work on this copy on your computer using Git to do the version control. To record your changes in Git, then push them to GitHub with:

```
$ git push -u origin branch_name
```

It is important to assert your code is well covered by test routines (coverage of at least 90%), well documented and follows PEP8 guidelines.

6. Create a ‘Pull request’ to send your changes for review.

If your pull request addresses an issue, please use the title to describe the issue and mention the issue number in the pull request description to ensure a link is created to the original issue.

3.1.3 Tutorial

This tutorial will walk you through generating a pool of classifiers and applying several dynamic selection techniques for the classification of unknown samples. The tutorial assumes that you are already familiar with the [Python language](#) and the [scikit-learn](#) library. Users not familiar with either Python and scikit-learn can start by checking out their tutorials.

Running Dynamic selection with Bagging

In this first tutorial, we do a step-by-step run of the `example_bagging.py`, that is included in the examples part of the DESlib.

The first step is to run the example to check if everything is working as intended:

```
cd examples
python example_bagging.py
```

This script run six different dynamic selection models: Three DCS (OLA, A-Priori, MCB) and four DES (KNORA-Union, KNORA-Eliminate, DES-P and META-DES)

The example outputs the classification accuracy of each dataset:

```
Evaluating DS techniques:
Classification accuracy KNORA-Union:  0.973404255319
Classification accuracy KNORA-Eliminate:  0.968085106383
Classification accuracy DESP:  0.973404255319
Classification accuracy OLA:  0.968085106383
```

```
Classification accuracy A priori: 0.973404255319
Classification accuracy MCB: 0.968085106383
Classification accuracy META-DES: 0.973404255319
```

Code analysis:

The first thing to do is to import the corresponding DCS and DES algorithms that are tested as well as the other required libraries:

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
from sklearn.calibration import CalibratedClassifierCV
from sklearn.ensemble import BaggingClassifier

#importing DCS techniques from DESlib
from deslib.dcs.ola import OLA
from deslib.dcs.a_priori import APriori
from deslib.dcs.mcb import MCB

#import DES techniques from DESlib
from deslib.des.des_p import DESP
from deslib.des.knora_u import KNORAU
from deslib.des.knora_e import KNORAE
from deslib.des.meta_des import METADES
```

As DESlib is built on top of scikit-learn classifier, code will usually required the import of routines from this library.

Preparing the dataset:

Before exploiting the models, we need to prepare the dataset. We use the breast cancer dataset from scikit learn. The first step is to normalize the dataset so that it has zero mean and unit variance, which is a common requirement for many machine learning algorithms. This step can be easily done using the StandardScaler class from [scikit-learn](#).

Following that we divide it into three partitions: Training, Test, and Dynamic Selection (DSEL). We usually call the dataset that is used for the competence level estimation as the dynamic selection dataset (DSEL) instead of the validation dataset.

```
data = load_breast_cancer()
X = data.data
y = data.target
# split the data into training and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)

# Scale the variables to have 0 mean and unit variance
scalar = StandardScaler()
X_train = scalar.fit_transform(X_train)
X_test = scalar.transform(X_test)

# Split the data into training and DSEL for DS techniques
X_train, X_dsel, y_train, y_dsel = train_test_split(X_train, y_train, test_size=0.5)
```

starting from 0 to L-1 (where L is the number of classes). If your dataset does not follow this requirement, you can use the LabelEncoder class from scikit-learn to prepare the data. As the datasets loaded from scikit-learn already follow

this rule, we can skip this step.

Training a pool of classifiers:

The next step is to generate a pool of classifiers. Each implemented method receives as an input a list of classifiers. This list can be either homogeneous (i.e., all base classifiers are of the same type) or heterogeneous (base classifiers of different types). The library supports any type of base classifiers from scikit-learn library.

In this example, we generate a pool composed of 10 Perceptron classifiers using the Bagging technique. It is important to mention that some DS techniques require that the base classifiers are capable of estimating probabilities (i.e., implements the `predict_proba` function). For the Perceptron model, this can be achieved by calibrating the outputs of the base classifiers using the `CalibratedClassifierCV` class from scikit-learn.

```
model = CalibratedClassifierCV(Perceptron(max_iter=10))

# Train a pool of 10 classifiers
pool_classifiers = BaggingClassifier(model, n_estimators=10)
pool_classifiers.fit(X_train, y_train)
```

Building the DS models

Initializing DS techniques Here we initialize the DS techniques. Three DCS and four DES techniques are considered in this example: The only parameter that is required by the techniques is the pool of classifiers.

```
# DCS techniques
ola = OLA(pool_classifiers)
mcb = MCB(pool_classifiers)
apriori = APriori(pool_classifiers)

# DES techniques
knorau = KNORAU(pool_classifiers)
kne = KNORAE(pool_classifiers)
desp = DESP(pool_classifiers)
meta = METADES(pool_classifiers)
```

All others are optional parameters which can be specified explicitly changed in the instantiation of each method.

Fitting the DS techniques:

The next step is to fit the DS model. We call the function `fit` to prepare the DS techniques for the classification of new data by pre-processing the information required to apply the DS techniques, such as, fitting the algorithm used to estimate the region of competence (k-NN, k-Means) and calculating the source of competence of the base classifiers for each sample in the dynamic selection dataset.

```
knorau.fit(X_dsel, y_dsel)
kne.fit(X_dsel, y_dsel)
desp.fit(X_dsel, y_dsel)
ola.fit(X_dsel, y_dsel)
mcb.fit(X_dsel, y_dsel)
apriori.fit(X_dsel, y_dsel)
meta.fit(X_dsel, y_dsel)
```

Estimating classification accuracy:

Estimating the classification accuracy of each method is very easy. Each DS technique implements the function `score` from scikit-learn in order to estimate the classification accuracy.

```
print('Classification accuracy OLA: ', ola.score(X_test, y_test))
print('Classification accuracy A priori: ', apriori.score(X_test, y_test))
print('Classification accuracy KNORA-Union: ', knorau.score(X_test, y_test))
print('Classification accuracy KNORA-Eliminate: ', kne.score(X_test, y_test))
print('Classification accuracy DESP: ', desp.score(X_test, y_test))
print('Classification accuracy META-DES: ', apriori.score(X_test, y_test))
```

However, you may need to calculate the predictions of the model or the estimation of probabilities instead of only computing the accuracy. Class labels and posterior probabilities can be easily calculated using the **predict** and **predict_proba** methods:

```
metades.predict(X_test)
metades.predict_proba(X_test)
```

Changing parameters

Changing the hyper-parameters is very easy. We just need to pass its value when instantiating a new method. For example, we can change the size of the neighborhood used to estimate the competence level by setting the `k` value.

```
# DES techniques
knorau = KNORAU(pool_classifiers, k=5)
kne = KNORAE(pool_classifiers, k=5)
```

Also, we can change the mode DES algorithm works (dynamic selection, dynamic weighting or hybrid) by setting its mode: `.. code-block:: python`

```
meta = METADES(pool_classifiers, Hc=0.8, k=5, mode='hybrid')
```

In this code block, we change the size of the neighborhood (`k=5`), the value of the sample selection mechanism (`Hc=0.8`) and also, state that the META-DES algorithm should work in a hybrid dynamic selection with and weighting mode. The library accepts the change of several hyper-parameters. A list containing each one for all technique available as well as its impact in the algorithm is presented in the [API Reference](#).

Applying the Dynamic Frienemy Pruning (DFP)

The library also implements the Dynamic Frienemy Pruning (DFP) proposed in¹. So any dynamic selection technique can be applied using the FIRE (Frienemy Indecision Region Dynamic Ensemble Selection) framework. That is easily done by setting the DFP to true when initializing a DS technique. In this example, we show how to start the FIRE-KNORA-U, FIRE-KNORA-E and FIRE-MCB techniques.

```
fire_knorau = KNORAU(pool_classifiers, DFP=True)
fire_kne = KNORAE(pool_classifiers, DFP=True)
fire_mcb = MCB(pool_classifiers, DFP=True)
```

We can also set the size of the neighborhood that is used to decide whether the query sample is located in a safe region or in an indecision region (`safe_k`):

¹ : Oliveira, D.V.R., Cavalcanti, G.D.C. and Sabourin, R., Online Pruning of Base Classifiers for Dynamic Ensemble Selection, Pattern Recognition, vol. 72, December 2017, pp 44-58.


```
fire_knorau = KNORAU(pool_classifiers, DFP=True, safe_k=3)
fire_kne = KNORAE(pool_classifiers, DFP=True, safe_k=5)
fire_mcb = MCB(pool_classifiers, DFP=True, safe_k=7)
```

So, the `fire_knorau` will use a neighborhood composed of 3 samples, `fire_knorae` of 5 and `fire_mcb` of 7 in order to compute whether a given sample is located in a indecision or safe region.

More tutorials on how to use different aspects of the library can be found in [examples page](#)

References

3.2 API Reference

This is the full API documentation of the *DESlib*. Currently the library is divided into four modules:

3.2.1 Dynamic Classifier Selection (DCS)

This module contains the implementation of techniques in which only the base classifier that attained the highest competence level is selected for the classification of the query. The `deslib.dcs` provides a set of key dynamic classifier selection algorithms (DCS).

DCS base class

```
class deslib.dcs.base.DCS(pool_classifiers, k=7, DFP=False, safe_k=None, with_IH=False,
                          IH_rate=0.3, selection_method='best', diff_thresh=0.1,
                          rng=<mtrand.RandomState object>)
```

Base class for a Dynamic Classifier Selection (dcs) method. All dynamic classifier selection classes should inherit from this class.

Warning: This class should not be used directly, use derived classes instead.

Parameters

pool_classifiers [list of classifiers] The generated_pool of classifiers trained for the corresponding classification problem. The classifiers should support methods “predict” and “predict_proba”.

k [int (Default = 7)] Number of neighbors used to estimate the competence of the base classifiers.

DFP [Boolean (Default = False)] Determines if the dynamic frienemy pruning is applied.

with_IH [Boolean (Default = False)] Whether the hardness level of the region of competence is used to decide between using the DS algorithm or the KNN for classification of a given query sample.

safe_k [int (default = None)] The size of the indecision region.

IH_rate [float (default = 0.3)] Hardness threshold. If the hardness level of the competence region is lower than the `IH_rate` the KNN classifier is used. Otherwise, the DS algorithm is used for classification.

selection_method [String (Default = “best”)] Determines which method is used to select the base classifier after the competences are estimated.

diff_thresh [float (Default = 0.1)] Threshold to measure the difference between the competence level of the base classifiers for the random and diff selection schemes. If the difference is lower than the threshold, their performance are considered equivalent.

rng [numpy.random.RandomState instance] Random number generator to assure reproducible results.

References

Woods, Kevin, W. Philip Kegelmeyer, and Kevin Bowyer. “Combination of multiple classifiers using local accuracy estimates.” IEEE transactions on pattern analysis and machine intelligence 19.4 (1997): 405-410.

Britto, Alceu S., Robert Sabourin, and Luiz ES Oliveira. “Dynamic selection of classifiers—a comprehensive review.” Pattern Recognition 47.11 (2014): 3665-3680.

G. Giacinto and F. Roli, Methods for Dynamic Classifier Selection 10th Int. Conference on Image Analysis and Proc., Venice, Italy (1999), 659-664.

R. M. O. Cruz, R. Sabourin, and G. D. Cavalcanti, “Dynamic classifier selection: Recent advances and perspectives,” Information Fusion, vol. 41, pp. 195 – 216, 2018.

classify_with_ds (*query*, *predictions*, *probabilities=None*)

Predicts the class label of the corresponding query sample.

If self.selection_method == “all”, the majority voting scheme is used to aggregate the predictions of all classifiers with the max competence level estimates for each test examples.

Parameters

query [array of shape = [n_samples, n_features]] The test examples.

predictions [array of shape = [n_samples, n_classifiers]] Predictions of the base classifiers for all test examples.

probabilities [array of shape = [n_samples, n_classifiers, n_classes]] Probabilities estimates of each base classifier for all test examples. (For methods that always require probabilities from the base classifiers).

Returns

predicted_label [array of shape = [n_samples]] Predicted class label for each test example.

estimate_competence (*query*, *predictions=None*)

estimate the competence of each base classifier for the classification of the query sample.

Parameters

query [array of shape = [n_samples, n_features]] The test examples.

predictions [array of shape = [n_samples, n_classifiers]] Predictions of the base classifiers for all test examples.

Returns

competences [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

predict_proba_with_ds (*query*, *predictions*, *probabilities*)

Predicts the posterior probabilities of the corresponding query sample.

If self.selection_method == “all”, get the probability estimates of the selected ensemble. Otherwise, the technique gets the probability estimates from the selected base classifier

Parameters

- query** [array of shape = [n_samples, n_features]] The test example
- predictions** [array of shape = [n_samples, n_classifiers]] Predictions of the base classifiers for all test examples.
- probabilities** [array of shape = [n_samples, n_classifiers, n_classes]] Probabilities estimates of each base classifier for all test examples.

Returns

- predicted_proba** [array = [n_samples, n_classes]] Probability estimates for all test examples.

select (*competences*)

Select the most competent classifier for the classification of the query sample given the competence level estimates. Four selection schemes are available.

Best : The base classifier with the highest competence level is selected. In cases where more than one base classifier achieves the same competence level, the one with the lowest index is selected. This method is the standard for the LCA, OLA, MLA techniques.

Diff : Select the base classifier that is significantly better than the others in the pool (when the difference between its competence level and the competence level of the other base classifiers is higher than a predefined threshold). If no base classifier is significantly better, the base classifier is selected randomly among the member with equivalent competence level.

Random : Selects a random base classifier among all base classifiers that achieved the same competence level.

ALL : all base classifiers with the max competence level estimates are selected (note that in this case the DCS technique becomes a DES technique).

Parameters

- competences** [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

Returns

- selected_classifiers** [array of shape [n_samples]] Indices of the selected base classifier for each sample. If the selection_method is set to 'all', a boolean matrix is returned, containing True for the selected base classifiers, otherwise false.

A posteriori

```
class deslib.dcs.a_posteriori.APosteriori (pool_classifiers, k=7, DFP=False,
                                         with_IH=False, safe_k=None, IH_rate=0.3,
                                         selection_method='diff', diff_thresh=0.1,
                                         rng=<mtrand.RandomState object>)
```

A Posteriori Dynamic classifier selection.

The A Posteriori method uses the probability of correct classification of a given base classifier c_i for each neighbor x_k with respect to a single class. Consider a classifier c_i that assigns a test sample to class w_l . Then, only the samples belonging to class w_l are taken into account during the competence level estimates. Base classifiers with a higher probability of correct classification have a higher competence level. Moreover, the method also weights the influence of each neighbor x_k according to its Euclidean distance to the query sample. The closest neighbors have a higher influence on the competence level estimate. In cases where no sample in the region of competence belongs to the predicted class, w_l , the competence level estimate of the base classifier is equal to zero.

A single classifier is selected only if its competence level is significantly higher than that of the other base classifiers in the pool (higher than a pre-defined threshold). Otherwise, all classifiers in the pool are combined using the majority voting rule. The selection methodology can be modified by modifying the hyper-parameter `selection_method`.

Parameters

pool_classifiers [list of classifiers] The generated_pool of classifiers trained for the corresponding classification problem. The classifiers should support methods “predict” and “predict_proba”.

k [int (Default = 7)] Number of neighbors used to estimate the competence of the base classifiers.

DFP [Boolean (Default = False)] Determines if the dynamic frienemy pruning is applied.

with_IH [Boolean (Default = False)] Whether the hardness level of the region of competence is used to decide between using the DS algorithm or the KNN for classification of a given query sample.

safe_k [int (default = None)] The size of the indecision region.

IH_rate [float (default = 0.3)] Hardness threshold. If the hardness level of the competence region is lower than the IH_rate the KNN classifier is used. Otherwise, the DS algorithm is used for classification.

selection_method [String (Default = “best”)] Determines which method is used to select the base classifier after the competences are estimated.

diff_thresh [float (Default = 0.1)] Threshold to measure the difference between the competence level of the base classifiers for the random and diff selection schemes. If the difference is lower than the threshold, their performance are considered equivalent.

rng [numpy.random.RandomState instance] Random number generator to assure reproducible results.

References

G. Giacinto and F. Roli, Methods for Dynamic Classifier Selection 10th Int. Conf. on Image Anal. and Proc., Venice, Italy (1999), 659-664.

Ko, Albert HR, Robert Sabourin, and Alceu Souza Britto Jr. “From dynamic classifier selection to dynamic ensemble selection.” Pattern Recognition 41.5 (2008): 1718-1731.

Britto, Alceu S., Robert Sabourin, and Luiz ES Oliveira. “Dynamic selection of classifiers—a comprehensive review.” Pattern Recognition 47.11 (2014): 3665-3680.

R. M. O. Cruz, R. Sabourin, and G. D. Cavalcanti, “Dynamic classifier selection: Recent advances and perspectives,” Information Fusion, vol. 41, pp. 195 – 216, 2018.

estimate_competence (*query*, *predictions=None*)

estimate the competence of each base classifier c_i for the classification of the query sample using the A Posteriori method.

The competence level is estimated based on the probability of correct classification of the base classifier c_i , for each neighbor x_k belonging to a specific class w_l . In this case, w_l is the class predicted by the base classifier c_i , for the query sample. This method also weights the influence of each training sample according to its Euclidean distance to the query instance. The closest samples have a higher influence in the computation of the competence level. The competence level estimate is represented by the following

equation:

$$\delta_{i,j} = \frac{\sum_{\mathbf{x}_k \in \omega_l} P(\omega_l | \mathbf{x}_k, c_i) W_k}{\sum_{k=1}^K P(\omega_l | \mathbf{x}_k, c_i) W_k}$$

where $\delta_{i,j}$ represents the competence level of c_i for the classification of query.

Parameters

query [array of shape = [n_samples, n_features]] The query sample.

predictions [array of shape = [n_samples, n_classifiers]] Predictions of the base classifiers for the test examples.

Returns

competences [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

fit (X, y)

Prepare the DS model by setting the KNN algorithm and pre-processing the information required to apply the DS method.

Parameters

X [array of shape = [n_samples, n_features]] Data used to fit the model.

y [array of shape = [n_samples]] class labels of each example in X.

Returns

self

predict (X)

Predict the class label for each sample in X.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_labels [array of shape = [n_samples]] Predicted class label for each sample in X.

predict_proba (X)

Estimates the posterior probabilities for sample in X.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_proba [array of shape = [n_samples, n_classes]] Probabilities estimates for each sample in X.

score (X, y, sample_weight=None)

Returns the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

X [array-like, shape = (n_samples, n_features)] Test samples.

y [array-like, shape = (n_samples) or (n_samples, n_outputs)] True labels for X.

sample_weight [array-like, shape = [n_samples], optional] Sample weights.

Returns

score [float] Mean accuracy of self.predict(X) wrt. y.

select (*competences*)

Select the most competent classifier for the classification of the query sample given the competence level estimates. Four selection schemes are available.

Best : The base classifier with the highest competence level is selected. In cases where more than one base classifier achieves the same competence level, the one with the lowest index is selected. This method is the standard for the LCA, OLA, MLA techniques.

Diff : Select the base classifier that is significantly better than the others in the pool (when the difference between its competence level and the competence level of the other base classifiers is higher than a pre-defined threshold). If no base classifier is significantly better, the base classifier is selected randomly among the member with equivalent competence level.

Random : Selects a random base classifier among all base classifiers that achieved the same competence level.

ALL : all base classifiers with the max competence level estimates are selected (note that in this case the DCS technique becomes a DES technique).

Parameters

competences [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

Returns

selected_classifiers [array of shape [n_samples]] Indices of the selected base classifier for each sample. If the selection_method is set to 'all', a boolean matrix is returned, containing True for the selected base classifiers, otherwise false.

A Priori

```
class deslib.dcs.a_priori.APriori (pool_classifiers, k=7, DFP=False, with_IH=False,  
                                     safe_k=None, IH_rate=0.3, selection_method='diff',  
                                     diff_thresh=0.1, rng=<mtrand.RandomState object>)
```

A Priori dynamic classifier selection.

The A Priori method uses the probability of correct classification of a given base classifier c_i for each neighbor x_k for the competence level estimation. Base classifiers with a higher probability of correct classification have a higher competence level. Moreover, the method also weights the influence of each neighbor x_k according to its Euclidean distance to the query sample. The closest neighbors have a higher influence on the competence level estimate.

A single classifier is selected only if its competence level is significantly higher than that of the other base classifiers in the pool (higher than a pre-defined threshold). Otherwise, all classifiers in the pool are combined using the majority voting rule.

Parameters

pool_classifiers [list of classifiers] The generated_pool of classifiers trained for the corresponding classification problem. The classifiers should support methods "predict" and "predict_proba".

k [int (Default = 7)] Number of neighbors used to estimate the competence of the base classifiers.

DFP [Boolean (Default = False)] Determines if the dynamic frienemy pruning is applied.

with_IH [Boolean (Default = False)] Whether the hardness level of the region of competence is used to decide between using the DS algorithm or the KNN for classification of a given query sample.

safe_k [int (default = None)] The size of the indecision region.

IH_rate [float (default = 0.3)] Hardness threshold. If the hardness level of the competence region is lower than the IH_rate the KNN classifier is used. Otherwise, the DS algorithm is used for classification.

selection_method [String (Default = “best”)] Determines which method is used to select the base classifier after the competences are estimated.

diff_thresh [float (Default = 0.1)] Threshold to measure the difference between the competence level of the base classifiers for the random and diff selection schemes. If the difference is lower than the threshold, their performance are considered equivalent.

rng [numpy.random.RandomState instance] Random number generator to assure reproducible results.

References

G. Giacinto and F. Roli, Methods for Dynamic Classifier Selection 10th Int. Conf. on Image Anal. and Proc., Venice, Italy (1999), 659-664.

Ko, Albert HR, Robert Sabourin, and Alceu Souza Britto Jr. “From dynamic classifier selection to dynamic ensemble selection.” Pattern Recognition 41.5 (2008): 1718-1731.

Britto, Alceu S., Robert Sabourin, and Luiz ES Oliveira. “Dynamic selection of classifiers—a comprehensive review.” Pattern Recognition 47.11 (2014): 3665-3680.

R. M. O. Cruz, R. Sabourin, and G. D. Cavalcanti, “Dynamic classifier selection: Recent advances and perspectives,” Information Fusion, vol. 41, pp. 195 – 216, 2018.

estimate_competence (*query*, *predictions=None*)

estimate the competence of each base classifier c_i for the classification of the query sample using the A Priori rule:

The competence level is estimated based on the probability of correct classification of the base classifier c_i , considering all samples in the region of competence. This method also weights the influence of each training sample according to its Euclidean distance to the query instance. The closest samples have a higher influence in the computation of the competence level. The competence level estimate is represented by the following equation:

$$\delta_{i,j} = \frac{\sum_{k=1}^K P(\omega_l \mid \mathbf{x}_k \in \omega_l, c_i) W_k}{\sum_{k=1}^K W_k}$$

where $\delta_{i,j}$ represents the competence level of c_i for the classification of query.

Parameters

query [array of shape = [n_samples, n_features]] The test examples.

predictions [array of shape = [n_samples, n_classifiers]] Predictions of the base classifiers for the test examples.

Returns

competences [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

fit (X, y)

Prepare the DS model by setting the KNN algorithm and pre-processing the information required to apply the DS method.

Parameters

X [array of shape = [n_samples, n_features]] Data used to fit the model.

y [array of shape = [n_samples]] class labels of each example in X.

Returns

self

predict (X)

Predict the class label for each sample in X.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_labels [array of shape = [n_samples]] Predicted class label for each sample in X.

predict_proba (X)

Estimates the posterior probabilities for sample in X.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_proba [array of shape = [n_samples, n_classes]] Probabilities estimates for each sample in X.

score (X, y, *sample_weight=None*)

Returns the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

X [array-like, shape = (n_samples, n_features)] Test samples.

y [array-like, shape = (n_samples) or (n_samples, n_outputs)] True labels for X.

sample_weight [array-like, shape = [n_samples], optional] Sample weights.

Returns

score [float] Mean accuracy of self.predict(X) wrt. y.

select (*competences*)

Select the most competent classifier for the classification of the query sample given the competence level estimates. Four selection schemes are available.

Best : The base classifier with the highest competence level is selected. In cases where more than one base classifier achieves the same competence level, the one with the lowest index is selected. This method is the standard for the LCA, OLA, MLA techniques.

Diff : Select the base classifier that is significantly better than the others in the pool (when the difference between its competence level and the competence level of the other base classifiers is higher than a predefined threshold). If no base classifier is significantly better, the base classifier is selected randomly among the member with equivalent competence level.

Random : Selects a random base classifier among all base classifiers that achieved the same competence level.

ALL : all base classifiers with the max competence level estimates are selected (note that in this case the DCS technique becomes a DES technique).

Parameters

competences [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

Returns

selected_classifiers [array of shape [n_samples]] Indices of the selected base classifier for each sample. If the selection_method is set to 'all', a boolean matrix is returned, containing True for the selected base classifiers, otherwise false.

Local Class Accuracy (LCA)

```
class deslib.dcs.lca.LCA(pool_classifiers, k=7, DFP=False, with_IH=False, safe_k=None,
                        IH_rate=0.3, selection_method='best', diff_thresh=0.1,
                        rng=<mtrand.RandomState object>)
```

Local Class Accuracy (LCA).

Evaluates the competence level of each individual classifiers and select the most competent one to predict the label of each test sample. The competence of each base classifier is calculated based on its local accuracy with respect to some output class. Consider a classifier c_i that assigns a test sample to class w_l . The competence level of c_i is estimated by the percentage of the local training samples assigned to class w_l that it predicts the correct class label.

The LCA method selects the base classifier presenting the highest competence level. In a case where more than one base classifier achieves the same competence level, the one that was evaluated first is selected. The selection methodology can be modified by changing the hyper-parameter selection_method.

Parameters

pool_classifiers [list of classifiers] The generated_pool of classifiers trained for the corresponding classification problem. The classifiers should support the method "predict".

k [int (Default = 7)] Number of neighbors used to estimate the competence of the base classifiers.

DFP [Boolean (Default = False)] Determines if the dynamic frienemy pruning is applied.

with_IH [Boolean (Default = False)] Whether the hardness level of the region of competence is used to decide between using the DS algorithm or the KNN for classification of a given query sample.

safe_k [int (default = None)] The size of the indecision region.

IH_rate [float (default = 0.3)] Hardness threshold. If the hardness level of the competence region is lower than the IH_rate the KNN classifier is used. Otherwise, the DS algorithm is used for classification.

selection_method [String (Default = "best")] Determines which method is used to select the base classifier after the competences are estimated.

diff_thresh [float (Default = 0.1)] Threshold to measure the difference between the competence level of the base classifiers for the random and diff selection schemes. If the difference is lower than the threshold, their performance are considered equivalent.

rng [numpy.random.RandomState instance] Random number generator to assure reproducible results.

References

Woods, Kevin, W. Philip Kegelmeyer, and Kevin Bowyer. “Combination of multiple classifiers using local accuracy estimates.” IEEE transactions on pattern analysis and machine intelligence 19.4 (1997): 405-410.

Britto, Alceu S., Robert Sabourin, and Luiz ES Oliveira. “Dynamic selection of classifiers—a comprehensive review.” Pattern Recognition 47.11 (2014): 3665-3680.

R. M. O. Cruz, R. Sabourin, and G. D. Cavalcanti, “Dynamic classifier selection: Recent advances and perspectives,” Information Fusion, vol. 41, pp. 195 – 216, 2018.

estimate_competence (*query*, *predictions=None*)

estimate the competence of each base classifier c_i for the classification of the query sample using the local class accuracy method.

In this algorithm the k-Nearest Neighbors of the test sample are estimated. Then, the local accuracy of the base classifiers is estimated by its classification accuracy taking into account only the samples from the class w_l in this neighborhood. In this case, w_l is the class predicted by the base classifier c_i , for the query sample. The competence level estimate is represented by the following equation:

$$\delta_{i,j} = \frac{\sum_{\mathbf{x}_k \in \omega_l} P(\omega_l | \mathbf{x}_k, c_i)}{\sum_{k=1}^K P(\omega_l | \mathbf{x}_k, c_i)}$$

where $\delta_{i,j}$ represents the competence level of c_i for the classification of query.

Parameters

query [array of shape = [n_samples, n_features]] The test examples.

predictions [array of shape = [n_samples, n_classifiers]] Predictions of the base classifiers for the test examples.

Returns

competences [array of shape = [n_samples, n_classifiers]] The competence level estimated for each base classifier and test example.

fit (*X*, *y*)

Prepare the DS model by setting the KNN algorithm and pre-processing the information required to apply the DS methods

Parameters

X [array of shape = [n_samples, n_features]] The input data.

y [array of shape = [n_samples]] class labels of each example in X.

Returns

self

predict (*X*)

Predict the class label for each sample in X.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_labels [array of shape = [n_samples]] Predicted class label for each sample in X.

predict_proba(X)

Estimates the posterior probabilities for sample in X.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_proba [array of shape = [n_samples, n_classes]] Probabilities estimates for each sample in X.

score(X, y, sample_weight=None)

Returns the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

X [array-like, shape = (n_samples, n_features)] Test samples.

y [array-like, shape = (n_samples) or (n_samples, n_outputs)] True labels for X.

sample_weight [array-like, shape = [n_samples], optional] Sample weights.

Returns

score [float] Mean accuracy of self.predict(X) wrt. y.

select(competences)

Select the most competent classifier for the classification of the query sample given the competence level estimates. Four selection schemes are available.

Best : The base classifier with the highest competence level is selected. In cases where more than one base classifier achieves the same competence level, the one with the lowest index is selected. This method is the standard for the LCA, OLA, MLA techniques.

Diff : Select the base classifier that is significantly better than the others in the pool (when the difference between its competence level and the competence level of the other base classifiers is higher than a predefined threshold). If no base classifier is significantly better, the base classifier is selected randomly among the member with equivalent competence level.

Random : Selects a random base classifier among all base classifiers that achieved the same competence level.

ALL : all base classifiers with the max competence level estimates are selected (note that in this case the DCS technique becomes a DES technique).

Parameters

competences [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

Returns

selected_classifiers [array of shape [n_samples]] Indices of the selected base classifier for each sample. If the selection_method is set to 'all', a boolean matrix is returned, containing True for the selected base classifiers, otherwise false.

Multiple Classifier Behaviour (MCB)

```
class deslib.dcs.mcb.MCB(pool_classifiers, k=7, DFP=False, with_IH=False, safe_k=None,  
                        IH_rate=0.3, similarity_threshold=0.7, selection_method='diff',  
                        diff_thresh=0.1, rng=<mtrand.RandomState object>)
```

Multiple Classifier Behaviour (MCB).

The MCB method evaluates the competence level of each individual classifiers taking into account the local accuracy of the base classifier in the region of competence. The region of competence is defined using the k-NN and behavioral knowledge space (BKS) method. First the k-nearest neighbors of the test sample are computed. Then, the set containing the k-nearest neighbors is filtered based on the similarity of the query sample and its neighbors using the decision space (BKS representation).

A single classifier c_i is selected only if its competence level is significantly higher than that of the other base classifiers in the pool (higher than a pre-defined threshold). Otherwise, all classifiers in the pool are combined using the majority voting rule. The selection methodology can be modified by changing the hyper-parameter `selection_method`.

Parameters

pool_classifiers [list of classifiers] The generated_pool of classifiers trained for the corresponding classification problem. The classifiers should support the method “predict”.

k [int (Default = 7)] Number of neighbors used to estimate the competence of the base classifiers.

DFP [Boolean (Default = False)] Determines if the dynamic frienemy pruning is applied.

with_IH [Boolean (Default = False)] Whether the hardness level of the region of competence is used to decide between using the DS algorithm or the KNN for classification of a given query sample.

safe_k [int (default = None)] The size of the indecision region.

IH_rate [float (default = 0.3)] Hardness threshold. If the hardness level of the competence region is lower than the IH_rate the KNN classifier is used. Otherwise, the DS algorithm is used for classification.

selection_method [String (Default = “best”)] Determines which method is used to select the base classifier after the competences are estimated.

diff_thresh [float (Default = 0.1)] Threshold to measure the difference between the competence level of the base classifiers for the random and diff selection schemes. If the difference is lower than the threshold, their performance are considered equivalent.

rng [numpy.random.RandomState instance] Random number generator to assure reproducible results.

References

Giacinto, Giorgio, and Fabio Roli. “Dynamic classifier selection based on multiple classifier behaviour.” Pattern Recognition 34.9 (2001): 1879-1881.

Britto, Alceu S., Robert Sabourin, and Luiz ES Oliveira. “Dynamic selection of classifiers—a comprehensive review.” Pattern Recognition 47.11 (2014): 3665-3680.

Huang, Yea S., and Ching Y. Suen. “A method of combining multiple experts for the recognition of unconstrained handwritten numerals.” IEEE Transactions on Pattern Analysis and Machine Intelligence 17.1 (1995): 90-94.

Huang, Yea S., and Ching Y. Suen. “The behavior-knowledge space method for combination of multiple classifiers.” IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1993.

R. M. O. Cruz, R. Sabourin, and G. D. Cavalcanti, “Dynamic classifier selection: Recent advances and perspectives,” Information Fusion, vol. 41, pp. 195 – 216, 2018.

estimate_competence (*query*, *predictions=None*)

estimate the competence of each base classifier c_i for the classification of the query sample using the Multiple Classifier Behaviour criterion.

The region of competence in this method is estimated taking into account the feature space and the decision space (using the behaviour knowledge space method [4]). First, the k-Nearest Neighbors of the query sample are defined in the feature space to compose the region of competence. Then, the similarity in the BKS space between the query and the instances in the region of competence are estimated using the following equations:

$$S(\tilde{\mathbf{x}}_j, \tilde{\mathbf{x}}_k) = \frac{1}{M} \sum_{i=1}^M T(\mathbf{x}_j, \mathbf{x}_k)$$

$$T(\mathbf{x}_j, \mathbf{x}_k) = \begin{cases} 1 & \text{if } c_i(\mathbf{x}_j) = c_i(\mathbf{x}_k), \\ 0 & \text{if } c_i(\mathbf{x}_j) \neq c_i(\mathbf{x}_k). \end{cases}$$

Where $S(\tilde{\mathbf{x}}_j, \tilde{\mathbf{x}}_k)$ denotes the similarity between two samples based on the behaviour knowledge space method (BKS). Instances with similarity lower than a predefined threshold are removed from the region of competence. The competence level of the base classifiers are estimated as their classification accuracy in the final region of competence.

Parameters

query [array of shape = [n_samples, n_features]] The test samples.

predictions [array of shape = [n_samples, n_classifiers]] Predictions of the base classifiers for the test examples.

Returns

competences [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

fit (*X*, *y*)

Prepare the DS model by setting the KNN algorithm and pre-processing the information required to apply the DS methods

Parameters

X [array of shape = [n_samples, n_features]] The input data.

y [array of shape = [n_samples]] class labels of each example in X.

Returns

self

predict (*X*)

Predict the class label for each sample in X.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_labels [array of shape = [n_samples]] Predicted class label for each sample in X.

predict_proba (*X*)

Estimates the posterior probabilities for sample in *X*.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_proba [array of shape = [n_samples, n_classes]] Probabilities estimates for each sample in *X*.

score (*X*, *y*, *sample_weight=None*)

Returns the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

X [array-like, shape = (n_samples, n_features)] Test samples.

y [array-like, shape = (n_samples) or (n_samples, n_outputs)] True labels for *X*.

sample_weight [array-like, shape = [n_samples], optional] Sample weights.

Returns

score [float] Mean accuracy of self.predict(*X*) wrt. *y*.

select (*competences*)

Select the most competent classifier for the classification of the query sample given the competence level estimates. Four selection schemes are available.

Best : The base classifier with the highest competence level is selected. In cases where more than one base classifier achieves the same competence level, the one with the lowest index is selected. This method is the standard for the LCA, OLA, MLA techniques.

Diff : Select the base classifier that is significantly better than the others in the pool (when the difference between its competence level and the competence level of the other base classifiers is higher than a predefined threshold). If no base classifier is significantly better, the base classifier is selected randomly among the member with equivalent competence level.

Random : Selects a random base classifier among all base classifiers that achieved the same competence level.

ALL : all base classifiers with the max competence level estimates are selected (note that in this case the DCS technique becomes a DES technique).

Parameters

competences [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

Returns

selected_classifiers [array of shape [n_samples]] Indices of the selected base classifier for each sample. If the selection_method is set to 'all', a boolean matrix is returned, containing True for the selected base classifiers, otherwise false.

Modified Local Accuracy (MLA)

```
class deslib.dcs.mla.MLA(pool_classifiers, k=7, DFP=False, with_IH=False, safe_k=None,
                          IH_rate=0.3, selection_method='best', diff_thresh=0.1,
                          rng=<mtrand.RandomState object>)
```

Modified Local Accuracy (MLA).

Similar to the LCA technique. The only difference is that the output of each base classifier is weighted by the distance between the test sample and each pattern in the region of competence for the estimation of the classifiers competences. Only the classifier that achieved the highest competence level is select to predict the label of the test sample x .

The MLA method selects the base classifier presenting the highest competence level. In a case where more than one base classifier achieves the same competence level, the one that was evaluated first is selected. The selection methodology can be modified by changing the hyper-parameter `selection_method`.

Parameters

pool_classifiers [list of classifiers] The generated_pool of classifiers trained for the corresponding classification problem. The classifiers should support the method “predict”.

k [int (Default = 7)] Number of neighbors used to estimate the competence of the base classifiers.

DFP [Boolean (Default = False)] Determines if the dynamic frienemy pruning is applied.

with_IH [Boolean (Default = False)] Whether the hardness level of the region of competence is used to decide between using the DS algorithm or the KNN for classification of a given query sample.

safe_k [int (default = None)] The size of the indecision region.

IH_rate [float (default = 0.3)] Hardness threshold. If the hardness level of the competence region is lower than the `IH_rate` the KNN classifier is used. Otherwise, the DS algorithm is used for classification.

selection_method [String (Default = “best”)] Determines which method is used to select the base classifier after the competences are estimated.

diff_thresh [float (Default = 0.1)] Threshold to measure the difference between the competence level of the base classifiers for the random and diff selection schemes. If the difference is lower than the threshold, their performance are considered equivalent.

rng [numpy.random.RandomState instance] Random number generator to assure reproducible results.

References

Woods, Kevin, W. Philip Kegelmeyer, and Kevin Bowyer. “Combination of multiple classifiers using local accuracy estimates.” IEEE transactions on pattern analysis and machine intelligence 19.4 (1997): 405-410.

Britto, Alceu S., Robert Sabourin, and Luiz ES Oliveira. “Dynamic selection of classifiers—a comprehensive review.” Pattern Recognition 47.11 (2014): 3665-3680.

R. M. O. Cruz, R. Sabourin, and G. D. Cavalcanti, “Dynamic classifier selection: Recent advances and perspectives,” Information Fusion, vol. 41, pp. 195 – 216, 2018.

estimate_competence (*query*, *predictions*=None)

estimate the competence of each base classifier c_i for the classification of the query sample using the Modified Local Accuracy (MLA) method.

The competence level of the base classifiers is estimated by its classification accuracy taking into account only the samples belonging to a given class w_l . In this case, w_l is the class predicted by the base classifier c_i , for the query sample. This method also weights the influence of each training sample according to its Euclidean distance to the query instance. The closest samples have a higher influence in the computation of the competence level. The competence level estimate is represented by the following equation:

$$\delta_{i,j} = \sum_{k=1}^K P(\omega_l \mid \mathbf{x}_k \in \omega_l, c_i) W_k$$

where $\delta_{i,j}$ represents the competence level of c_i for the classification of query.

Parameters

query [array of shape = [n_samples, n_features]] The query sample.

predictions [array of shape = [n_samples, n_classifiers]] Predictions of the base classifiers for the test examples.

Returns

competences [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier.

fit (*X*, *y*)

Prepare the DS model by setting the KNN algorithm and pre-processing the information required to apply the DS methods

Parameters

X [array of shape = [n_samples, n_features]] The input data.

y [array of shape = [n_samples]] class labels of each example in X.

Returns

self

predict (*X*)

Predict the class label for each sample in X.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_labels [array of shape = [n_samples]] Predicted class label for each sample in X.

predict_proba (*X*)

Estimates the posterior probabilities for sample in X.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_proba [array of shape = [n_samples, n_classes]] Probabilities estimates for each sample in X.

score (*X*, *y*, *sample_weight=None*)

Returns the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

X [array-like, shape = (n_samples, n_features)] Test samples.

y [array-like, shape = (n_samples) or (n_samples, n_outputs)] True labels for X.

sample_weight [array-like, shape = [n_samples], optional] Sample weights.

Returns

score [float] Mean accuracy of self.predict(X) wrt. y.

select (*competences*)

Select the most competent classifier for the classification of the query sample given the competence level estimates. Four selection schemes are available.

Best : The base classifier with the highest competence level is selected. In cases where more than one base classifier achieves the same competence level, the one with the lowest index is selected. This method is the standard for the LCA, OLA, MLA techniques.

Diff : Select the base classifier that is significantly better than the others in the pool (when the difference between its competence level and the competence level of the other base classifiers is higher than a predefined threshold). If no base classifier is significantly better, the base classifier is selected randomly among the member with equivalent competence level.

Random : Selects a random base classifier among all base classifiers that achieved the same competence level.

ALL : all base classifiers with the max competence level estimates are selected (note that in this case the DCS technique becomes a DES technique).

Parameters

competences [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

Returns

selected_classifiers [array of shape [n_samples]] Indices of the selected base classifier for each sample. If the selection_method is set to 'all', a boolean matrix is returned, containing True for the selected base classifiers, otherwise false.

Overall Local Accuracy (OLA)

```
class deslib.dcs.ola.OLA(pool_classifiers, k=7, DFP=False, with_IH=False, safe_k=None,
                        IH_rate=0.3, selection_method='best', diff_thresh=0.1,
                        rng=<mtrand.RandomState object>)
```

Overall Classifier Accuracy (OLA).

The OLA method evaluates the competence level of each individual classifiers and select the most competent one to predict the label of each test sample x. The competence of each base classifier is calculated as its classification accuracy in the neighborhood of x (region of competence).

The LCA method selects the base classifier presenting the highest competence level. In a case where more than one base classifier achieves the same competence level, the one that was evaluated first is selected. The selection methodology can be modified by changing the hyper-parameter selection_method.

Parameters

pool_classifiers [list of classifiers] The generated_pool of classifiers trained for the corresponding classification problem. The classifiers should support the method "predict".

k [int (Default = 7)] Number of neighbors used to estimate the competence of the base classifiers.

DFP [Boolean (Default = False)] Determines if the dynamic frienemy pruning is applied.

with_IH [Boolean (Default = False)] Whether the hardness level of the region of competence is used to decide between using the DS algorithm or the KNN for classification of a given query sample.

safe_k [int (default = None)] The size of the indecision region.

IH_rate [float (default = 0.3)] Hardness threshold. If the hardness level of the competence region is lower than the IH_rate the KNN classifier is used. Otherwise, the DS algorithm is used for classification.

selection_method [String (Default = “best”)] Determines which method is used to select the base classifier after the competences are estimated.

diff_thresh [float (Default = 0.1)] Threshold to measure the difference between the competence level of the base classifiers for the random and diff selection schemes. If the difference is lower than the threshold, their performance are considered equivalent.

rng [numpy.random.RandomState instance] Random number generator to assure reproducible results.

References

Woods, Kevin, W. Philip Kegelmeyer, and Kevin Bowyer. “Combination of multiple classifiers using local accuracy estimates.” IEEE transactions on pattern analysis and machine intelligence 19.4 (1997): 405-410.

Britto, Alceu S., Robert Sabourin, and Luiz ES Oliveira. “Dynamic selection of classifiers—a comprehensive review.” Pattern Recognition 47.11 (2014): 3665-3680.

R. M. O. Cruz, R. Sabourin, and G. D. Cavalcanti, “Dynamic classifier selection: Recent advances and perspectives,” Information Fusion, vol. 41, pp. 195 – 216, 2018.

estimate_competence (*query*, *predictions=None*)

estimate the competence level of each base classifier c_i for the classification of the query sample.

The competences for each base classifier c_i is estimated by its classification accuracy considering the k-Nearest Neighbors (region of competence). The competence level estimate is represented by the following equation:

$$\delta_{i,j} = \frac{1}{K} \sum_{k=1}^K P(\omega_l \mid \mathbf{x}_k \in \omega_l, c_i)$$

where $\delta_{i,j}$ represents the competence level of c_i for the classification of query.

Parameters

query [array of shape = [m_samples, n_features]] The test examples.

predictions [array of shape = [n_samples, n_classifiers]] Predictions of the base classifiers for the test examples.

Returns

competences [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

fit (*X*, *y*)

Prepare the DS model by setting the KNN algorithm and pre-processing the information required to apply the DS methods

Parameters

X [array of shape = [n_samples, n_features]] The input data.

y [array of shape = [n_samples]] class labels of each example in X.

Returns

self

predict (*X*)

Predict the class label for each sample in X.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_labels [array of shape = [n_samples]] Predicted class label for each sample in X.

predict_proba (*X*)

Estimates the posterior probabilities for sample in X.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_proba [array of shape = [n_samples, n_classes]] Probabilities estimates for each sample in X.

score (*X*, *y*, *sample_weight=None*)

Returns the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

X [array-like, shape = (n_samples, n_features)] Test samples.

y [array-like, shape = (n_samples) or (n_samples, n_outputs)] True labels for X.

sample_weight [array-like, shape = [n_samples], optional] Sample weights.

Returns

score [float] Mean accuracy of self.predict(X) wrt. y.

select (*competences*)

Select the most competent classifier for the classification of the query sample given the competence level estimates. Four selection schemes are available.

Best : The base classifier with the highest competence level is selected. In cases where more than one base classifier achieves the same competence level, the one with the lowest index is selected. This method is the standard for the LCA, OLA, MLA techniques.

Diff : Select the base classifier that is significantly better than the others in the pool (when the difference between its competence level and the competence level of the other base classifiers is higher than a predefined threshold). If no base classifier is significantly better, the base classifier is selected randomly among the member with equivalent competence level.

Random : Selects a random base classifier among all base classifiers that achieved the same competence level.

ALL : all base classifiers with the max competence level estimates are selected (note that in this case the DCS technique becomes a DES technique).

Parameters

competences [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

Returns

selected_classifiers [array of shape [n_samples]] Indices of the selected base classifier for each sample. If the selection_method is set to 'all', a boolean matrix is returned, containing True for the selected base classifiers, otherwise false.

Modified Rank

```
class deslib.dcs.rank.Rank(pool_classifiers, k=7, DFP=False, with_IH=False, safe_k=None,
                           IH_rate=0.3, selection_method='best', diff_thresh=0.1,
                           rng=<mtrand.RandomState object>)
```

Modified Classifier Rank.

The modified classifier rank method evaluates the competence level of each individual classifiers and select the most competent one to predict the label of each test sample x . The competence of each base classifier is calculated as the number of correctly classified samples, starting from the closest neighbor of x . The classifier with the highest number of correctly classified samples is considered the most competent.

The Rank method selects the base classifier presenting the highest competence level. In a case where more than one base classifier achieves the same competence level, the one that was evaluated first is selected. The selection methodology can be modified by changing the hyper-parameter selection_method.

Parameters

pool_classifiers [list of classifiers] The generated_pool of classifiers trained for the corresponding classification problem. The classifiers should support the method "predict".

k [int (Default = 7)] Number of neighbors used to estimate the competence of the base classifiers.

DFP [Boolean (Default = False)] Determines if the dynamic frienemy pruning is applied.

with_IH [Boolean (Default = False)] Whether the hardness level of the region of competence is used to decide between using the DS algorithm or the KNN for classification of a given query sample.

safe_k [int (default = None)] The size of the indecision region.

IH_rate [float (default = 0.3)] Hardness threshold. If the hardness level of the competence region is lower than the IH_rate the KNN classifier is used. Otherwise, the DS algorithm is used for classification.

selection_method [String (Default = "best")] Determines which method is used to select the base classifier after the competences are estimated.

diff_thresh [float (Default = 0.1)] Threshold to measure the difference between the competence level of the base classifiers for the random and diff selection schemes. If the difference is lower than the threshold, their performance are considered equivalent.

rng [numpy.random.RandomState instance] Random number generator to assure reproducible results.

References

Woods, Kevin, W. Philip Kegelmeyer, and Kevin Bowyer. “Combination of multiple classifiers using local accuracy estimates.” *IEEE transactions on pattern analysis and machine intelligence* 19.4 (1997): 405-410.

M. Sabourin, A. Mitiche, D. Thomas, G. Nagy, Classifier combination for handprinted digit recognition, *International Conference on Document Analysis and Recognition* (1993) 163–166.

Britto, Alceu S., Robert Sabourin, and Luiz ES Oliveira. “Dynamic selection of classifiers—a comprehensive review.” *Pattern Recognition* 47.11 (2014): 3665-3680.

R. M. O. Cruz, R. Sabourin, and G. D. Cavalcanti, “Dynamic classifier selection: Recent advances and perspectives,” *Information Fusion*, vol. 41, pp. 195 – 216, 2018.

estimate_competence (*query*, *predictions=None*)

estimate the competence level of each base classifier c_i for the classification of the query sample using the modified ranking scheme. The rank of the base classifier is estimated by the number of consecutive correctly classified samples in the defined region of competence.

Parameters

query [array of shape = [n_samples, n_features]] The test examples.

predictions [array of shape = [n_samples, n_classifiers]] Predictions of the base classifiers for the test examples.

Returns

competences [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

fit (*X*, *y*)

Prepare the DS model by setting the KNN algorithm and pre-processing the information required to apply the DS methods

Parameters

X [array of shape = [n_samples, n_features]] The input data.

y [array of shape = [n_samples]] class labels of each example in X.

Returns

self

predict (*X*)

Predict the class label for each sample in X.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_labels [array of shape = [n_samples]] Predicted class label for each sample in X.

predict_proba (*X*)

Estimates the posterior probabilities for sample in X.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_proba [array of shape = [n_samples, n_classes]] Probabilities estimates for each sample in X.

score (X, y, *sample_weight=None*)

Returns the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

X [array-like, shape = (n_samples, n_features)] Test samples.

y [array-like, shape = (n_samples) or (n_samples, n_outputs)] True labels for X.

sample_weight [array-like, shape = [n_samples], optional] Sample weights.

Returns

score [float] Mean accuracy of self.predict(X) wrt. y.

select (*competences*)

Select the most competent classifier for the classification of the query sample given the competence level estimates. Four selection schemes are available.

Best : The base classifier with the highest competence level is selected. In cases where more than one base classifier achieves the same competence level, the one with the lowest index is selected. This method is the standard for the LCA, OLA, MLA techniques.

Diff : Select the base classifier that is significantly better than the others in the pool (when the difference between its competence level and the competence level of the other base classifiers is higher than a predefined threshold). If no base classifier is significantly better, the base classifier is selected randomly among the member with equivalent competence level.

Random : Selects a random base classifier among all base classifiers that achieved the same competence level.

ALL : all base classifiers with the max competence level estimates are selected (note that in this case the DCS technique becomes a DES technique).

Parameters

competences [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

Returns

selected_classifiers [array of shape [n_samples]] Indices of the selected base classifier for each sample. If the selection_method is set to 'all', a boolean matrix is returned, containing True for the selected base classifiers, otherwise false.

3.2.2 Dynamic Ensemble Selection (DES)

Dynamic ensemble selection strategies refer to techniques that select an ensemble of classifier rather than a single one. All base classifiers that attain a minimum competence level are selected to compose the ensemble of classifiers. The *deslib.des* provides a set of key dynamic ensemble selection algorithms (DES).

DES base class

class `deslib.des.base.DES` (*pool_classifiers*, *k*=7, *DFP*=False, *with_IH*=False, *safe_k*=None, *IH_rate*=0.3, *mode*='selection', *needs_proba*=False)

Base class for a Dynamic Ensemble Selection (DES).

All dynamic ensemble selection techniques should inherit from this class.

Warning: This class should not be instantiated directly, use derived classes instead.

Parameters

pool_classifiers [list of classifiers] The generated_pool of classifiers trained for the corresponding classification problem. The classifiers should support methods “predict” and “predict_proba”.

k [int (Default = 7)] Number of neighbors used to estimate the competence of the base classifiers.

DFP [Boolean (Default = False)] Determines if the dynamic frienemy pruning is applied.

with_IH [Boolean (Default = False)] Whether the hardness level of the region of competence is used to decide between using the DS algorithm or the KNN for classification of a given query sample.

safe_k [int (default = None)] The size of the indecision region.

IH_rate [float (default = 0.3)] Hardness threshold. If the hardness level of the competence region is lower than the IH_rate the KNN classifier is used. Otherwise, the DS algorithm is used for classification.

mode [String (Default = “selection”)] Whether the technique will perform dynamic selection, dynamic weighting or an hybrid approach for classification.

needs_proba [Boolean (Default = False)] Determines whether the method always needs base classifiers that estimate probabilities.

References

Britto, Alceu S., Robert Sabourin, and Luiz ES Oliveira. “Dynamic selection of classifiers—a comprehensive review.” *Pattern Recognition* 47.11 (2014): 3665-3680.

R. M. O. Cruz, R. Sabourin, and G. D. Cavalcanti, “Dynamic classifier selection: Recent advances and perspectives,” *Information Fusion*, vol. 41, pp. 195 – 216, 2018.

classify_with_ds (*query*, *predictions*, *probabilities*=None)

Predicts the label of the corresponding query sample.

If self.mode == “selection”, the selected ensemble is combined using the majority voting rule

If self.mode == “weighting”, all base classifiers are used for classification, however their influence in the final decision are weighted according to their estimated competence level. The weighted majority voting scheme is used to combine the decisions of the base classifiers.

If self.mode == “hybrid”, A hybrid Dynamic selection and weighting approach is used. First an ensemble with the competent base classifiers are selected. Then, their decisions are aggregated using the weighted majority voting rule according to its competence level estimates.

Parameters

query [array of shape = [n_samples, n_features]] The test examples.

predictions [array of shape = [n_samples, n_classifiers]] Predictions of the base classifier for all test examples.

probabilities [array of shape = [n_samples, n_classifiers, n_classes]] Probabilities estimates of each base classifier for all test examples. (For methods that always require probabilities from the base classifiers).

Returns

predicted_label [array of shape = [n_samples]] Predicted class label for each test example.

estimate_competence (*query, predictions*)

Estimate the competence of each base classifier c_i the classification of the query sample x . Returns an array containing the level of competence estimated for each base classifier. The size of the vector is equals to the size of the generated_pool of classifiers.

Parameters

query [array of shape = [n_samples, n_features]] The test examples

predictions [array of shape = [n_samples, n_classifiers]] Predictions of the base classifiers for all test examples.

Returns

competences [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

estimate_competence_from_proba (*query, probabilities*)

estimate the competence of each base classifier c_i the classification of the query sample x , for methods that require probabilities. Returns an array containing the level of competence estimated for each base classifier. The size of the vector is equals to the size of the generated_pool of classifiers.

Parameters

query [array of shape = [n_samples, n_features]] The query sample.

probabilities [array of shape = [n_samples, n_classifiers, n_classes]] Probabilities estimates of each base classifier for all test examples.

Returns

competences [array = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

predict_proba_with_ds (*query, predictions, probabilities*)

Predicts the posterior probabilities of the corresponding query sample.

If self.mode == “selection”, the selected ensemble is used to estimate the probabilities. The average rule is used to give probabilities estimates.

If self.mode == “weighting”, all base classifiers are used for estimating the probabilities, however their influence in the final decision are weighted according to their estimated competence level. A weighted average method is used to give the probabilities estimates.

If self.mode == “Hybrid”, A hybrid Dynamic selection and weighting approach is used. First an ensemble with the competent base classifiers are selected. Then, their decisions are aggregated using a weighted average rule to give the probabilities estimates.

Parameters

query [array of shape = [n_samples, n_features]] The test examples.

predictions [array of shape = [n_samples, n_classifiers]] Predictions of the base classifier for all test examples.

probabilities [array of shape = [n_samples, n_classifiers, n_classes]] Probabilities estimates of each base classifier for all test examples.

Returns

predicted_proba [array = [n_samples, n_classes]] The probability estimates for all test examples.

select (*competences*)

Select the most competent classifiers to compose an ensemble for the classification of the query sample X.

Parameters

competences [array of shape = [n_samples, n_classifiers]] Estimated competence level of each base classifier for each test example.

Returns

selected_classifiers [array of shape = [n_samples, n_classifiers]] Boolean matrix containing True if the base classifier is select, False otherwise.

META-DES

```
class deslib.des.meta_des.METADES (pool_classifiers, meta_classifier=MultinomialNB(alpha=1.0,
class_prior=None, fit_prior=True), k=7, kp=5, Hc=1.0,
selection_threshold=0.5, mode='selection', DFP=False,
with_IH=False, safe_k=None, IH_rate=0.3)
```

Meta learning for dynamic ensemble selection (META-DES).

The META-DES framework is based on the assumption that the dynamic ensemble selection problem can be considered as a meta-problem. This meta-problem uses different criteria regarding the behavior of a base classifier c_i , in order to decide whether it is competent enough to classify a given test sample.

The framework performs a meta-training stage, in which, the meta-features are extracted from each instance belonging to the training and the dynamic selection dataset (DSEL). Then, the extracted meta-features are used to train the meta-classifier λ . The meta-classifier is trained to predict whether or not a base classifier c_i is competent enough to classify a given input sample.

When an unknown sample is presented to the system, the meta-features for each base classifier c_i in relation to the input sample are calculated and presented to the meta-classifier. The meta-classifier estimates the competence level of the base classifier c_i for the classification of the query sample. Base classifiers with competence level higher than a pre-defined threshold are selected. If no base classifier is selected, the whole pool is used for classification.

Parameters

pool_classifiers [list of classifiers] The generated_pool of classifiers trained for the corresponding classification problem. The classifiers should support methods “predict” and “predict_proba”.

k [int (Default = 7)] Number of neighbors used to estimate the competence of the base classifiers.

kp [int (Default = 5)] Number of output profiles used to estimate the competence of the base classifiers.

Hc [float (Default = 1.0)] Sample selection threshold.

selection_threshold [float(Default = 0.5)] Threshold used to select the base classifier. Only the base classifiers with competence level higher than the selection_threshold are selected to compose the ensemble.

mode [String (Default = “selection”)] Determines the mode of META-des that is used (selection, weighting or hybrid).

DFP [Boolean (Default = False)] Determines if the dynamic frienemy pruning is applied.

with_IH [Boolean (Default = False)] Whether the hardness level of the region of competence is used to decide between using the DS algorithm or the KNN for classification of a given query sample.

safe_k [int (default = None)] The size of the indecision region.

IH_rate [float (default = 0.3)] Hardness threshold. If the hardness level of the competence region is lower than the IH_rate the KNN classifier is used. Otherwise, the DS algorithm is used for classification.

References

Cruz, R.M., Sabourin, R., Cavalcanti, G.D. and Ren, T.I., 2015. META-DES: A dynamic ensemble selection framework using meta-learning. *Pattern Recognition*, 48(5), pp.1925-1935.

Cruz, R.M., Sabourin, R. and Cavalcanti, G.D., 2015, July. META-des. H: a dynamic ensemble selection technique using meta-learning and a dynamic weighting approach. In *Neural Networks (IJCNN), 2015 International Joint Conference on* (pp. 1-8).

R. M. O. Cruz, R. Sabourin, and G. D. Cavalcanti, “Dynamic classifier selection: Recent advances and perspectives,” *Information Fusion*, vol. 41, pp. 195 – 216, 2018.

estimate_competence_from_proba (*query, probabilities*)

Estimate the competence of each base classifier c_i the classification of the query sample. This method received an array with the pre-calculated probability estimates for each query.

First, the meta-features of each base classifier c_i for the classification of the query sample are estimated. These meta-features are passed down to the meta-classifier λ for the competence level estimation.

Parameters

query [array of shape = [n_samples, n_features]] The test examples.

probabilities [array of shape = [n_samples, n_classifiers, n_classes]] Probabilities estimates obtained by each each base classifier for each query sample.

Returns

competences [array of shape = [n_samples, n_classifiers]] The competence level estimated for each base classifier and test example.

fit (*X, y*)

Prepare the DS model by setting the KNN algorithm and pre-processing the information required to apply the DS method.

This method also extracts the meta-features and trains the meta-classifier λ if the meta-classifier was not yet trained.

Parameters

X [array of shape = [n_samples, n_features]] Data used to fit the model.

y [array of shape = [n_samples]] class labels of each example in X.

Returns

self

predict (*X*)

Predict the class label for each sample in *X*.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_labels [array of shape = [n_samples]] Predicted class label for each sample in *X*.

predict_proba (*X*)

Estimates the posterior probabilities for sample in *X*.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_proba [array of shape = [n_samples, n_classes]] Probabilities estimates for each sample in *X*.

score (*X*, *y*, *sample_weight=None*)

Returns the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

X [array-like, shape = (n_samples, n_features)] Test samples.

y [array-like, shape = (n_samples) or (n_samples, n_outputs)] True labels for *X*.

sample_weight [array-like, shape = [n_samples], optional] Sample weights.

Returns

score [float] Mean accuracy of self.predict(*X*) wrt. *y*.

select (*competences*)

Selects the base classifiers that obtained a competence level higher than the predefined threshold defined in self.selection_threshold.

Parameters

competences [array of shape = [n_samples, n_classifiers]] The competence level estimated for each base classifier and test example.

Returns

selected_classifiers [array of shape = [n_samples, n_classifiers]] Boolean matrix containing True if the base classifier is select, False otherwise.

DES Clustering

```
class deslib.des.des_clustering.DESClustering(pool_classifiers, k=5, pct_accuracy=0.5,  
                                              pct_diversity=0.33, more_diverse=True,  
                                              metric='DF', rng=<mtrand.RandomState  
                                              object>)
```

Dynamic ensemble selection-Clustering (DES-Clustering).

This method selects an ensemble of classifiers taking into account the accuracy and diversity of the base classifiers. The K-means algorithm is used to define the region of competence. For each cluster, the N most accurate

classifiers are first selected. Then, the J more diverse classifiers from the N most accurate classifiers are selected to compose the ensemble.

Parameters

- pool_classifiers** [list of classifiers] The generated_pool of classifiers trained for the corresponding classification problem. The classifiers should support the method “predict”.
- k** [int (Default = 5)] Number of neighbors used to estimate the competence of the base classifiers.
- pct_accuracy** [float (Default = 0.5)] Percentage of base classifiers selected based on accuracy
- pct_diversity** [float (Default = 0.33)] Percentage of base classifiers selected based n diversity
- more_diverse** [Boolean (Default = True)] Whether we select the most or the least diverse classifiers to add to the pre-selected ensemble
- metric** [String (Default = ‘df’)] Diversity diversity_func used to estimate the diversity of the base classifiers. Can be either the double fault (df), Q-statistics (Q), or error correlation (corr)
- rng** [numpy.random.RandomState instance] Random number generator to assure reproducible results.

References

Soares, R. G., Santana, A., Canuto, A. M., & de Souto, M. C. P. “Using accuracy and more_diverse to select classifiers to build ensembles.” International Joint Conference on Neural Networks (IJCNN), 2006.

Britto, Alceu S., Robert Sabourin, and Luiz ES Oliveira. “Dynamic selection of classifiers—a comprehensive review.” Pattern Recognition 47.11 (2014): 3665-3680.

R. M. O. Cruz, R. Sabourin, and G. D. Cavalcanti, “Dynamic classifier selection: Recent advances and perspectives,” Information Fusion, vol. 41, pp. 195 – 216, 2018.

estimate_competence (*query*, *predictions=None*)

Get the competence estimates of each base classifier c_i for the classification of the query sample.

In this case, the competences were already pre-calculated for each cluster. So this method computes the nearest cluster and get the pre-calculated competences of the base classifiers for the corresponding cluster.

Parameters

- query** [array of shape = [n_samples, n_features]] The query sample.
- predictions** [array of shape = [n_samples, n_classifiers]] Predictions of the base classifiers for all test examples.

Returns

- competences** [array = [n_samples, n_classifiers]] The competence level estimated for each base classifier.

fit (*X*, *y*)

Train the DS model by setting the Clustering algorithm and pre-processing the information required to apply the DS methods.

First the data is divided into K clusters. Then, for each cluster, the N most accurate classifiers are first selected. Then, the J more diverse classifiers from the N most accurate classifiers are selected to compose the ensemble of the corresponding cluster. An ensemble of classifiers is assigned to each of the K clusters.

Parameters

X [array of shape = [n_samples, n_features]] Data used to fit the model.

y [array of shape = [n_samples]] class labels of each example in X.

Returns

self

predict (X)

Predict the class label for each sample in X.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_labels [array of shape = [n_samples]] Predicted class label for each sample in X.

predict_proba (X)

Estimates the posterior probabilities for sample in X.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_proba [array of shape = [n_samples, n_classes]] Probabilities estimates for each sample in X.

score (X, y, sample_weight=None)

Returns the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

X [array-like, shape = (n_samples, n_features)] Test samples.

y [array-like, shape = (n_samples) or (n_samples, n_outputs)] True labels for X.

sample_weight [array-like, shape = [n_samples], optional] Sample weights.

Returns

score [float] Mean accuracy of self.predict(X) wrt. y.

select (query)

Select an ensemble with the most accurate and most diverse classifier for the classification of the query.

The ensemble for each cluster was already pre-calculated in the fit method. So, this method calculates the closest cluster, and returns the ensemble associated to this cluster.

Parameters

query [array of shape = [n_samples, n_features]] The test examples.

Returns

selected_classifiers [array of shape = [n_samples, self.k]] Indices of the selected base classifier for each test example.

Dynamic Ensemble Selection performance (DES-P)

class `deslib.des.des_p.DESP` (*pool_classifiers*, *k*=7, *DFP*=False, *with_IH*=False, *safe_k*=None, *IH_rate*=0.3, *mode*='selection')

Dynamic ensemble selection-Performance(DES-P).

This method selects all base classifiers that achieve a classification performance, in the region of competence, that is higher than the random classifier (RC). The performance of the random classifier is defined by $RC = 1/L$, where L is the number of classes in the problem. If no base classifier is selected, the whole pool is used for classification.

Parameters

pool_classifiers [list of classifiers] The generated_pool of classifiers trained for the corresponding classification problem. The classifiers should support the method “predict”.

k [int (Default = 7)] Number of neighbors used to estimate the competence of the base classifiers.

DFP [Boolean (Default = False)] Determines if the dynamic frienemy pruning is applied.

with_IH [Boolean (Default = False)] Whether the hardness level of the region of competence is used to decide between using the DS algorithm or the KNN for classification of a given query sample.

safe_k [int (default = None)] The size of the indecision region.

IH_rate [float (default = 0.3)] Hardness threshold. If the hardness level of the competence region is lower than the *IH_rate* the KNN classifier is used. Otherwise, the DS algorithm is used for classification.

mode [String (Default = “selection”)] Whether the technique will perform dynamic selection, dynamic weighting or an hybrid approach for classification.

References

Woloszynski, Tomasz, et al. “A measure of competence based on random classification for dynamic ensemble selection.” *Information Fusion* 13.3 (2012): 207-213.

Woloszynski, Tomasz, and Marek Kurzynski. “A probabilistic model of classifier competence for dynamic ensemble selection.” *Pattern Recognition* 44.10 (2011): 2656-2668.

R. M. O. Cruz, R. Sabourin, and G. D. Cavalcanti, “Dynamic classifier selection: Recent advances and perspectives,” *Information Fusion*, vol. 41, pp. 195 – 216, 2018.

estimate_competence (*query*, *predictions*=None)

estimate the competence of each base classifier c_i for the classification of the query sample base on its local performance.

$$\delta_{i,j} = \hat{P}(c_i | \theta_j) - \frac{1}{L}$$

Parameters

query [array of shape = [n_samples, n_features]] The test examples.

predictions [array of shape = [n_samples, n_classifiers]] Predictions of the base classifiers for all test examples.

Returns

competences [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

fit (X, y)

Prepare the DS model by setting the KNN algorithm and pre-processing the information required to apply the DS methods

Parameters

X [array of shape = [n_samples, n_features]] The input data.

y [array of shape = [n_samples]] class labels of each example in X.

Returns

self

predict (X)

Predict the class label for each sample in X.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_labels [array of shape = [n_samples]] Predicted class label for each sample in X.

predict_proba (X)

Estimates the posterior probabilities for sample in X.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_proba [array of shape = [n_samples, n_classes]] Probabilities estimates for each sample in X.

score (X, y, *sample_weight=None*)

Returns the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

X [array-like, shape = (n_samples, n_features)] Test samples.

y [array-like, shape = (n_samples) or (n_samples, n_outputs)] True labels for X.

sample_weight [array-like, shape = [n_samples], optional] Sample weights.

Returns

score [float] Mean accuracy of self.predict(X) wrt. y.

select (*competences*)

Selects all base classifiers that obtained a local classification accuracy higher than the Random Classifier. The performance of the random classifier is denoted $1/L$, where L is the number of classes in the problem.

Parameters

competences [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

Returns

selected_classifiers [array of shape = [n_samples, n_classifiers]] Boolean matrix containing True if the base classifier is select, False otherwise.

DES-KNN

```
class deslib.des.des_knn.DESKNN (pool_classifiers, k=7, DFP=False, with_IH=False,  
                                safe_k=None, IH_rate=0.3, pct_accuracy=0.5,  
                                pct_diversity=0.3, more_diverse=True, metric='DF')
```

Dynamic ensemble Selection KNN (DES-KNN).

This method selects an ensemble of classifiers taking into account the accuracy and diversity of the base classifiers. The k-NN algorithm is used to define the region of competence. The N most accurate classifiers in the region of competence are first selected. Then, the J more diverse classifiers from the N most accurate classifiers are selected to compose the ensemble.

Parameters

pool_classifiers [list of classifiers] The generated_pool of classifiers trained for the corresponding classification problem. The classifiers should support the method “predict”.

k [int (Default = 5)] Number of neighbors used to estimate the competence of the base classifiers.

DFP [Boolean (Default = False)] Determines if the dynamic frienemy pruning is applied.

with_IH [Boolean (Default = False)] Whether the hardness level of the region of competence is used to decide between using the DS algorithm or the KNN for classification of a given query sample.

safe_k [int (default = None)] The size of the indecision region.

IH_rate [float (default = 0.3)] Hardness threshold. If the hardness level of the competence region is lower than the IH_rate the KNN classifier is used. Otherwise, the DS algorithm is used for classification.

pct_accuracy [float (Default = 0.5)] Percentage of base classifiers selected based on accuracy

pct_diversity [float (Default = 0.3)] Percentage of base classifiers selected based n diversity

more_diverse [Boolean (Default = True)] Whether we select the most or the least diverse classifiers to add to the pre-selected ensemble

metric [String (Default = ‘df’)] Diversity diversity_func used to estimate the diversity of the base classifiers. Can be either the double fault (df), Q-statistics (Q), or error correlation (corr)

References

Soares, R. G., Santana, A., Canuto, A. M., & de Souto, M. C. P. “Using accuracy and more_diverse to select classifiers to build ensembles.” International Joint Conference on Neural Networks (IJCNN)., 2006.

Britto, Alceu S., Robert Sabourin, and Luiz ES Oliveira. “Dynamic selection of classifiers—a comprehensive review.” Pattern Recognition 47.11 (2014): 3665-3680.

R. M. O. Cruz, R. Sabourin, and G. D. Cavalcanti, “Dynamic classifier selection: Recent advances and perspectives,” Information Fusion, vol. 41, pp. 195 – 216, 2018.

estimate_competence (*query*, *predictions*=None)

estimate the competence level of each base classifier c_i for the classification of the query sample.

The competence is estimated using the accuracy and diversity criteria. First the classification accuracy of the base classifiers in the region of competence is estimated. Then the diversity of the base classifiers in the region of competence is estimated.

The method returns two arrays: One containing the accuracy and the other the diversity of each base classifier.

Parameters

query [array of shape = [n_samples, n_features]] The query sample.

predictions [array of shape = [n_samples, n_classifiers]] Predictions of the base classifiers for all test examples.

Returns

accuracy [array of shape = [n_samples, n_classifiers]] Local Accuracy estimates (competences) of the base classifiers for all query samples.

diversity [array of shape = [n_samples, n_classifiers]] Average pairwise diversity of each base classifiers for all test examples.

Notes

This technique uses both the accuracy and diversity information to perform dynamic selection. For this reason the function returns a dictionary containing these two values instead of a single ndarray containing the competence level estimates for each base classifier.

fit (X, y)

Prepare the DS model by setting the KNN algorithm and pre-processing the information required to apply the DS methods

Parameters

X [array of shape = [n_samples, n_features]] The input data.

y [array of shape = [n_samples]] class labels of each example in X.

Returns

self

predict (X)

Predict the class label for each sample in X.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_labels [array of shape = [n_samples]] Predicted class label for each sample in X.

predict_proba (X)

Estimates the posterior probabilities for sample in X.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_proba [array of shape = [n_samples, n_classes]] Probabilities estimates for each sample in X.

score (*X*, *y*, *sample_weight=None*)

Returns the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

X [array-like, shape = (n_samples, n_features)] Test samples.

y [array-like, shape = (n_samples) or (n_samples, n_outputs)] True labels for X.

sample_weight [array-like, shape = [n_samples], optional] Sample weights.

Returns

score [float] Mean accuracy of self.predict(X) wrt. y.

select (*accuracy*, *diversity*)

Select an ensemble containing the N most accurate and the J most diverse classifiers for the classification of the query sample.

Parameters

accuracy [array of shape = [n_samples, n_classifiers]] Local Accuracy estimates (competence) of each base classifiers for all query samples.

diversity [array of shape = [n_samples, n_classifiers]] Average pairwise diversity of each base classifiers for all test examples.

Returns

selected_classifiers [array of shape = [n_samples, self.J]] Matrix containing the indices of the J selected base classifier for each test example.

k-Nearest Output Profiles (KNOP)

class deslib.des.knop.**KNOP** (*pool_classifiers*, *k=7*, *DFP=False*, *with_IH=False*, *safe_k=None*, *IH_rate=0.3*)
k-Nearest Output Profiles (KNOP).

This method selects all classifiers that correctly classified at least one sample belonging to the region of competence of the query sample. In this case, the region of competence is estimated using the decisions of the base classifier (output profiles). Thus, the similarity between the query and the validation samples are measured in the decision space rather than the feature space. Each selected classifier has a number of votes equals to the number of samples in the region of competence that it predicts the correct label. The votes obtained by all base classifiers are aggregated to obtain the final ensemble decision.

Parameters

pool_classifiers [type, the generated_pool of classifiers trained for the corresponding]
classification problem.

k [int (Default = 7)] Number of neighbors used to estimate the competence of the base classifiers.

DFP [Boolean (Default = False)] Determines if the dynamic frienemy pruning is applied.

with_IH [Boolean (Default = False)] Whether the hardness level of the region of competence is used to decide between using the DS algorithm or the KNN for classification of a given query sample.

safe_k [int (default = None)] The size of the indecision region.

IH_rate [float (default = 0.3)] Hardness threshold. If the hardness level of the competence region is lower than the IH_rate the KNN classifier is used. Otherwise, the DS algorithm is used for classification.

References

Cavalin, Paulo R., Robert Sabourin, and Ching Y. Suen. “LoGID: An adaptive framework combining local and global incremental learning for dynamic selection of ensembles of HMMs.” *Pattern Recognition* 45.9 (2012): 3544-3556.

Cavalin, Paulo R., Robert Sabourin, and Ching Y. Suen. “Dynamic selection approaches for multiple classifier systems.” *Neural Computing and Applications* 22.3-4 (2013): 673-688.

Ko, Albert HR, Robert Sabourin, and Alceu Souza Britto Jr. “From dynamic classifier selection to dynamic ensemble selection.” *Pattern Recognition* 41.5 (2008): 1718-1731.

Britto, Alceu S., Robert Sabourin, and Luiz ES Oliveira. “Dynamic selection of classifiers—a comprehensive review.” *Pattern Recognition* 47.11 (2014): 3665-3680.

R. M. O. Cruz, R. Sabourin, and G. D. Cavalcanti, “Dynamic classifier selection: Recent advances and perspectives,” *Information Fusion*, vol. 41, pp. 195 – 216, 2018.

estimate_competence_from_proba (*query*, *probabilities*)

The competence of the base classifiers is simply estimated as the number of samples in the region of competence that it correctly classified. This method received an array with the pre-calculated probability estimates for each query.

This information is later used to determine the number of votes obtained for each base classifier.

Parameters

query [array of shape = [n_samples, n_features]] The test examples.

probabilities [array of shape = [n_samples, n_classifiers, n_classes]] Probabilities estimates obtained by each each base classifier for each query sample.

Returns

competences [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

fit (*X*, *y*)

Train the DS model by setting the KNN algorithm and pre-process the information required to apply the DS methods. In this case, the scores of the base classifiers for the dynamic selection dataset (DSEL) are pre-calculated to transform each sample in DSEL into an output profile.

Parameters

X [array of shape = [n_samples, n_features]] Data used to fit the model.

y [array of shape = [n_samples]] class labels of each example in X.

Returns

self

predict (*X*)

Predict the class label for each sample in X.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_labels [array of shape = [n_samples]] Predicted class label for each sample in X.

predict_proba (X)

Estimates the posterior probabilities for sample in X.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_proba [array of shape = [n_samples, n_classes]] Probabilities estimates for each sample in X.

score (X, y, sample_weight=None)

Returns the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

X [array-like, shape = (n_samples, n_features)] Test samples.

y [array-like, shape = (n_samples) or (n_samples, n_outputs)] True labels for X.

sample_weight [array-like, shape = [n_samples], optional] Sample weights.

Returns

score [float] Mean accuracy of self.predict(X) wrt. y.

select (competences)

Select the base classifiers for the classification of the query sample.

Each base classifier can be selected more than once. The number of times a base classifier is selected (votes) is equals to the number of samples it correctly classified in the region of competence.

Parameters

competences [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

Returns

selected_classifiers [array of shape = [n_samples, n_classifiers]] Boolean matrix containing True if the base classifier is select, False otherwise.

k-Nearest Oracle-Eliminate (KNORA-E)

```
class deslib.des.knora_e.KNORAE (pool_classifiers, k=7, DFP=False, with_IH=False,  
                                safe_k=None, IH_rate=0.3)
```

k-Nearest Oracles Eliminate (KNORA-E).

This method searches for a local Oracle, which is a base classifier that correctly classify all samples belonging to the region of competence of the test sample. All classifiers with a perfect performance in the region of competence are selected (local Oracles). In the case that no classifiers achieve a perfect accuracy, the size of the region of competence is reduced (by removing the farthest neighbor) and the performance of the classifiers are re-evaluated. The outputs of the selected ensemble of classifiers is combined using the majority voting scheme. If no base classifier is selected, the whole pool is used for classification.

Parameters

pool_classifiers [list of classifiers] The generated_pool of classifiers trained for the corresponding classification problem. The classifiers should support the method “predict”.

k [int (Default = 7)] Number of neighbors used to estimate the competence of the base classifiers.

DFP [Boolean (Default = False)] Determines if the dynamic frienemy pruning is applied.

with_IH [Boolean (Default = False)] Whether the hardness level of the region of competence is used to decide between using the DS algorithm or the KNN for classification of a given query sample.

safe_k [int (default = None)] The size of the indecision region.

IH_rate [float (default = 0.3)] Hardness threshold. If the hardness level of the competence region is lower than the IH_rate the KNN classifier is used. Otherwise, the DS algorithm is used for classification.

References

Ko, Albert HR, Robert Sabourin, and Alceu Souza Britto Jr. “From dynamic classifier selection to dynamic ensemble selection.” Pattern Recognition 41.5 (2008): 1718-1731.

Britto, Alceu S., Robert Sabourin, and Luiz ES Oliveira. “Dynamic selection of classifiers—a comprehensive review.” Pattern Recognition 47.11 (2014): 3665-3680.

R. M. O. Cruz, R. Sabourin, and G. D. Cavalcanti, “Dynamic classifier selection: Recent advances and perspectives,” Information Fusion, vol. 41, pp. 195 – 216, 2018.

estimate_competence (*query*, *predictions=None*)

Estimate the competence of the base classifiers. In the case of the KNORA-E technique, the classifiers are only considered competent when they achieve a 100% accuracy in the region of competence. For each base, we estimate the maximum size of the region of competence that it is a local oracle. The competence level estimate is then the maximum size of the region of competence that the corresponding base classifier is considered a local Oracle.

Parameters

query [array of shape = [n_samples, n_features]] The test examples.

predictions [array of shape = [n_samples, n_classifiers]] Predictions of the base classifiers for all test examples.

Returns

competences [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

fit (*X*, *y*)

Prepare the DS model by setting the KNN algorithm and pre-processing the information required to apply the DS methods

Parameters

X [array of shape = [n_samples, n_features]] The input data.

y [array of shape = [n_samples]] class labels of each example in X.

Returns

self

predict (*X*)

Predict the class label for each sample in *X*.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_labels [array of shape = [n_samples]] Predicted class label for each sample in *X*.

predict_proba (*X*)

Estimates the posterior probabilities for sample in *X*.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_proba [array of shape = [n_samples, n_classes]] Probabilities estimates for each sample in *X*.

score (*X*, *y*, *sample_weight=None*)

Returns the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

X [array-like, shape = (n_samples, n_features)] Test samples.

y [array-like, shape = (n_samples) or (n_samples, n_outputs)] True labels for *X*.

sample_weight [array-like, shape = [n_samples], optional] Sample weights.

Returns

score [float] Mean accuracy of self.predict(*X*) wrt. *y*.

select (*competences*)

Selects all base classifiers that obtained a local accuracy of 100% in the region of competence (i.e., local oracle). In the case that no base classifiers obtain 100% accuracy, the size of the region of competence is reduced and the search for the local oracle is restarted.

Parameters

competences [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

Returns

selected_classifiers [array of shape = [n_samples, n_classifiers]] Boolean matrix containing True if the base classifier is select, False otherwise.

Notes

Instead of re-applying the method several times (reducing the size of the region of competence), we compute the number of consecutive correct classification of each base classifier starting from the closest neighbor to the more distant in the `estimate_competence` function. The number of consecutive correct classification represents the size of the region of competence in which the corresponding base classifier is an Local Oracle. Then, we select all base classifiers with the maximum value for the number of consecutive correct classification. This speed up the selection process.

k-Nearest Oracle Union (KNORA-U)

```
class deslib.des.knora_u.KNORAU (pool_classifiers, k=7, DFP=False, with_IH=False,
                                safe_k=None, IH_rate=0.3)
```

k-Nearest Oracles Union (KNORA-U).

This method selects all classifiers that correctly classified at least one sample belonging to the region of competence of the query sample. Each selected classifier has a number of votes equals to the number of samples in the region of competence that it predicts the correct label. The votes obtained by all base classifiers are aggregated to obtain the final ensemble decision.

Parameters

pool_classifiers [list of classifiers] The generated_pool of classifiers trained for the corresponding classification problem. The classifiers should support the method “predict”.

k [int (Default = 7)] Number of neighbors used to estimate the competence of the base classifiers.

DFP [Boolean (Default = False)] Determines if the dynamic frienemy pruning is applied.

with_IH [Boolean (Default = False)] Whether the hardness level of the region of competence is used to decide between using the DS algorithm or the KNN for classification of a given query sample.

safe_k [int (default = None)] The size of the indecision region.

IH_rate [float (default = 0.3)] Hardness threshold. If the hardness level of the competence region is lower than the IH_rate the KNN classifier is used. Otherwise, the DS algorithm is used for classification.

References

Ko, Albert HR, Robert Sabourin, and Alceu Souza Britto Jr. “From dynamic classifier selection to dynamic ensemble selection.” Pattern Recognition 41.5 (2008): 1718-1731.

Britto, Alceu S., Robert Sabourin, and Luiz ES Oliveira. “Dynamic selection of classifiers—a comprehensive review.” Pattern Recognition 47.11 (2014): 3665-3680.

R. M. O. Cruz, R. Sabourin, and G. D. Cavalcanti, “Dynamic classifier selection: Recent advances and perspectives,” Information Fusion, vol. 41, pp. 195 – 216, 2018.

estimate_competence (*query*, *predictions*=None)

The competence of the base classifiers is simply estimated as the number of samples in the region of competence that it correctly classified.

This information is later used to determine the number of votes obtained for each base classifier.

Parameters

query [array of shape = [n_samples, n_features]] The test examples.

predictions [array of shape = [n_samples, n_classifiers]] Predictions of the base classifiers for all test examples.

Returns

competences [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

fit (*X*, *y*)

Prepare the DS model by setting the KNN algorithm and pre-processing the information required to apply the DS methods

Parameters

X [array of shape = [n_samples, n_features]] The input data.

y [array of shape = [n_samples]] class labels of each example in X.

Returns

self

predict (*X*)

Predict the class label for each sample in X.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_labels [array of shape = [n_samples]] Predicted class label for each sample in X.

predict_proba (*X*)

Estimates the posterior probabilities for sample in X.

Parameters

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_proba [array of shape = [n_samples, n_classes]] Probabilities estimates for each sample in X.

score (*X*, *y*, *sample_weight=None*)

Returns the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

X [array-like, shape = (n_samples, n_features)] Test samples.

y [array-like, shape = (n_samples) or (n_samples, n_outputs)] True labels for X.

sample_weight [array-like, shape = [n_samples], optional] Sample weights.

Returns

score [float] Mean accuracy of self.predict(X) wrt. y.

select (*competences*)

Select the base classifiers for the classification of the query sample.

Each base classifier can be selected more than once. The number of times a base classifier is selected (votes) is equals to the number of samples it correctly classified in the region of competence.

Parameters

competences [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

Returns

selected_classifiers [array of shape = [n_samples, n_classifiers]] Boolean matrix containing True if the base classifier is select, False otherwise.

Probabilistic

```
class deslib.des.probablistic.Probabilistic(pool_classifiers, k=None, DFP=False,
                                             with_IH=False, safe_k=None,
                                             IH_rate=0.3, mode='selection', selec-
                                             tion_threshold=None)
```

Base class for a DS method based on the potential function model. ALL DS methods based on the Potential function should inherit from this class

Warning: This class should not be used directly. Use derived classes instead.

Parameters

pool_classifiers [list of classifiers] The generated_pool of classifiers trained for the corresponding classification problem. The classifiers should support methods “predict” and “predict_proba”.

k [int (Default = None)] Number of neighbors used to estimate the competence of the base classifiers. If k = None, the whole dynamic selection dataset is used, and the influence of each sample is based on its distance to the query.

DFP [Boolean (Default = False)] Determines if the dynamic frienemy pruning is applied.

with_IH [Boolean (Default = False)] Whether the hardness level of the region of competence is used to decide between using the DS algorithm or the KNN for classification of a given query sample.

safe_k [int (default = None)] The size of the indecision region.

IH_rate [float (default = 0.3)] Hardness threshold. If the hardness level of the competence region is lower than the IH_rate the KNN classifier is used. Otherwise, the DS algorithm is used for classification.

mode [String (Default = “selection”)] Whether the technique will perform dynamic selection, dynamic weighting or an hybrid approach for classification.

References

T.Woloszynski, M. Kurzynski, A probabilistic model of classifier competence for dynamic ensemble selection, Pattern Recognition 44 (2011) 2656–2668.

12. Rastrigin, R. Erenstein, Method of collective recognition, Vol. 595, 1981, (in Russian).

Britto, Alceu S., Robert Sabourin, and Luiz ES Oliveira. “Dynamic selection of classifiers—a comprehensive review.” Pattern Recognition 47.11 (2014): 3665-3680.

R. M. O. Cruz, R. Sabourin, and G. D. Cavalcanti, “Dynamic classifier selection: Recent advances and perspectives,” Information Fusion, vol. 41, pp. 195 – 216, 2018.

estimate_competence (query, predictions=None)

estimate the competence of each base classifier c_i using the source of competence C_{src} and the potential function model. The source of competence C_{src} for all data points in DSEL is already pre-computed in the fit() steps.

$$\delta_{i,j} = \frac{\sum_{k=1}^N C_{src} \exp(-d(\mathbf{x}_k, \mathbf{x}_q)^2)}{\exp(-d(\mathbf{x}_k, \mathbf{x}_q)^2)}$$

Parameters

query [array of shape = [n_samples, n_features]] The test examples.

predictions [array of shape = [n_samples, n_classifiers]] Predictions of the base classifiers for all test examples.

Returns

competences [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

fit (X, y)

Train the DS model by setting the KNN algorithm and pre-processing the information required to apply the DS methods. In the case of probabilistic techniques, the source of competence (C_src) is calculated for each data point in DSEL in order to speed up the process during the testing phases.

C_src is estimated with the source_competence() function that is overridden by each DS method based on this paradigm

Parameters

X [array of shape = [n_samples, n_features]] Data used to fit the model.

y [array of shape = [n_samples]] class labels of each example in X.

Returns

self

static potential_func (dist)

Gaussian potential function to decrease the influence of the source of competence as the distance between \mathbf{x}_k and the query \mathbf{x}_q increases. The function is computed using the following equation:

$$potential = \exp(-dist(\mathbf{x}_k, \mathbf{x}_q)^2)$$

where dist represents the Euclidean distance between \mathbf{x}_k and \mathbf{x}_q

Parameters

dist [array of shape = [self.n_samples]] distance between the corresponding sample to the query

Returns

The result of the potential function for each value in (dist)

select (competences)

Selects the base classifiers that obtained a competence level higher than the predefined threshold. In this case, the threshold indicates the competence of the random classifier.

Parameters

competences [array of shape = [n_samples, n_classifiers]] Competence level estimated for each base classifier and test example.

Returns

selected_classifiers [array of shape = [n_samples, n_classifiers]] Boolean matrix containing True if the base classifier is select, False otherwise.

source_competence ()

Method used to estimate the source of competence at each data point.

Each DS technique based on this paradigm should define its computation of C_src

Returns

C_src [array of shape = [n_samples, n_classifiers]] The competence source for each base classifier at each data point.

Randomized Reference Classifier (RRC)

class deslib.des.probabilistic.**RRC** (*pool_classifiers*, *k=None*, *DFP=False*, *with_IH=False*, *safe_k=None*, *IH_rate=0.3*, *mode='selection'*)
 DES technique based on the Randomized Reference Classifier method (DES-RRC).

Parameters

pool_classifiers [type, the generated_pool of classifiers trained for the corresponding classification problem.

pool_classifiers [list of classifiers] The generated_pool of classifiers trained for the corresponding classification problem. The classifiers should support methods “predict” and “predict_proba”.

k [int (Default = None)] Number of neighbors used to estimate the competence of the base classifiers. If *k* = None, the whole dynamic selection dataset is used, and the influence of each sample is based on its distance to the query.

DFP [Boolean (Default = False)] Determines if the dynamic frienemy pruning is applied.

with_IH [Boolean (Default = False)] Whether the hardness level of the region of competence is used to decide between using the DS algorithm or the KNN for classification of a given query sample.

safe_k [int (default = None)] The size of the indecision region.

IH_rate [float (default = 0.3)] Hardness threshold. If the hardness level of the competence region is lower than the *IH_rate* the KNN classifier is used. Otherwise, the DS algorithm is used for classification.

mode [String (Default = “selection”)] Whether the technique will perform dynamic selection, dynamic weighting or an hybrid approach for classification.

References

Woloszynski, Tomasz, and Marek Kurzynski. “A probabilistic model of classifier competence for dynamic ensemble selection.” *Pattern Recognition* 44.10 (2011): 2656-2668.

R. M. O. Cruz, R. Sabourin, and G. D. Cavalcanti, “Dynamic classifier selection: Recent advances and perspectives,” *Information Fusion*, vol. 41, pp. 195 – 216, 2018.

source_competence()

Calculates the source of competence using the randomized reference classifier (RRC) method.

The source of competence **C_src** at the validation point \mathbf{x}_k calculated using the probabilistic model based on the supports obtained by the base classifier and randomized reference classifier (RRC) model. The probabilistic modeling of the classifier competence is calculated using the *ccprmod* function.

Returns

C_src [array of shape = [n_samples, n_classifiers]] The competence source for each base classifier at each data point.

DES-Kullback Leibler

class `deslib.des.probabilistic.DESKL` (*pool_classifiers*, *k=None*, *DFP=False*, *with_IH=False*, *safe_k=None*, *IH_rate=0.3*, *mode='selection'*)

Dynamic Ensemble Selection-Kullback-Leibler divergence (DES-KL).

This method estimates the competence of the classifier from the information theory perspective. The competence of the base classifiers is calculated as the KL divergence between the vector of class supports produced by the base classifier and the outputs of a random classifier (RC). $RC = 1/L$, L being the number of classes in the problem. Classifiers with a competence higher than the competence of the random classifier is selected.

Parameters

pool_classifiers [list of classifiers] The generated_pool of classifiers trained for the corresponding classification problem. The classifiers should support methods “predict” and “predict_proba”.

k [int (Default = None)] Number of neighbors used to estimate the competence of the base classifiers. If $k = \text{None}$, the whole dynamic selection dataset is used, and the influence of each sample is based on its distance to the query.

DFP [Boolean (Default = False)] Determines if the dynamic frienemy pruning is applied.

with_IH [Boolean (Default = False)] Whether the hardness level of the region of competence is used to decide between using the DS algorithm or the KNN for classification of a given query sample.

safe_k [int (default = None)] The size of the indecision region.

IH_rate [float (default = 0.3)] Hardness threshold. If the hardness level of the competence region is lower than the *IH_rate* the KNN classifier is used. Otherwise, the DS algorithm is used for classification.

mode [String (Default = “selection”)] Whether the technique will perform dynamic selection, dynamic weighting or an hybrid approach for classification.

References

Woloszynski, Tomasz, et al. “A measure of competence based on random classification for dynamic ensemble selection.” *Information Fusion* 13.3 (2012): 207-213.

Woloszynski, Tomasz, and Marek Kurzynski. “A probabilistic model of classifier competence for dynamic ensemble selection.” *Pattern Recognition* 44.10 (2011): 2656-2668.

R. M. O. Cruz, R. Sabourin, and G. D. Cavalcanti, “Dynamic classifier selection: Recent advances and perspectives,” *Information Fusion*, vol. 41, pp. 195 – 216, 2018.

source_competence ()

Calculates the source of competence using the KL divergence method.

The source of competence C_{src} at the validation point \mathbf{x}_k is calculated by the KL divergence between the vector of class supports produced by the base classifier and the outputs of a random classifier (RC) $RC = 1/L$, L being the number of classes in the problem. The value of C_{src} is negative if the base classifier misclassified the instance \mathbf{x}_k .

Returns

C_src [array of shape = [n_samples, n_classifiers]] The competence source for each base classifier at each data point.

DES-Minimum Difference

```
class deslib.des.probabilistic.MinimumDifference (pool_classifiers, k=None,
                                                DFP=False, with_IH=False,
                                                safe_k=None, IH_rate=0.3,
                                                mode='selection')
```

Computes the competence level of the classifiers based on the difference between the support obtained by each class. The competence level at a data point \mathbf{x}_k is equal to the minimum difference between the support obtained to the correct class and the support obtained for different classes.

The influence of each sample \mathbf{x}_k is defined according to a Gaussian function model[2]. Samples that are closer to the query have a higher influence in the competence estimation.

Parameters

pool_classifiers [list of classifiers] The generated_pool of classifiers trained for the corresponding classification problem. The classifiers should support methods “predict” and “predict_proba”.

k [int (Default = None)] Number of neighbors used to estimate the competence of the base classifiers. If $k = \text{None}$, the whole dynamic selection dataset is used, and the influence of each sample is based on its distance to the query.

DFP [Boolean (Default = False)] Determines if the dynamic frienemy pruning is applied.

with_IH [Boolean (Default = False)] Whether the hardness level of the region of competence is used to decide between using the DS algorithm or the KNN for classification of a given query sample.

safe_k [int (default = None)] The size of the indecision region.

IH_rate [float (default = 0.3)] Hardness threshold. If the hardness level of the competence region is lower than the IH_rate the KNN classifier is used. Otherwise, the DS algorithm is used for classification.

mode [String (Default = “selection”)] Whether the technique will perform dynamic selection, dynamic weighting or an hybrid approach for classification.

References

[1] B. Antosik, M. Kurzynski, New measures of classifier competence – heuristics and application to the design of multiple classifier systems., in: Computer recognition systems 4., 2011, pp. 197–206.

[2] Woloszynski, Tomasz, and Marek Kurzynski. “A probabilistic model of classifier competence for dynamic ensemble selection.” Pattern Recognition 44.10 (2011): 2656-2668.

source_competence ()

Calculates the source of competence using the Minimum Difference method.

The source of competence C_{src} at the validation point \mathbf{x}_k calculated by the Minimum Difference between the supports obtained to the correct class and the support obtained by the other classes

Returns

C_{src} [array of shape = [n_samples, n_classifiers]] The competence source for each base classifier at each data point.

DES-Exponential

```
class deslib.des.probablistic.Exponential (pool_classifiers, k=None, DFP=False,  
                                           safe_k=None, with_IH=False, IH_rate=0.3,  
                                           mode='selection')
```

The source of competence C_{src} at the validation point \mathbf{x}_k is a product of two factors: The absolute value of the competence and the sign. The value of the source competence is inverse proportional to the normalized entropy of its supports vector. The sign of competence is simply determined by correct/incorrect classification of \mathbf{x}_k [1].

The influence of each sample \mathbf{x}_k is defined according to a Gaussian function model[2]. Samples that are closer to the query have a higher influence in the competence estimation.

Parameters

pool_classifiers [list of classifiers] The generated_pool of classifiers trained for the corresponding classification problem. The classifiers should support methods “predict” and “predict_proba”.

k [int (Default = None)] Number of neighbors used to estimate the competence of the base classifiers. If $k = \text{None}$, the whole dynamic selection dataset is used, and the influence of each sample is based on its distance to the query.

DFP [Boolean (Default = False)] Determines if the dynamic frienemy pruning is applied.

with_IH [Boolean (Default = False)] Whether the hardness level of the region of competence is used to decide between using the DS algorithm or the KNN for classification of a given query sample.

safe_k [int (default = None)] The size of the indecision region.

IH_rate [float (default = 0.3)] Hardness threshold. If the hardness level of the competence region is lower than the IH_rate the KNN classifier is used. Otherwise, the DS algorithm is used for classification.

mode [String (Default = “selection”)] Whether the technique will perform dynamic selection, dynamic weighting or an hybrid approach for classification.

References

[1] B. Antosik, M. Kurzynski, New measures of classifier competence – heuristics and application to the design of multiple classifier systems., in: Computer recognition systems 4., 2011, pp. 197–206.

[2] Woloszynski, Tomasz, and Marek Kurzynski. “A probabilistic model of classifier competence for dynamic ensemble selection.” Pattern Recognition 44.10 (2011): 2656-2668.

source_competence ()

The source of competence C_{src} at the validation point \mathbf{x}_k is a product of two factors: The absolute value of the competence and the sign. The value of the source competence is inverse proportional to the normalized entropy of its supports vector. The sign of competence is simply determined by correct/incorrect classification of the instance \mathbf{x}_k .

Returns

C_src [array of shape = [n_samples, n_classifiers]] The competence source for each base classifier at each data point.

DES-Logarithmic

```
class deslib.des.probabilistic.Logarithmic (pool_classifiers, k=None, DFP=False,  
                                           with_IH=False, safe_k=None, IH_rate=0.3,  
                                           mode='selection')
```

This method estimates the competence of the classifier based on the logarithmic difference between the supports obtained by the base classifier.

Parameters

- pool_classifiers** [list of classifiers] The generated_pool of classifiers trained for the corresponding classification problem. The classifiers should support methods “predict” and “predict_proba”.
- k** [int (Default = None)] Number of neighbors used to estimate the competence of the base classifiers. If k = None, the whole dynamic selection dataset is used, and the influence of each sample is based on its distance to the query.
- DFP** [Boolean (Default = False)] Determines if the dynamic frienemy pruning is applied.
- with_IH** [Boolean (Default = False)] Whether the hardness level of the region of competence is used to decide between using the DS algorithm or the KNN for classification of a given query sample.
- safe_k** [int (default = None)] The size of the indecision region.
- IH_rate** [float (default = 0.3)] Hardness threshold. If the hardness level of the competence region is lower than the IH_rate the KNN classifier is used. Otherwise, the DS algorithm is used for classification.
- mode** [String (Default = “selection”)] Whether the technique will perform dynamic selection, dynamic weighting or an hybrid approach for classification.

References

B. Antosik, M. Kurzynski, New measures of classifier competence – heuristics and application to the design of multiple classifier systems., in: Computer recognition systems 4., 2011, pp. 197–206.

T.Woloszynski, M. Kurzynski, A measure of competence based on randomized reference classifier for dynamic ensemble selection, in: International Conference on Pattern Recognition (ICPR), 2010, pp. 4194–4197.

source_competence ()

The source of competence C_{src} at the validation point x_k is calculated by logarithm function in the support obtained by the base classifier.

Returns

C_src [array of shape = [n_samples, n_classifiers]] The competence source for each base classifier at each data point.

3.2.3 Static ensembles

This module provides the implementation of static ensemble techniques that are usually used as a baseline for the comparison of DS methods: Single Best (SB), Static Selection (SS) and Oracle. The `deslib.static` provides a set of static ensemble methods which are often used as a baseline to compare the performance of dynamic selection algorithms.

Oracle

class `deslib.static.oracle.Oracle` (*pool_classifiers*)

Abstract method that always selects the base classifier that predicts the correct label if such classifier exists. This method is often used to measure the upper-limit performance that can be achieved by a dynamic classifier selection technique. It is used as a benchmark by several dynamic selection algorithms

Parameters

pool_classifiers [list of classifiers] The generated_pool of classifiers trained for the corresponding classification problem. The classifiers should support methods “predict”.

References

Kuncheva, Ludmila I. Combining pattern classifiers: methods and algorithms. John Wiley & Sons, 2004.

R. M. O. Cruz, R. Sabourin, and G. D. Cavalcanti, “Dynamic classifier selection: Recent advances and perspectives,” Information Fusion, vol. 41, pp. 195 – 216, 2018.

predict (*X*, *y*)

Prepare the labels using the Oracle model.

Parameters

X [array of shape = [n_samples, n_features]] The data to be classified

y [array of shape = [n_samples]] Class labels of each sample in X.

Returns

predicted_labels [array of shape = [n_samples]] Predicted class for each sample in X.

score (*X*, *y*)

Prepare the labels using the Oracle model.

Parameters

X [array of shape = [n_samples, n_features]] The data to be classified

y [array of shape = [n_samples]] Class labels of each sample in X.

Returns

accuracy [float] Classification accuracy of the Oracle model.

Single Best

class `deslib.static.single_best.SingleBest` (*pool_classifiers*)

Classification method that selects the classifier in the pool with highest score to be used for classification. Usually, the performance of the single best classifier is estimated based on the validation data.

Parameters

pool_classifiers [list of classifiers] The generated_pool of classifiers trained for the corresponding classification problem. The classifiers should support methods “predict”.

References

Britto, Alceu S., Robert Sabourin, and Luiz ES Oliveira. “Dynamic selection of classifiers—a comprehensive review.” *Pattern Recognition* 47.11 (2014): 3665-3680.

Kuncheva, Ludmila I. *Combining pattern classifiers: methods and algorithms*. John Wiley & Sons, 2004.

R. M. O. Cruz, R. Sabourin, and G. D. Cavalcanti, “Dynamic classifier selection: Recent advances and perspectives,” *Information Fusion*, vol. 41, pp. 195 – 216, 2018.

fit (*X*, *y*)

Fit the model by selecting the base classifier with the highest accuracy in the dataset. The single best classifier is kept in `self.best_clf` and its index is kept in `self.best_clf_index`.

Parameters

X [array of shape = [n_samples, n_features]] Data used to fit the model.

y [array of shape = [n_samples]] class labels of each example in X.

predict (*X*)

Predict the label of each sample in X and returns the predicted label.

Parameters

X [array of shape = [n_samples, n_features]] The data to be classified

Returns

predicted_labels [array of shape = [n_samples]] Predicted class for each sample in X.

predict_proba (*X*)

Estimates the posterior probabilities for each class for each sample in X. The returned probability estimates for all classes are ordered by the label of classes.

Parameters

X [array of shape = [n_samples, n_features]] The data to be classified

Returns

predicted_proba [array of shape = [n_samples, n_classes]] Posterior probabilities estimates for each class.

Static Selection

class `deslib.static.static_selection.StaticSelection` (*pool_classifiers*,
pct_classifiers=0.5)

Ensemble model that selects N classifiers with the best performance in a dataset

Parameters

pool_classifiers [list of classifiers] The generated_pool of classifiers trained for the corresponding classification problem. The classifiers should support methods “predict”.

pct_classifiers [float (Default = 0.5)] percentage of base classifier that should be selected by the selection scheme.

References

Britto, Alceu S., Robert Sabourin, and Luiz ES Oliveira. “Dynamic selection of classifiers—a comprehensive review.” *Pattern Recognition* 47.11 (2014): 3665-3680.

Kuncheva, Ludmila I. Combining pattern classifiers: methods and algorithms. John Wiley & Sons, 2004.

R. M. O. Cruz, R. Sabourin, and G. D. Cavalcanti, “Dynamic classifier selection: Recent advances and perspectives,” Information Fusion, vol. 41, pp. 195 – 216, 2018.

fit (*X*, *y*)

Fit the static selection model by select an ensemble of classifier containing the base classifiers with highest accuracy in the given dataset.

Parameters

X [array of shape = [n_samples, n_features]] Data used to fit the model.

y [array of shape = [n_samples]] class labels of each example in X.

predict (*X*)

Predict the label of each sample in X and returns the predicted label.

Parameters

X [array of shape = [n_samples, n_features]] The data to be classified

Returns

predicted_labels [array of shape = [n_samples]] Predicted class for each sample in X.

3.2.4 Utils

Utility functions for ensemble methods such as diversity and aggregation methods. The `deslib.util` This module includes various utilities. They are divided into four parts:

`deslib.util.aggregation` - Implementation of aggregation functions such as majority voting and averaging. Such functions can be applied to any list of classifiers.

`deslib.util.diversity` - Implementation of different measures of diversity between classifiers.

`deslib.util.prob_functions` - Functions to estimate the competence of a base classifier based on the probability estimates.

`deslib.util.instance_hardness` - Functions to measure the hardness level of a given instance

Diversity

This file contains the implementation of key diversity measures found in the ensemble literature:

- Double Fault
- Negative Double fault
- Q-statistics
- Ratio of errors

The implementation are made according to the specifications from the book “Combining Pattern Classifiers”.

`deslib.util.diversity.Q_statistic` (*y*, *y_pred1*, *y_pred2*)

Calculates the Q-statistics diversity measure between a pair of classifiers. The Q value is in a range [-1, 1]. Classifiers that tend to classify the same object correctly will have positive values of Q, and Q = 0 for two independent classifiers.

Parameters

y [array of shape = [n_samples]:] class labels of each sample.

y_pred1 [array of shape = [n_samples]:] predicted class labels by the classifier 1 for each sample.

y_pred2 [array of shape = [n_samples]:] predicted class labels by the classifier 2 for each sample.

Returns

Q [The q-statistic measure between two classifiers]

`deslib.util.diversity.agreement_measure(y, y_pred1, y_pred2)`

Calculates the agreement measure between a pair of classifiers. This measure is calculated by the frequency that both classifiers either obtained the correct or incorrect prediction for any given sample

Parameters

y [array of shape = [n_samples]:] class labels of each sample.

y_pred1 [array of shape = [n_samples]:] predicted class labels by the classifier 1 for each sample.

y_pred2 [array of shape = [n_samples]:] predicted class labels by the classifier 2 for each sample.

Returns

agreement [The frequency at which both classifiers agrees]

`deslib.util.diversity.compute_pairwise_diversity(targets, prediction_matrix, diversity_func)`

Computes the pairwise diversity matrix.

Parameters

targets [array of shape = [n_samples]:] Class labels of each sample in X.

prediction_matrix [array of shape = [n_samples, n_classifiers]:] Predicted class labels for each classifier in the pool

diversity_func [Function used to estimate the pairwise diversity]

Returns

diversity [array of shape = [n_classifiers]] The average pairwise diversity matrix calculated for the pool of classifiers

`deslib.util.diversity.correlation_coefficient(y, y_pred1, y_pred2)`

Calculates the correlation between two classifiers using oracle outputs. coefficient is a value in a range [-1, 1].

Parameters

y [array of shape = [n_samples]:] class labels of each sample.

y_pred1 [array of shape = [n_samples]:] predicted class labels by the classifier 1 for each sample.

y_pred2 [array of shape = [n_samples]:] predicted class labels by the classifier 2 for each sample.

Returns

rho [The correlation coefficient measured between two classifiers]

`deslib.util.diversity.disagreement_measure(y, y_pred1, y_pred2)`

Calculates the disagreement measure between a pair of classifiers. This measure is calculated by the frequency that only one classifier makes the correct prediction.

Parameters

y [array of shape = [n_samples]:] class labels of each sample.

y_pred1 [array of shape = [n_samples]:] predicted class labels by the classifier 1 for each sample.

y_pred2 [array of shape = [n_samples]:] predicted class labels by the classifier 2 for each sample.

Returns

disagreement [The frequency at which both classifiers disagrees]

`deslib.util.diversity.double_fault(y, y_pred1, y_pred2)`

Calculates the double fault (df) measure. This measure represents the probability that both classifiers makes the wrong prediction. A lower value of df means the base classifiers are less likely to make the same error. This measure must be minimized to increase diversity.

Parameters

y [array of shape = [n_samples]:] class labels of each sample.

y_pred1 [array of shape = [n_samples]:] predicted class labels by the classifier 1 for each sample.

y_pred2 [array of shape = [n_samples]:] predicted class labels by the classifier 2 for each sample.

Returns

df [The double fault measure between two classifiers]

References

Giacinto, Giorgio, and Fabio Roli. "Design of effective neural network ensembles for image classification purposes." Image and Vision Computing 19.9 (2001): 699-707.

`deslib.util.diversity.negative_double_fault(y, y_pred1, y_pred2)`

The negative of the double fault measure. This measure should be maximized for a higher diversity.

Parameters

y [array of shape = [n_samples]:] class labels of each sample.

y_pred1 [array of shape = [n_samples]:] predicted class labels by the classifier 1 for each sample.

y_pred2 [array of shape = [n_samples]:] predicted class labels by the classifier 2 for each sample.

Returns

df [The negative double fault measure between two classifiers]

References

Giacinto, Giorgio, and Fabio Roli. “Design of effective neural network ensembles for image classification purposes.” *Image and Vision Computing* 19.9 (2001): 699-707.

`deslib.util.diversity.ratio_errors(y, y_pred1, y_pred2)`

Calculates Ratio of errors diversity measure between a pair of classifiers. A higher value means that the base classifiers are less likely to make the same errors. The ratio must be maximized for a higher diversity.

Parameters

y [array of shape = [n_samples]:] class labels of each sample.

y_pred1 [array of shape = [n_samples]:] predicted class labels by the classifier 1 for each sample.

y_pred2 [array of shape = [n_samples]:] predicted class labels by the classifier 2 for each sample.

Returns

ratio [The q-statistic measure between two classifiers]

References

Aksela, Matti. “Comparison of classifier selection methods for improving committee performance.” *Multiple Classifier Systems* (2003): 159-159.

Aggregation

This file contains the implementation of different aggregation functions to combine the outputs of the base classifiers to give the final decision.

`deslib.util.aggregation.average_combiner(classifier_ensemble, X)`

Ensemble combination using the Average rule.

Parameters

classifier_ensemble [list of shape = [n_classifiers]] containing the ensemble of classifiers used in the aggregation scheme.

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_label [array of shape = [n_samples]] The label of each query sample predicted using the majority voting rule

`deslib.util.aggregation.average_rule(predictions)`

Apply the average fusion rule to the predicted vector of class supports (predictions).

Parameters

predictions [np array of shape = [n_samples, n_classifiers, n_classes]] vector of class supports predicted by each base classifier for sample

Returns

predicted_label [array of shape = [n_samples]] The label of each query sample predicted using the majority voting rule

`deslib.util.aggregation.majority_voting(classifier_ensemble, X)`

Apply the majority voting rule to predict the label of each sample in X.

Parameters

classifier_ensemble [list of shape = [n_classifiers]] containing the ensemble of classifiers used in the aggregation scheme.

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_label [array of shape = [n_samples]] The label of each query sample predicted using the majority voting rule

`deslib.util.aggregation.majority_voting_rule(votes)`

Applies the majority voting rule to the estimated votes.

Parameters

votes [array of shape = [n_samples, n_classifiers],] The votes obtained by each classifier for each sample.

Returns

predicted_label [array of shape = [n_samples]] The label of each query sample predicted using the majority voting rule

`deslib.util.aggregation.maximum_combiner(classifier_ensemble, X)`

Ensemble combination using the Maximum rule.

Parameters

classifier_ensemble [list of shape = [n_classifiers]] containing the ensemble of classifiers used in the aggregation scheme.

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_label [array of shape = [n_samples]] The label of each query sample predicted using the majority voting rule

`deslib.util.aggregation.maximum_rule(predictions)`

Apply the product fusion rule to the predicted vector of class supports (predictions).

Parameters

predictions [np array of shape = [n_samples, n_classifiers, n_classes]] vector of class supports predicted by each base classifier for sample

Returns

predicted_label [array of shape = [n_samples]] The label of each query sample predicted using the majority voting rule

`deslib.util.aggregation.median_combiner(classifier_ensemble, X)`

Ensemble combination using the Median rule.

Parameters

classifier_ensemble [list of shape = [n_classifiers]] containing the ensemble of classifiers used in the aggregation scheme.

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_label [array of shape = [n_samples]] The label of each query sample predicted using the majority voting rule

`deslib.util.aggregation.median_rule(predictions)`

Apply the product fusion rule to the predicted vector of class supports (predictions).

Parameters

predictions [np array of shape = [n_samples, n_classifiers, n_classes]] vector of class supports predicted by each base classifier for sample

Returns

predicted_label [array of shape = [n_samples]] The label of each query sample predicted using the majority voting rule

`deslib.util.aggregation.minimum_combiner(classifier_ensemble, X)`

Ensemble combination using the Minimum rule.

Parameters

classifier_ensemble [list of shape = [n_classifiers]] containing the ensemble of classifiers used in the aggregation scheme.

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_label [array of shape = [n_samples]] The label of each query sample predicted using the majority voting rule

`deslib.util.aggregation.minimum_rule(predictions)`

Apply the product fusion rule to the predicted vector of class supports (predictions).

Parameters

predictions [np array of shape = [n_samples, n_classifiers, n_classes]] vector of class supports predicted by each base classifier for sample

Returns

list_proba [array of shape = [n_classifiers, n_samples, n_classes]] probabilities predicted by each base classifier in the ensemble for all samples in X.

`deslib.util.aggregation.predict_proba_ensemble(classifier_ensemble, X)`

Estimates the posterior probabilities of the give ensemble for each sample in X.

Parameters

classifier_ensemble [list of shape = [n_classifiers]] containing the ensemble of classifiers used in the aggregation scheme.

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_proba [array of shape = [n_samples, n_classes]] posterior probabilities estimates for each samples in X.

`deslib.util.aggregation.predict_proba_ensemble_weighted(classifier_ensemble, weights, X)`

Estimates the posterior probabilities for each sample in X.

Parameters

classifier_ensemble [list of shape = [n_classifiers]] containing the ensemble of classifiers used to estimate the probabilities.

weights [array of shape = [n_samples, n_classifiers]] Weights associated to each base classifier for each sample

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_proba [array of shape = [n_samples, n_classes]] posterior probabilities estimates for each samples in X.

`deslib.util.aggregation.product_combiner(classifier_ensemble, X)`
Ensemble combination using the Product rule.

Parameters

classifier_ensemble [list of shape = [n_classifiers]] containing the ensemble of classifiers used in the aggregation scheme.

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_label [array of shape = [n_classifiers, n_samples, n_classes]] probabilities predicted by each base classifier in the ensemble for all samples in X.

`deslib.util.aggregation.product_rule(predictions)`
Apply the product fusion rule to the predicted vector of class supports (predictions).

Parameters

predictions [array of shape = [n_samples, n_classifiers, n_classes]] vector of class supports predicted by each base classifier for sample

Returns

predicted_label [array of shape = [n_samples]] The label of each query sample predicted using the majority voting rule

`deslib.util.aggregation.weighted_majority_voting(classifier_ensemble, weights, X)`
Apply the weighted majority voting rule to predict the label of each sample in X. The size of the weights vector should be equal to the size of the ensemble.

Parameters

classifier_ensemble [list of shape = [n_classifiers]] containing the ensemble of classifiers used in the aggregation scheme.

weights [array of shape = [n_samples, n_classifiers]] Weights associated to each base classifier for each sample

X [array of shape = [n_samples, n_features]] The input data.

Returns

predicted_label [array of shape = [n_samples]] The label of each query sample predicted using the majority voting rule

`deslib.util.aggregation.weighted_majority_voting_rule(votes, weights, labels_set=None)`

Applies the weighted majority voting rule based on the votes obtained by each base classifier and their respective weights.

Parameters

votes [array of shape = [n_samples, n_classifiers],] The votes obtained by each classifier for each sample.

weights [array of shape = [n_samples, n_classifiers]] Weights associated to each base classifier for each sample

labels_set [(Default=None) set with the possible classes in the problem]

Returns

predicted_label [array of shape = [n_samples]] The label of each query sample predicted using the majority voting rule

Probabilistic Functions

This file contains the implementation of several functions used to estimate the competence level of a base classifiers based on posterior probabilities predicted for each class.

`deslib.util.prob_functions.ccprmod(supports, idx_correct_label, B=20)`

Python implementation of the `ccprmod.m` (Classifier competence based on probabilistic modelling) function. Matlab code is available at: <http://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/28391/versions/6/previews/ccprmod.m/index.html>

Parameters

supports: array of shape = [n_samples, n_classes] containing the supports obtained by the base classifier for each class.

idx_correct_label: array of shape = [n_samples] containing the index of the correct class.

B [int (Default = 20)] number of points used in the calculation of the competence, higher values result in a more accurate estimation.

Returns

C_src [array of shape = [n_samples]] representing the classifier competences at each data point

References

T.Woloszynski, M. Kurzynski, A probabilistic model of classifier competence for dynamic ensemble selection, Pattern Recognition 44 (2011) 2656–2668.

Examples

```
>>> supports = [[0.3, 0.6, 0.1], [1.0/3, 1.0/3, 1.0/3]]
>>> idx_correct_label = [1, 0]
>>> ccprmod(supports, idx_correct_label)
ans = [0.784953394056843, 0.332872292262951]
```

`deslib.util.prob_functions.entropy_func(n_classes, supports, is_correct)`

Calculate the entropy in the support obtained by the base classifier. The value of the source competence is inverse proportional to the normalized entropy of its supports vector and the sign of competence is simply determined by the correct/incorrect classification.

Parameters

n_classes [int] The number of classes in the problem

supports: array of shape = [n_samples, n_classes] containing the supports obtained by the base classifier for each class.

is_correct: array of shape = [n_samples] array with 1 whether the base classifier predicted the correct label and -1 otherwise

Returns

C_src [array of shape = [n_samples]] representing the classifier competences at each data point

References

B. Antosik, M. Kurzynski, New measures of classifier competence – heuristics and application to the design of multiple classifier systems., in: Computer recognition systems 4., 2011, pp. 197–206.

`deslib.util.prob_functions.exponential_func(n_classes, support_correct)`

Calculate the exponential function based on the support obtained by the base classifier for the correct class label.

Parameters

n_classes [int] The number of classes in the problem

support_correct: array of shape = [n_samples] containing the supports obtained by the base classifier for the correct class

Returns

C_src [array of shape = [n_samples]] representing the classifier competences at each data point

`deslib.util.prob_functions.log_func(n_classes, support_correct)`

Calculate the logarithm in the support obtained by the base classifier.

Parameters

n_classes [int] The number of classes in the problem

support_correct: array of shape = [n_samples] containing the supports obtained by the base classifier for the correct class

Returns

C_src [array of shape = [n_samples]] representing the classifier competences at each data point

References

T.Woloszynski, M. Kurzynski, A measure of competence based on randomized reference classifier for dynamic ensemble selection, in: International Conference on Pattern Recognition (ICPR), 2010, pp. 4194–4197.

`deslib.util.prob_functions.min_difference(supports, idx_correct_label)`

The minimum difference between the supports obtained for the correct class and the vector of class supports. The value of the source competence is negative if the sample is misclassified and positive otherwise.

Parameters

supports: array of shape = [n_samples, n_classes] containing the supports obtained by the base classifier for each class

idx_correct_label: array of shape = [n_samples] containing the index of the correct class

Returns

C_src [array of shape = [n_samples]] representing the classifier competences at each data point

References

B. Antosik, M. Kurzynski, New measures of classifier competence – heuristics and application to the design of multiple classifier systems., in: Computer recognition systems 4., 2011, pp. 197–206.

`deslib.util.prob_functions.softmax(w, theta=1.0)`

Takes an vector `w` of `S` `N`-element and returns a vectors where each column of the vector sums to 1, with elements exponentially proportional to the respective elements in `N`.

Parameters

w [array of shape = [N, M]]

theta [float (default = 1.0)] used as a multiplier prior to exponentiation.

Returns

dist [array of shape = [N, M]] which the sum of each row sums to 1 and the elements are exponentially proportional to the respective elements in `N`

Instance Hardness

This file contains the implementation of different measures of instance hardness.

`deslib.util.instance_hardness.hardness_region_competence(neighbors_idx, labels, safe_k)`

Calculate the Instance hardness of the sample based on its neighborhood. The sample is deemed hard to classify when there is overlap between different classes in the region of competence. This method does not takes into account the target label of the test sample

This hardness measure is used to select whether use DS or use the KNN for the classification of a given query sample

Parameters

neighbors_idx [array of shape = [n_samples_test, k]] Indices of the nearest neighbors for each considered sample

labels [array of shape = [n_samples_train]] labels associated with each training sample

safe_k [int] Number of neighbors used to estimate the hardness of the corresponding region

Returns

hardness [array of shape = [n_samples]] The Hardness level associated with each example.

References

Smith, M.R., Martinez, T. and Giraud-Carrier, C., 2014. An instance level analysis of data complexity. Machine learning, 95(2), pp.225-256

3.3 General examples

Examples showing how to use different aspect of the library

3.3.1 Example Random Forest

In this example we use a pool of classifiers generated using the Random Forest method rather than Bagging. We also show how to change the size of the region of competence, used to estimate the local competence of the base classifiers.

This demonstrates that the library accepts any kind of base classifiers as long as they implement the predict and predict_proba functions. Moreover, any ensemble generation method such as Boosting or Rotation Trees can be used to generate a pool containing diverse base classifiers. We also included the performance of the RandomForest classifier as a baseline comparison.

```
# Example of a dcs techniques
from deslib.dcs.ola import OLA
from deslib.dcs.mcb import MCB
from deslib.des.des_p import DESP
from deslib.des.knora_u import KNORAU
from deslib.des.meta_des import METADES
from sklearn.datasets import load_breast_cancer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

# Example of a des techniques
from deslib.des.knora_e import KNORAE

if __name__ == "__main__":

    # Generate a classification dataset
    data = load_breast_cancer()
    X = data.data
    y = data.target

    # split the data into training and test data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
    RF = RandomForestClassifier()
    RF.fit(X_train, y_train)

    X_train, X_dsel, y_train, y_dsel = train_test_split(X, y, test_size=0.50)

    # Training a random forest to be used as the pool of classifiers. We set the
    ↪ maximum depth of the tree so that it
    # can estimate probabilities
    pool_classifiers = RandomForestClassifier(n_estimators=10, max_depth=5)
    pool_classifiers.fit(X_train, y_train)

    # Initialize a DS technique. Here we specify the size of the region of competence,
    ↪ (5 neighbors)
    knorau = KNORAU(pool_classifiers)
    kne = KNORAE(pool_classifiers, k=5)
    desp = DESP(pool_classifiers, k=5)
    ola = OLA(pool_classifiers, k=5)
    mcb = MCB(pool_classifiers, k=5)
    meta = METADES(pool_classifiers, k=5)

    # Fit the DS techniques
    knorau.fit(X_dsel, y_dsel)
    kne.fit(X_dsel, y_dsel)
    desp.fit(X_dsel, y_dsel)
    meta.fit(X_dsel, y_dsel)
    ola.fit(X_dsel, y_dsel)
    mcb.fit(X_dsel, y_dsel)
```

```

# Calculate classification accuracy of each technique
print('Classification accuracy RF: ', RF.score(X_test, y_test))
print('Evaluating DS techniques:')
print('Classification accuracy KNORAU: ', knorau.score(X_test, y_test))
print('Classification accuracy KNORA-Eliminate: ', kne.score(X_test, y_test))
print('Classification accuracy DESP: ', desp.score(X_test, y_test))
print('Classification accuracy OLA: ', ola.score(X_test, y_test))
print('Classification accuracy MCB: ', mcb.score(X_test, y_test))
print('Classification accuracy META-DES: ', meta.score(X_test, y_test))

```

Total running time of the script: (0 minutes 0.000 seconds)

3.3.2 Example Bagging

In this example we show how to apply different DCS and DES techniques for a classification dataset.

A very important aspect in dynamic selection is the generation of a pool of classifiers. A common practice in the dynamic selection literature is to use the Bagging (Bootstrap Aggregating) method to generate a pool containing base classifiers that are both diverse and informative.

In this example we generate a pool of classifiers using the Bagging technique implemented on the Scikit-learn library. Then, we compare the results obtained by combining this pool of classifiers using the standard Bagging combination approach versus the application of dynamic selection technique to select the set of most competent classifiers

```

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
from sklearn.calibration import CalibratedClassifierCV
from sklearn.ensemble import BaggingClassifier

from deslib.dcs.ola import OLA
from deslib.dcs.a_priori import APriori
from deslib.dcs.mcb import MCB
from deslib.des.des_p import DESP
from deslib.des.knora_u import KNORAU
from deslib.des.knora_e import KNORAE
from deslib.des.meta_des import METADES

if __name__ == "__main__":
    # Generate a classification dataset
    data = load_breast_cancer()
    X = data.data
    y = data.target
    # split the data into training and test data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)

    # Scale the variables to have 0 mean and unit variance
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Split the data into training and DSEL for DS techniques
    X_train, X_dsel, y_train, y_dsel = train_test_split(X_train, y_train, test_size=0.
→5)
    # Considering a pool composed of 10 base classifiers

```

```

# Calibrating Perceptrons to estimate probabilities
model = CalibratedClassifierCV(Perceptron(max_iter=100))

# Train a pool of 10 classifiers
pool_classifiers = BaggingClassifier(model, n_estimators=100)
pool_classifiers.fit(X_train, y_train)

# Initialize the DS techniques
knorau = KNORAU(pool_classifiers)
kne = KNORAE(pool_classifiers)
desp = DESP(pool_classifiers)
ola = OLA(pool_classifiers)
mcb = MCB(pool_classifiers)
apriori = APriori(pool_classifiers)
meta = METADES(pool_classifiers)

# Fit the des techniques
knorau.fit(X_dsel, y_dsel)
kne.fit(X_dsel, y_dsel)
desp.fit(X_dsel, y_dsel)

# Fit the dcs techniques
ola.fit(X_dsel, y_dsel)
mcb.fit(X_dsel, y_dsel)
apriori.fit(X_dsel, y_dsel)
meta.fit(X_dsel, y_dsel)

# Calculate classification accuracy of each technique
print('Evaluating DS techniques:')
print('Classification accuracy KNORA-Union: ', knorau.score(X_test, y_test))
print('Classification accuracy KNORA-Eliminate: ', kne.score(X_test, y_test))
print('Classification accuracy DESP: ', desp.score(X_test, y_test))
print('Classification accuracy OLA: ', ola.score(X_test, y_test))
print('Classification accuracy A priori: ', apriori.score(X_test, y_test))
print('Classification accuracy MCB: ', mcb.score(X_test, y_test))
print('Classification accuracy META-DES: ', meta.score(X_test, y_test))

```

Total running time of the script: (0 minutes 0.000 seconds)

3.3.3 Example Heterogeneous

In this example we show that the framework can also be used using different classifier models in the pool of classifiers. Such pool of classifiers are called Heterogeneous.

Here we consider a pool of classifiers composed of a Gaussian Naive Bayes, Perceptron, k-NN, Decision tree Linear SVM and Gaussian SVM

```

# Importing dynamic selection techniques:
from deslib.dcs.a_posteriori import APosteriori
from deslib.dcs.mcb import MCB
from deslib.dcs.lca import LCA
from deslib.des.probabilistic import RRC
from deslib.des.knop import KNOP
from deslib.des.knora_e import KNORAE

# Base classifier models:

```

```

from sklearn.linear_model import Perceptron
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.calibration import CalibratedClassifierCV

# Importing dataset
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler

if __name__ == "__main__":
    # Generate a classification dataset
    data = load_breast_cancer()
    X = data.data
    y = data.target

    # split the data into training and test data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)

    # Scale the variables to have 0 mean and unit variance
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Split the data into training and DSEL for DS techniques
    X_train, X_dsel, y_train, y_dsel = train_test_split(X, y, test_size=0.5)

    model_perceptron = CalibratedClassifierCV(Perceptron(max_iter=100)).fit(X_train,
↪y_train)
    model_linear_svm = CalibratedClassifierCV(LinearSVC()).fit(X_train, y_train)
    model_svc = SVC(probability=True).fit(X_train, y_train)
    model_bayes = GaussianNB().fit(X_train, y_train)
    model_tree = DecisionTreeClassifier().fit(X_train, y_train)
    model_knn = KNeighborsClassifier(n_neighbors=5).fit(X_train, y_train)
    pool_classifiers = [model_perceptron, model_linear_svm, model_svc, model_bayes,
↪model_tree, model_knn]

    # Initializing the DS techniques
    knop = KNOP(pool_classifiers)
    rrc = RRC(pool_classifiers)
    lca = LCA(pool_classifiers)
    mcb = MCB(pool_classifiers)
    aposteriori = APosteriori(pool_classifiers)

    # Fitting the techniques
    knop.fit(X_dsel, y_dsel)
    rrc.fit(X_dsel, y_dsel)
    lca.fit(X_dsel, y_dsel)
    mcb.fit(X_dsel, y_dsel)
    aposteriori.fit(X_dsel, y_dsel)

    # Calculate classification accuracy of each technique
    print('Evaluating DS techniques:')
    print('Classification accuracy KNOP: ', knop.score(X_test, y_test))

```

```
print('Classification accuracy RRC: ', rrc.score(X_test, y_test))
print('Classification accuracy LCA: ', lca.score(X_test, y_test))
print('Classification accuracy A posteriori: ', aposteriori.score(X_test, y_test))
```

Total running time of the script: (0 minutes 0.000 seconds)

3.3.4 Example DFP

In this example we show how to apply the dynamic frienemy pruning (DFP) to different dynamic selection techniques.

The DFP method is an online pruning model which analyzes the region of competence to know if it is composed of samples from different classes (indecision region). Then, it remove the base classifiers that do not correctly classifies at least a pair of samples coming from different classes (i.e., the base classifiers that do not cross the local region).

The DFP is shown to significantly improve the performance of several dynamic selection algorithms when dealing with heavily imbalanced problems, as it avoids the classifiers that are biased towards the majority class in predicting the label for the query.

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

# Example of dcs techniques:
from deslib.dcs.a_posteriori import APosteriori
from deslib.dcs.lca import LCA
from deslib.dcs.ola import OLA
from deslib.dcs.a_priori import APriori

# Example of des techniques:
from deslib.des.meta_des import METADES
from deslib.des.des_p import DESP

rng = np.random.RandomState(654321)

# Generate a classification dataset
X, y = make_classification(n_classes=2, n_samples=1000, weights=[0.2, 0.8], random_
    ↳state=rng)
# split the data into training and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_
    ↳state=rng)

# Split the data into training and DSEL for DS techniques
X_train, X_dsel, y_train, y_dsel = train_test_split(X_train, y_train, test_size=0.5,
    ↳random_state=rng)
# Considering a pool composed of 10 base classifiers
pool_classifiers = RandomForestClassifier(n_estimators=10, random_state=rng, max_
    ↳depth=10)
pool_classifiers.fit(X_train, y_train)

# DS techniques without DFP
apriori = APriori(pool_classifiers)
aposteriori = APosteriori(pool_classifiers)
ola = OLA(pool_classifiers)
lca = LCA(pool_classifiers)
desp = DESP(pool_classifiers)
```



```

meta = METADES(pool_classifiers)

apriori.fit(X_dsel, y_dsel)
aposteriori.fit(X_dsel, y_dsel)
ola.fit(X_dsel, y_dsel)
lca.fit(X_dsel, y_dsel)
desp.fit(X_dsel, y_dsel)
meta.fit(X_dsel, y_dsel)

print('Evaluating DS techniques:')
print('Classification accuracy of OLA: ', ola.score(X_test, y_test))
print('Classification accuracy of LCA: ', lca.score(X_test, y_test))
print('Classification accuracy of A priori: ', apriori.score(X_test, y_test))
print('Classification accuracy of A posteriori: ', aposteriori.score(X_test, y_test))
print('Classification accuracy of DES-P: ', desp.score(X_test, y_test))
print('Classification accuracy of META-DES: ', meta.score(X_test, y_test))

# Testing fire:
fire_apriori = APriori(pool_classifiers, DFP=True)
fire_aposteriori = APosteriori(pool_classifiers, DFP=True)
fire_ola = OLA(pool_classifiers, DFP=True)
fire_lca = LCA(pool_classifiers, DFP=True)
fire_desp = DESP(pool_classifiers, DFP=True)
fire_meta = METADES(pool_classifiers, DFP=True)

fire_apriori.fit(X_dsel, y_dsel)
fire_aposteriori.fit(X_dsel, y_dsel)
fire_ola .fit(X_dsel, y_dsel)
fire_lca.fit(X_dsel, y_dsel)
fire_desp.fit(X_dsel, y_dsel)
fire_meta.fit(X_dsel, y_dsel)

print('Evaluating FIRE-DS techniques:')
print('Classification accuracy of FIRE-OLA: ', fire_ola.score(X_test, y_test))
print('Classification accuracy of FIRE-LCA: ', fire_lca.score(X_test, y_test))
print('Classification accuracy of FIRE-A priori: ', fire_apriori.score(X_test, y_
↪test))
print('Classification accuracy of FIRE-A posteriori: ', aposteriori.score(X_test, y_
↪test))
print('Classification accuracy of FIRE-DES-P: ', fire_desp.score(X_test, y_test))
print('Classification accuracy of FIRE-META-DES: ', fire_meta.score(X_test, y_test))

```

Total running time of the script: (0 minutes 0.000 seconds)

3.4 Release history

3.4.1 Version 0.2

- Second release of the stable API. By [Rafael M O Cruz](#) and [Luiz G Hafemann](#).

Changes

- Implemented Label Encoding: labels are no longer required to be integers starting from 0. Categorical (strings) and non-sequential integers are supported (similarly to scikit-learn).

- Batch processing: Vectorized implementation of predictions. Large speed-up in computation time (100x faster in some cases).
- Predict_proba: only required (in the base estimators) if using methods that rely on probabilities (or if requesting probabilities from the ensemble).
- Improved documentation: Included additional examples, a step-by-step tutorial on how to use the library.
- New integration tests: Now covering predict_proba, IH and DFP.
- Bug fixes on 1) predict_proba 2) KNOP with DFP.

3.4.2 Version 0.1

API

- First release of the stable API. By [Rafael M O Cruz](#) and [Luiz G Hafemann](#).

Implemented methods:

- **DES techniques currently available are:**
 1. META-DES
 2. K-Nearest-Oracle-Eliminate (KNORA-E)
 3. K-Nearest-Oracle-Union (KNORA-U)
 4. Dynamic Ensemble Selection-Performance(DES-P)
 5. K-Nearest-Output Profiles (KNOP)
 6. Randomized Reference Classifier (DES-RRC)
 7. DES Kullback-Leibler Divergence (DES-KL)
 8. DES-Exponential
 9. DES-Logarithmic
 10. DES-Minimum Difference
 11. DES-Clustering
 12. DES-KNN
- **DCS techniques:**
 1. Modified Classifier Rank (Rank)
 2. Overall Local Accuracy (OLA)
 3. Local Class Accuracy (LCA)
 4. Modified Local Accuracy (MLA)
 5. Multiple Classifier Behaviour (MCB)
 6. A Priori Selection (A Priori)
 7. A Posteriori Selection (A Posteriori)
- **Baseline methods:**
 1. Oracle

2. Single Best
 3. Static Selection
- Dynamic Frienemy Prunning (DFP)
 - Diversity measures
 - Aggregation functions

Version 0.1

API

- First release of the stable API. By [Rafael M O Cruz](#) and [Luiz G Hafemann](#).

Implemented methods:

- **DES techniques currently available are:**
 1. META-DES
 2. K-Nearest-Oracle-Eliminate (KNORA-E)
 3. K-Nearest-Oracle-Union (KNORA-U)
 4. Dynamic Ensemble Selection-Performance(DES-P)
 5. K-Nearest-Output Profiles (KNOP)
 6. Randomized Reference Classifier (DES-RRC)
 7. DES Kullback-Leibler Divergence (DES-KL)
 8. DES-Exponential
 9. DES-Logarithmic
 10. DES-Minimum Difference
 11. DES-Clustering
 12. DES-KNN
- **DCS techniques:**
 1. Modified Classifier Rank (Rank)
 2. Overall Locall Accuracy (OLA)
 3. Local Class Accuracy (LCA)
 4. Modified Local Accuracy (MLA)
 5. Multiple Classifier Behaviour (MCB)
 6. A Priori Selection (A Priori)
 7. A Posteriori Selection (A Posteriori)
- **Baseline methods:**
 1. Oracle
 2. Single Best

3. Static Selection

- Dynamic Frienemy Prunning (DFP)
- Diversity measures
- Aggregation functions

Version 0.2

- Second release of the stable API. By [Rafael M O Cruz](#) and [Luiz G Hafemann](#).

Changes

- Implemented Label Encoding: labels are no longer required to be integers starting from 0. Categorical (strings) and non-sequential integers are supported (similarly to scikit-learn).
- Batch processing: Vectorized implementation of predictions. Large speed-up in computation time (100x faster in some cases).
- Predict proba: only required (in the base estimators) if using methods that rely on probabilities (or if requesting probabilities from the ensemble).
- Improved documentation: Included additional examples, a step-by-step tutorial on how to use the library.
- New integration tests: Now covering predict_proba, IH and DFP.
- Bug fixes on 1) predict_proba 2) KNOP with DFP.

CHAPTER 4

Example

Here we present an example of the KNORA-E techniques using a random forest to generate the pool of classifiers:

```
from sklearn.ensemble import RandomForestClassifier
from deslib.des.knora_e import KNORAE

# Train a pool of 10 classifiers
pool_classifiers = RandomForestClassifier(n_estimators=10)
pool_classifiers.fit(X_train, y_train)

# Initialize the DES model
knorae = KNORAE(pool_classifiers)

# Preprocess the Dynamic Selection dataset (DSEL)
knorae.fit(X_dsel, y_dsel)

# Predict new examples:
knorae.predict(X_test)
```

The library accepts any list of classifiers (from scikit-learn) as input, including a list containing different classifier models (heterogeneous ensembles). More examples to use the API can be found in the [examples page](#).

If you use DESLib in a scientific paper, please consider citing the following paper:

Rafael M. O. Cruz, Luiz G. Hafemann, Robert Sabourin and George D. C. Cavalcanti **DESlib: A Dynamic ensemble selection library in Python.** arXiv preprint arXiv:1802.04967 (2018).

```
@article{cruz_deslib:2018,  
  title = {{DESlib}: {A} {Dynamic} ensemble selection library in {Python}},  
  journal = {arXiv preprint arXiv:1802.04967},  
  author = {Cruz, Rafael M. O. and Hafemann, Luiz G. and Sabourin, Robert and  
↪Cavalcanti, George D. C.},  
  year = {2018}  
}
```

5.1 References

d

- `deslib.dcs`, 13
- `deslib.dcs.a_posteriori`, 15
- `deslib.dcs.a_priori`, 18
- `deslib.dcs.base`, 13
- `deslib.dcs.lca`, 21
- `deslib.dcs.mcb`, 24
- `deslib.dcs.mla`, 27
- `deslib.dcs.ola`, 29
- `deslib.dcs.rank`, 32
- `deslib.des`, 34
- `deslib.des.base`, 35
- `deslib.des.des_clustering`, 39
- `deslib.des.des_knn`, 44
- `deslib.des.des_p`, 42
- `deslib.des.knop`, 46
- `deslib.des.knora_e`, 48
- `deslib.des.knora_u`, 51
- `deslib.des.meta_des`, 37
- `deslib.des.probabilistic`, 55
- `deslib.static`, 59
- `deslib.static.oracle`, 60
- `deslib.static.single_best`, 60
- `deslib.static.static_selection`, 61
- `deslib.util`, 62
- `deslib.util.aggregation`, 65
- `deslib.util.diversity`, 62
- `deslib.util.instance_hardness`, 71
- `deslib.util.prob_functions`, 69

A

agreement_measure() (in module deslib.util.diversity), 63
 APosteriori (class in deslib.dcs.a_posteriori), 15
 APriori (class in deslib.dcs.a_priori), 18
 average_combiner() (in module deslib.util.aggregation), 65
 average_rule() (in module deslib.util.aggregation), 65

C

ccprmod() (in module deslib.util.prob_functions), 69
 classify_with_ds() (deslib.dcs.base.DCS method), 14
 classify_with_ds() (deslib.des.base.DES method), 35
 compute_pairwise_diversity() (in module deslib.util.diversity), 63
 correlation_coefficient() (in module deslib.util.diversity), 63

D

DCS (class in deslib.dcs.base), 13
 DES (class in deslib.des.base), 35
 DESCustering (class in deslib.des.des_clustering), 39
 DESKL (class in deslib.des.probablistic), 56
 DESKNN (class in deslib.des.des_knn), 44
 deslib.dcs (module), 13
 deslib.dcs.a_posteriori (module), 15
 deslib.dcs.a_priori (module), 18
 deslib.dcs.base (module), 13
 deslib.dcs.lca (module), 21
 deslib.dcs.mcb (module), 24
 deslib.dcs.mla (module), 27
 deslib.dcs.ola (module), 29
 deslib.dcs.rank (module), 32
 deslib.des (module), 34
 deslib.des.base (module), 35
 deslib.des.des_clustering (module), 39
 deslib.des.des_knn (module), 44
 deslib.des.des_p (module), 42
 deslib.des.knop (module), 46
 deslib.des.knora_e (module), 48

deslib.des.knora_u (module), 51
 deslib.des.meta_des (module), 37
 deslib.des.probablistic (module), 53, 55–59
 deslib.static (module), 59
 deslib.static.oracle (module), 60
 deslib.static.single_best (module), 60
 deslib.static.static_selection (module), 61
 deslib.util (module), 62
 deslib.util.aggregation (module), 65
 deslib.util.diversity (module), 62
 deslib.util.instance_hardness (module), 71
 deslib.util.prob_functions (module), 69
 DESP (class in deslib.des.des_p), 42
 disagreement_measure() (in module deslib.util.diversity), 63
 double_fault() (in module deslib.util.diversity), 64

E

entropy_func() (in module deslib.util.prob_functions), 69
 estimate_competence() (deslib.dcs.a_posteriori.APosteriori method), 16
 estimate_competence() (deslib.dcs.a_priori.APriori method), 19
 estimate_competence() (deslib.dcs.base.DCS method), 14
 estimate_competence() (deslib.dcs.lca.LCA method), 22
 estimate_competence() (deslib.dcs.mcb.MCB method), 25
 estimate_competence() (deslib.dcs.mla.MLA method), 27
 estimate_competence() (deslib.dcs.ola.OLA method), 30
 estimate_competence() (deslib.dcs.rank.Rank method), 33
 estimate_competence() (deslib.des.base.DES method), 36
 estimate_competence() (deslib.des.des_clustering.DESClustering method), 40
 estimate_competence() (deslib.des.des_knn.DESKNN method), 44
 estimate_competence() (deslib.des.des_p.DESP method), 42
 estimate_competence() (deslib.des.knora_e.KNORAE method), 49

`estimate_competence()` (deslib.des.knora_u.KNORAU method), 51
`estimate_competence()` (deslib.des.probabilistic.Probabilistic method), 53
`estimate_competence_from_proba()` (deslib.des.base.DES method), 36
`estimate_competence_from_proba()` (deslib.des.knop.KNOP method), 47
`estimate_competence_from_proba()` (deslib.des.meta_des.METADES method), 38
`Exponential` (class in deslib.des.probabilistic), 58
`exponential_func()` (in module deslib.util.prob_functions), 70

F

`fit()` (deslib.dcs.a_posteriori.APosteriori method), 17
`fit()` (deslib.dcs.a_priori.APriori method), 20
`fit()` (deslib.dcs.lca.LCA method), 22
`fit()` (deslib.dcs.mcb.MCB method), 25
`fit()` (deslib.dcs.mla.MLA method), 28
`fit()` (deslib.dcs.ola.OLA method), 30
`fit()` (deslib.dcs.rank.Rank method), 33
`fit()` (deslib.des.des_clustering.DESClustering method), 40
`fit()` (deslib.des.des_knn.DESKNN method), 45
`fit()` (deslib.des.des_p.DESP method), 43
`fit()` (deslib.des.knop.KNOP method), 47
`fit()` (deslib.des.knora_e.KNORAE method), 49
`fit()` (deslib.des.knora_u.KNORAU method), 51
`fit()` (deslib.des.meta_des.METADES method), 38
`fit()` (deslib.des.probabilistic.Probabilistic method), 54
`fit()` (deslib.static.single_best.SingleBest method), 61
`fit()` (deslib.static.static_selection.StaticSelection method), 62

H

`hardness_region_competence()` (in module deslib.util.instance_hardness), 71

K

`KNOP` (class in deslib.des.knop), 46
`KNORAE` (class in deslib.des.knora_e), 48
`KNORAU` (class in deslib.des.knora_u), 51

L

`LCA` (class in deslib.dcs.lca), 21
`log_func()` (in module deslib.util.prob_functions), 70
`Logarithmic` (class in deslib.des.probabilistic), 59

M

`majority_voting()` (in module deslib.util.aggregation), 65
`majority_voting_rule()` (in module deslib.util.aggregation), 66

`maximum_combiner()` (in module deslib.util.aggregation), 66
`maximum_rule()` (in module deslib.util.aggregation), 66
`MCB` (class in deslib.dcs.mcb), 24
`median_combiner()` (in module deslib.util.aggregation), 66
`median_rule()` (in module deslib.util.aggregation), 67
`METADES` (class in deslib.des.meta_des), 37
`min_difference()` (in module deslib.util.prob_functions), 70
`minimum_combiner()` (in module deslib.util.aggregation), 67
`minimum_rule()` (in module deslib.util.aggregation), 67
`MinimumDifference` (class in deslib.des.probabilistic), 57
`MLA` (class in deslib.dcs.mla), 27

N

`negative_double_fault()` (in module deslib.util.diversity), 64

O

`OLA` (class in deslib.dcs.ola), 29
`Oracle` (class in deslib.static.oracle), 60

P

`potential_func()` (deslib.des.probabilistic.Probabilistic static method), 54
`predict()` (deslib.dcs.a_posteriori.APosteriori method), 17
`predict()` (deslib.dcs.a_priori.APriori method), 20
`predict()` (deslib.dcs.lca.LCA method), 22
`predict()` (deslib.dcs.mcb.MCB method), 25
`predict()` (deslib.dcs.mla.MLA method), 28
`predict()` (deslib.dcs.ola.OLA method), 31
`predict()` (deslib.dcs.rank.Rank method), 33
`predict()` (deslib.des.des_clustering.DESClustering method), 41
`predict()` (deslib.des.des_knn.DESKNN method), 45
`predict()` (deslib.des.des_p.DESP method), 43
`predict()` (deslib.des.knop.KNOP method), 47
`predict()` (deslib.des.knora_e.KNORAE method), 49
`predict()` (deslib.des.knora_u.KNORAU method), 52
`predict()` (deslib.des.meta_des.METADES method), 38
`predict()` (deslib.static.oracle.Oracle method), 60
`predict()` (deslib.static.single_best.SingleBest method), 61
`predict()` (deslib.static.static_selection.StaticSelection method), 62
`predict_proba()` (deslib.dcs.a_posteriori.APosteriori method), 17
`predict_proba()` (deslib.dcs.a_priori.APriori method), 20
`predict_proba()` (deslib.dcs.lca.LCA method), 23
`predict_proba()` (deslib.dcs.mcb.MCB method), 25
`predict_proba()` (deslib.dcs.mla.MLA method), 28
`predict_proba()` (deslib.dcs.ola.OLA method), 31
`predict_proba()` (deslib.dcs.rank.Rank method), 33

- predict_proba() (deslib.des.des_clustering.DESClustering method), 41
 predict_proba() (deslib.des.des_knn.DESKNN method), 45
 predict_proba() (deslib.des.des_p.DESP method), 43
 predict_proba() (deslib.des.knop.KNOP method), 48
 predict_proba() (deslib.des.knora_e.KNORAE method), 50
 predict_proba() (deslib.des.knora_u.KNORAU method), 52
 predict_proba() (deslib.des.meta_des.METADES method), 39
 predict_proba() (deslib.static.single_best.SingleBest method), 61
 predict_proba_ensemble() (in module deslib.util.aggregation), 67
 predict_proba_ensemble_weighted() (in module deslib.util.aggregation), 67
 predict_proba_with_ds() (deslib.dcs.base.DCS method), 14
 predict_proba_with_ds() (deslib.des.base.DES method), 36
 Probabilistic (class in deslib.des.probablistic), 53
 product_combiner() (in module deslib.util.aggregation), 68
 product_rule() (in module deslib.util.aggregation), 68
- ## Q
- Q_statistic() (in module deslib.util.diversity), 62
- ## R
- Rank (class in deslib.dcs.rank), 32
 ratio_errors() (in module deslib.util.diversity), 65
 RRC (class in deslib.des.probablistic), 55
- ## S
- score() (deslib.dcs.a_posteriori.APosteriori method), 17
 score() (deslib.dcs.a_priori.APriori method), 20
 score() (deslib.dcs.lca.LCA method), 23
 score() (deslib.dcs.mcb.MCB method), 26
 score() (deslib.dcs.mla.MLA method), 28
 score() (deslib.dcs.ola.OLA method), 31
 score() (deslib.dcs.rank.Rank method), 34
 score() (deslib.des.des_clustering.DESClustering method), 41
 score() (deslib.des.des_knn.DESKNN method), 45
 score() (deslib.des.des_p.DESP method), 43
 score() (deslib.des.knop.KNOP method), 48
 score() (deslib.des.knora_e.KNORAE method), 50
 score() (deslib.des.knora_u.KNORAU method), 52
 score() (deslib.des.meta_des.METADES method), 39
 score() (deslib.static.oracle.Oracle method), 60
 select() (deslib.dcs.a_posteriori.APosteriori method), 18
 select() (deslib.dcs.a_priori.APriori method), 20
 select() (deslib.dcs.base.DCS method), 15
 select() (deslib.dcs.lca.LCA method), 23
 select() (deslib.dcs.mcb.MCB method), 26
 select() (deslib.dcs.mla.MLA method), 29
 select() (deslib.dcs.ola.OLA method), 31
 select() (deslib.dcs.rank.Rank method), 34
 select() (deslib.des.base.DES method), 37
 select() (deslib.des.des_clustering.DESClustering method), 41
 select() (deslib.des.des_knn.DESKNN method), 46
 select() (deslib.des.des_p.DESP method), 43
 select() (deslib.des.knop.KNOP method), 48
 select() (deslib.des.knora_e.KNORAE method), 50
 select() (deslib.des.knora_u.KNORAU method), 52
 select() (deslib.des.meta_des.METADES method), 39
 select() (deslib.des.probablistic.Probablistic method), 54
 SingleBest (class in deslib.static.single_best), 60
 softmax() (in module deslib.util.prob_functions), 71
 source_competence() (deslib.des.probablistic.DESKL method), 56
 source_competence() (deslib.des.probablistic.Exponential method), 58
 source_competence() (deslib.des.probablistic.Logarithmic method), 59
 source_competence() (deslib.des.probablistic.MinimumDifference method), 57
 source_competence() (deslib.des.probablistic.Probablistic method), 54
 source_competence() (deslib.des.probablistic.RRC method), 55
 StaticSelection (class in deslib.static.static_selection), 61
- ## W
- weighted_majority_voting() (in module deslib.util.aggregation), 68
 weighted_majority_voting_rule() (in module deslib.util.aggregation), 68