deslib Documentation

Release 0.1

Rafael Cruz

Apr 24, 2018

Contents

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.

CHAPTER 1

Introduction:

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.

CHAPTER 2

Installation:

The package 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 automaticatically installed using the pip commands above.

CHAPTER 3

API Reference:

3.1 Dynamic Ensemble Selection

The *deslib.des* provides a set of key dynamic ensemble selection algorithms (DES). DES techniques by default selects all base classifiers that attain a certain competence level.

3.1.1 DES class

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

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.

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_instance(*query*)

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_features]] The test sample

Returns

predicted_label: The predicted label of the query

estimate_competence(*query*)

Estimate the competence of each base classifier ci 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 containing the test sample = [n_features]]

Returns

competences [array of shape = [n_classifiers]] The competence level estimated for each base classifier

predict_proba_instance(*query*)

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_features]] The test sample

Returns

——-

predicted_proba [array = [n_classes] with the probability estimates for all classes]

select(*competences*)

Select the most competent classifier for the classification of the query sample x. The most competent classifier (dcs) or an ensemble with the most competent classifiers (des) is returned

Parameters

competences [array of shape = [n_classifiers]] The estimated competence level for the base classifiers

Returns

indices [List of index of the selected base classifier(s)]

3.1.2 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*, *gamma=0.5*, *mode='selection'*, *DFP=False*, *with_IH=False*, *safe_k=None*, *IH_rate=0.3*)

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

This method works selects all classifiers that correctly classified at least one sample belonging to the region of competence of the test sample x. 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.

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.

gamma [float(Default = 0.5)] Threshold used to select the base classifier. Only the base classifiers with competence level higher than the gamma 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(*query*)

Estimate the competence of each base classifier ci 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_features]] The test sample

Returns

competences [array of shape = [n_classifiers]] The 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] with the data.]

y [class labels of each sample 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] with the]

probabilities estimates for each class in the classifier model.

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 Gamma.

Parameters

competences [array of shape = [n_classifiers]] The competence level estimated for each base classifier

Returns

indices [the indices of the selected base classifiers]

3.1.3 DES Clustering

class deslib.des.des_clustering.**DESClustering**(*pool_classifiers*, *k=5*, *mode='selection'*, *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 more_diverse of the base classifiers. The K-means algorithm is used to define the region of competence First the most accurate classifiers are selected. Next, the most diverse classifiers, in relation to the selected classifiers, are added to the ensemble

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 = 5)] Number of neighbors used to estimate the competence of the base classifiers.

mode [String (Default = "selection")] whether the technique will perform dynamic selection, dynamic weighting or an hybrid approach for classification

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*)

get the competence estimates of each base classifier ci for the classification of the query sample x.

In this case, the competences are pre-calculated based on each cluster. So this method computes the nearest cluster of the query sample and get the pre-calculated competences of the base classifiers for the nearest cluster.

Parameters

query [array of shape = [n_features]] The query sample

Returns

competences [array = [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. In this case, after fitting the roc_algorithm method, the ensemble containing most competent classifiers taking into account accuracy and diversity are estimated for each cluster.

Parameters

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

y [class labels of each sample 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] with the]

probabilities estimates for each class in the classifier model.

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.

Since the method is based on roc_algorithm, the ensemble for each cluster is already pre-calculated. So, we only need to estimate which is the nearest cluster and then get the classifiers that were pre-selected for this cluster

Parameters

query [array of shape = [n_features]] The query sample

Returns

indices [List containing the indices of the selected base classifiers]

3.1.4 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/M, where M is the number of classes in the problem.

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.

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*)

estimate the competence of each base classifier in the pool. The competence level is estimated based on the classification accuracy of the base classifier for the region of competence.

Parameters

query [array of shape = [n_features]] The test sample

Returns

competences [array of shape = [n_classifiers]] The 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 [matrix of shape = [n_samples, n_features] with the data.]

y [class labels of each sample 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] with the]

probabilities estimates for each class in the classifier model.

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_classifiers] containing the competence level estimated] for each base classifier.

Returns

indices [List with the indices of the selected base classifiers.]

3.1.5 DES-KNN

class deslib.des.des_knn.**DESKNN**(*pool_classifiers*, *k=7*, *DFP=False*, *with_IH=False*, *safe_k=None*, *IH_rate=0.3*, *mode='selection'*, *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 more_diverse of the base classifiers. First the most accurate classifiers are selected. Next, the most diverse classifiers, in relation to the selected classifiers, are added to the ensemble

Parameters

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

classification problem.

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.

mode [String (Default = "selection")] whether the technique will perform dynamic selection, dynamic weighting or an hybrid approach 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*)

get the competence estimates of each base classifier ci for the classification of the query sample x.

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 cf shape = [n_features]] The query sample

Returns

——-

competences [array of shape = [n_classifiers]] The competence level estimated for each base classifier

diversity [array of shape = [n_classifiers]] The diversity 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 [matrix of shape = [n_samples, n_features] with the data.]

y [class labels of each sample 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] with the]

probabilities estimates for each class in the classifier model.

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 containing the N most accurate ant the J most diverse classifiers for the classification of the query

Parameters

query [array of shape = [n_features]] The test sample

Returns

indices [the indices of the selected base classifiers]

3.1.6 KNOP

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

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(*query*)

In this method, the competence of the base classifiers is simply computed as the number of samples in the region of competence that it correctly classified. However, the region of competence here is estimated in the decision space using output profiles.

Returns an array containing the level of competence estimated. The size of the array is equals to the size of the generated_pool of classifiers.

Parameters

query [array of shape = [n_features]] The test sample to be classified

Returns

competences [array of shape = [n_classifiers]] The competence level estimated for each base classifier

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]] containing the input data.

y [array of shape = [n_samples]] Class labels of each sample 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] with the]

probabilities estimates for each class in the classifier model.

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 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

query [array of shape = [n_features]] The test sample to be classified

Returns

votes [array containing the votes of the ensemble for each class]

3.1.7 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 is selected. In the case that no classifiers achieve a perfect accuracy, the size of the region of competence is reduced (by one 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.

Parameters

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*)

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 (achieves 100%). The competence level estimate is then the maximum size of the region of competence that the corresponding base classifier is a local Oracle.

Parameters

query [array of shape = [n_features]] The test sample

Returns

competences [array of shape = [n_classifiers]] The competence level estimated for each base classifier in the pool

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 [matrix of shape = [n_samples, n_features] with the data.]

y [class labels of each sample 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] with the]

probabilities estimates for each class in the classifier model.

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_classifiers]] The competence level estimated for each base classifier

Returns

indices [List with the indices of the selected base classifiers]

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.

3.1.8 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 works selects all classifiers that correctly classified at least one sample belonging to the region of competence of the test sample x. 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.

Parameters

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.

aknn [Boolean (Default = False)] Determines the type of KNN algorithm that is used. set to true for the A-KNN method.

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*)

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

Parameters

query [array of shape = [n_features] containing the test sample]

Returns

competences [array of shape = [n_classifiers] containing the 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 [matrix of shape = [n_samples, n_features] with the data.]

y [class labels of each sample 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] with the]

probabilities estimates for each class in the classifier model.

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 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

query [array of shape = [n_features] containing the test sample]

Returns

votes [the number of votes for each class]

3.1.9 Probabilistic

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

estimate the competence of each base classifier ci 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.

Parameters

query [array containing the test sample = [n_features]]

Returns

competences [array of shape = [n_classifiers]] The competence level estimated for each base classifier

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 [matrix of shape = [n_samples, n_features] with the data.]

y [class labels of each sample in X.]

Returns

self

static potential_func(*dist*)
Gaussian potential function to decrease the influence of the source of competence as the distance between xk and the query increases

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_classifiers]] The estimated competence level for the base classifiers

Returns

indices [the indices of the selected base classifiers]

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.

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.

source_competence()

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

The source of competence C_src at the validation point xk 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-KL

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 = 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 xk calculated using 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 xk

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 (xk) 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 xk 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 = 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.

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 xk 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.probabilistic.**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 xk 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 xk [1].

The influence of each sample xk 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 = 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.

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 xk 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 xk.

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 xk is calculated by logarithm 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 Dynamic Classifier Selection

The *deslib.dcs* provides a set of key dynamic classifier selection algorithms (DCS).

3.2.1 DCS 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_instance(*query*)

Predicts the class label of the corresponding query sample.

If self.mode == "all", the majority voting scheme is used to aggregate the predictions of all classifiers with the max competence level estimate.

Parameters

query [array containing the test sample = [n_features]]

Returns

The predicted label of the query

estimate_competence(*query*)

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

Parameters

query [array containing the test sample = [n_features]]

Returns

competences [array of shape = [n_classifiers]] The competence level estimated for each base classifier in the pool

predict_proba_instance(*query*)

Predicts the posterior probabilities of the corresponding query sample.

If self.mode == "all", get the probability estimates of the selected ensemble. Otherwise, the technique gets the probability estimates from the selected base classifier

Parameters

query [array containing the test sample = [n_features]]

Returns

predicted_proba [array = [n_classes] with the probability estimates for all classes]

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).

Parameters

competences [array = [n_classifiers] containing the estimated competence level for the base classifiers]

Returns

selected_clf [index of the selected base classifier(s)]

3.2.2 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.

This method works similarly to the LCA technique. The only difference is that it uses the scores obtained by the base classifiers as well as the distance between the test sample and each pattern in the region of competence are also considered 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 = 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*)

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

The A Posteriori method considers the probability of correct classification of the base classifier ci, taking into account the supports obtained by the base classifier ci for the samples belonging to the region of competence. The probability of correct classification for a base classifier ci is calculated taking into account only the samples in the region of competence from a specific class wl. In this case, wl is the predict class of the base classifier ci 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.

Returns an array containing the level of competence estimated using the LCA method for each base classifier. The size of the array is equals to the size of the pool of classifiers.

Parameters

query [array cf shape = [n_features]] The query sample

Returns

competences [array of shape = [n_classifiers]] The competence level estimated for each base classifier

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] with the]

probabilities estimates for each class in the classifier model.

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).

Parameters

competences [array = [n_classifiers] containing the estimated competence level for the base classifiers]

Returns

selected_clf [index of the selected base classifier(s)]

3.2.3 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.

This method works similarly to the OLA technique. The only difference is that it uses the scores obtained by the base classifiers as well as the distance between the test sample and each pattern in the region of competence are also considered 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 = 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*)

estimate the competence of each base classifier ci the classification of the query sample using the A Priori method.

The A Priori method considers the probability of correct classification of the base classifier ci, in the region of competence, taking into account the supports obtained by the base classifier ci. Hence, the vector containing the posterior probabilities for each class is considered instead of only the label assigned to each sample 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.

Returns an array containing the level of competence estimated using the LCA method for each base classifier. The size of the array is equals to the size of the pool of classifiers.

Parameters

query [array cf shape = [n_features]] The query sample

Returns

——-

competences [array of shape = [n_classifiers]] The competence level estimated for each base classifier

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] with the]

probabilities estimates for each class in the classifier model.

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).

Parameters

competences [array = [n_classifiers] containing the estimated competence level for the base classifiers]

Returns

selected_clf [index of the selected base classifier(s)]

3.2.4 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 Classifier 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 that assigns a test sample to class Ci. The competence is estimated by the percentage of the local training samples assigned to class Ci by this classifier that have been correctly labeled.

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.

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*)

estimate the competence of each base classifier ci 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 belonging to the class wl in this neighborhood.

Returns an array containing the level of competence estimated using the LCA method for each base classifier. The size of the array is equals to the size of the pool of classifiers.

Parameters

query [array cf shape = [n_features]] The query sample

Returns

——-

competences [array of shape = [n_classifiers]] The 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 [matrix of shape = [n_samples, n_features] with the data.]

y [class labels of each sample 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] with the]

probabilities estimates for each class in the classifier model.

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).

Parameters

competences [array = [n_classifiers] containing the estimated competence level for the base classifiers]

Returns

selected_clf [index of the selected base classifier(s)]

3.2.5 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 both the local accuracy of the base

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

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*)

estimate the competence of each base classifier ci 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 its region of competence are estimated. Instances with similarity lower than a predefined threshold are removed from the region of competence.

Then, the competence level of the base classifiers are estimated based on their classification accuracy in the final region of competence.

Returns an array containing the level of competence estimated using the MCB method for each base classifier. The size of the array is equals to the size of the generated_pool of classifiers.

Parameters

query [array cf shape = [n_features]] The query sample

Returns

——-

competences [array of shape = [n_classifiers]] The 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 [matrix of shape = [n_samples, n_features] with the data.]

y [class labels of each sample 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] with the]

probabilities estimates for each class in the classifier model.

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).

Parameters

competences [array = [n_classifiers] containing the estimated competence level for the base classifiers]

Returns

selected_clf [index of the selected base classifier(s)]

3.2.6 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.

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.

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*)

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

Two versions of the LCA are considered for the competence estimates:

The Modified local accuracy of the base classifiers is estimated by its classification accuracy taking into account only the samples belonging to the class wl in the region of competence. In this case, wl is the predict class of the base classifier ci 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.

Parameters

query [array cf shape = [n_features]] The query sample

Returns

competences [array of shape = [n_classifiers]] The 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 [matrix of shape = [n_samples, n_features] with the data.]

y [class labels of each sample 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] with the]

probabilities estimates for each class in the classifier model.

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).

Parameters

competences [array = [n_classifiers] containing the estimated competence level for the base classifiers]

Returns

selected_clf [index of the selected base classifier(s)]

3.2.7 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).

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.

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*)

estimate the competence of each base classifier ci the classification of the query sample using the Overall Local Accuracy criterion.

The competences for each base classifier ci is estimated by its classification accuracy considering the k-Nearest Neighbors.

Returns an array containing the level of competence estimated using the OLA method for each base classifier. The size of the array is equals to the size of the generated_pool of classifiers.

Parameters

query [array cf shape = [n_features]] The query sample

Returns

——-

competences [array of shape = [n_classifiers]] The 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 [matrix of shape = [n_samples, n_features] with the data.]

y [class labels of each sample 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] with the]

probabilities estimates for each class in the classifier model.

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).

Parameters

competences [array = [n_classifiers] containing the estimated competence level for the base classifiers]

Returns

selected_clf [index of the selected base classifier(s)]

3.2.8 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 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 = 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*)

estimate the rank of each base classifier ci considering the whole neighborhood. The rank of the base classifier is estimated by the number of consecutive correctly classified samples in the defined region of competence.

Returns an array containing the level of competence (rank) estimated for each base classifier. The size of the array is equals to the size of the pool of classifiers.

Parameters

query [array of shape = [n_features]] The test sample

Returns

competences [array of shape = [n_classifiers]] The 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 [matrix of shape = [n_samples, n_features] with the data.]

y [class labels of each sample 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] with the]

probabilities estimates for each class in the classifier model.

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).

Parameters

competences [array = [n_classifiers] containing the estimated competence level for the base classifiers]

Returns

selected_clf [index of the selected base classifier(s)]

3.3 Static Selection

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.

3.3.1 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 [Classification accuracy of the Oracle model.]

3.3.2 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]] The data to be classified

y [array of shape = [n_samples]] Class labels of each sample 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.

3.3.3 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]] The data to be classified

y [array of shape = [n_samples]] Class labels of each sample 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.

3.4 Util

The *deslib.util* This module includes various utilities. They are divided into three 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.

3.4.1 Diversity

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 in X.

y_pred1 [array of shape = [n_samples]:] predicted class labels by the classifier 1 for each sample in X.

y_pred2 [array of shape = [n_samples]:] predicted class labels by the classifier 2 for each sample in X.

Returns

Q [The q-statistic measure between two classifiers]

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 in X.

y_pred1 [array of shape = [n_samples]:] predicted class labels by the classifier 1 for each sample in X.

y_pred2 [array of shape = [n_samples]:] predicted class labels by the classifier 2 for each sample in X.

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 in X.

y_pred1 [array of shape = [n_samples]:] predicted class labels by the classifier 1 for each sample in X.

y_pred2 [array of shape = [n_samples]:] predicted class labels by the classifier 2 for each sample in X.

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 in X.

y_pred1 [array of shape = [n_samples]:] predicted class labels by the classifier 1 for each sample in X.

y_pred2 [array of shape = [n_samples]:] predicted class labels by the classifier 2 for each sample in X.

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.

3.4.2 Aggregation

deslib.util.aggregation.**average_rule**(*classifier_ensemble*, *X*)
Apply the average_rule 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

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.**get_ensemble_votes**(*classifier_ensemble*, *X*)
Calculates the votes obtained by each based classifier in the ensemble for 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

votes [array of shape = [n_samples, n_classifiers]] The votes obtained by each base classifier

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.**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

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_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

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.**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*)

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

Returns

predicted_label [array of shape = [n_samples]] The label of each query sample predicted using the majority voting rule

3.4.3 Probabilistic Functions

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.

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

CHAPTER 4

Examples:

Example using 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 documentation and im the Examples directory.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index