brew Documentation

Release 0.1.4

Dayvid Victor

Jul 27, 2018

Contents

pypi package 0.1.4 build failing

This project has not being maintained for a while, so as of now we have abandoned it. If you want an alternative ensemble library in python, we recommend [DESLib](https://github.com/Menelau/DESlib) instead.

This project was started in 2014 by *Dayvid Victor* and *Thyago Porpino*
for the Multiple Classifier Systems class at Federal University of Pernambuco.

The aim of this project is to provide an easy API for Ensembling, Stacking,
Blending, Ensemble Generation, Ensemble Pruning, Dynamic Classifier Selection,
and Dynamic Ensemble Selection.

CHAPTER 1

Features

- General: Ensembling, Stacking and Blending.
- Ensemble Classifier Generators: Bagging, Random Subspace, SMOTE-Bagging, ICS-Bagging, SMOTE-ICS-Bagging.
- Dynamic Selection: Overall Local Accuracy (OLA), Local Class Accuracy (LCA), Multiple Classifier Behavior (MCB), K-Nearest Oracles Eliminate (KNORA-E), K-Nearest Oracles Union (KNORA-U), A Priori Dynamic Selection, A Posteriori Dynamic Selection, Dynamic Selection KNN (DSKNN).
- Ensemble Combination Rules: majority vote, min, max, mean and median.
- Ensemble Diversity Metrics: Entropy Measure E, Kohavi Wolpert Variance, Q Statistics, Correlation Coefficient p, Disagreement Measure, Agreement Measure, Double Fault Measure.
- Ensemble Pruning: Ensemble Pruning via Individual Contribution (EPIC).

CHAPTER 2

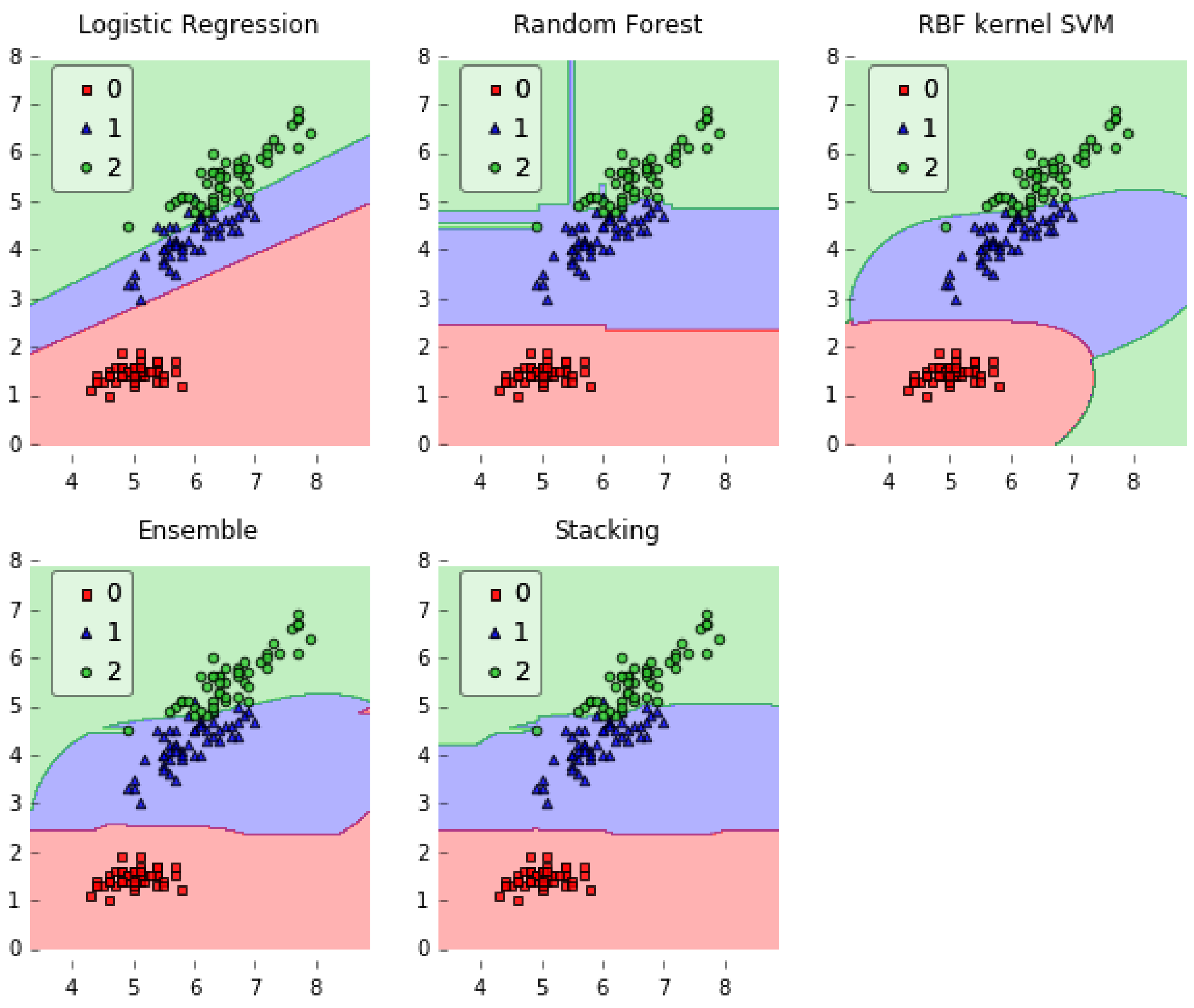
Example

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import itertools

import sklearn

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

from brew.base import Ensemble, EnsembleClassifier
from brew.stacking.stacker import EnsembleStack, EnsembleStackClassifier
from brew.combination.combiner import Combiner

from mlxtend.data import iris_data
from mlxtend.evaluate import plot_decision_regions

# Initializing Classifiers
clf1 = LogisticRegression(random_state=0)
clf2 = RandomForestClassifier(random_state=0)
clf3 = SVC(random_state=0, probability=True)

# Creating Ensemble
ensemble = Ensemble([clf1, clf2, clf3])
eclf = EnsembleClassifier(ensemble=ensemble, combiner=Combiner('mean'))

# Creating Stacking
layer_1 = Ensemble([clf1, clf2, clf3])
layer_2 = Ensemble([sklearn.clone(clf1)])

stack = EnsembleStack(cv=3)

stack.add_layer(layer_1)
```

(continues on next page)

(continued from previous page)

```
stack.add_layer(layer_2)

sclf = EnsembleStackClassifier(stack)

clf_list = [clf1, clf2, clf3, eclf, sclf]
lbl_list = ['Logistic Regression', 'Random Forest', 'RBF kernel SVM', 'Ensemble',
↪'Stacking']

# Loading some example data
X, y = iris_data()
X = X[:,[0, 2]]

# WARNING, WARNING, WARNING
# brew requires classes from 0 to N, no skipping allowed
d = {yi : i for i, yi in enumerate(set(y))}
y = np.array([d[yi] for yi in y])

# Plotting Decision Regions
gs = gridspec.GridSpec(2, 3)
fig = plt.figure(figsize=(10, 8))

itt = itertools.product([0, 1, 2], repeat=2)

for clf, lab, grd in zip(clf_list, lbl_list, itt):
    clf.fit(X, y)
    ax = plt.subplot(gs[grd[0], grd[1]])
    fig = plot_decision_regions(X=X, y=y, clf=clf, legend=2)
    plt.title(lab)
plt.show()
```


Logistic Regression
Random Forest
RBF kernel SVM
Ensemble
Stacking
0
1
2
0 1 2 3 4 5 6 7 8
4 5 6 7 8

CHAPTER 3

Dependencies

- Python 2.7+
- scikit-learn >= 0.15.2
- Numpy >= 1.6.1
- SciPy >= 0.9
- Matplotlib >= 0.99.1 (examples, only)
- mlxtend (examples, only)

CHAPTER 4

Installing

You can easily install brew using `pip`:

```
pip install brew
```

or, if you prefer an up-to-date version, get it from here:

```
pip install git+https://github.com/viisar/brew.git
```

CHAPTER 5

Important References

- Kuncheva, Ludmila I. Combining pattern classifiers: methods and algorithms. John Wiley & Sons, 2014.
- Zhou, Zhi-Hua. Ensemble methods: foundations and algorithms. CRC Press, 2012.

CHAPTER 6

Contents:

6.1 brew package

6.1.1 Subpackages

brew.combination package

Submodules

brew.combination.combiner module

class `brew.combination.combiner.`**`Combiner`**(*rule='majority_vote'*)

Bases: `object`

`combine`(*results*)

brew.combination.rules module

[1] Kittler, J.; Hatef, M.; Duin, R.P.W.; Matas, J., "On combining classifiers," Pattern Analysis and Machine Intelligence, IEEE Transactions on , vol.20, no.3, pp.226,239, Mar 1998

`brew.combination.rules.`**`majority_vote_rule`**(*votes*)

Implements the majority vote rule as defined by [1].

This rule can always be used, because even if the classifiers output posterior probabilities, you can for example, decide to vote for the class with the greatest probability. The important thing is to transform the classifiers probabilitities/decisions into a matrix of votes.

Parameters **`votes`** (*`Numpy 2d-array with rows representing each class, columns`*) – representing each classifier and elements representing votes (binary). Each column should sum up to one (i.e. a classifier can only vote for one class).

`brew.combination.rules.`**`max_rule`**(*probs*)

Implements the max rule as defined by [1].

This rule only makes sense if the classifiers output the posterior probabilities for each class.

Parameters **`probs`** (*`Numpy 2d-array with rows representing each class, columns`*) – representing each classifier and elements representing posterior probabilities. Each column should sum up to one as a sanity check that the probabilities are valid.

`brew.combination.rules.`**`mean_rule`**(*probs*)

Implements the first case of the median rule as defined by [1].

This rule only makes sense if the classifiers output the posterior probabilities for each class.

Parameters **`probs`** (*`Numpy 2d-array with rows representing each class, columns`*) – representing each classifier and elements representing posterior probabilities. Each column should sum up to one as a sanity check that the probabilities are valid.

`brew.combination.rules.`**`median_rule`**(*probs*)

Implements the second case of the median rule as defined by [1].

This rule only makes sense if the classifiers output the posterior probabilities for each class.

Parameters **`probs`** (*`Numpy 2d-array with rows representing each class, columns`*) – representing each classifier and elements representing posterior probabilities. Each column should sum up to one as a sanity check that the probabilities are valid.

`brew.combination.rules.`**`min_rule`**(*probs*)

Implements the min rule as defined by [1].

This rule only makes sense if the classifiers output the posterior probabilities for each class.

Parameters **`probs`** (*`Numpy 2d-array with rows representing each class, columns`*) – representing each classifier and elements representing posterior probabilities. Each column should sum up to one as a sanity check that the probabilities are valid.

Module contents

`class` `brew.combination.`**`Combiner`**(*rule='majority_vote'*)

Bases: `object`

`combine`(*results*)

brew.generation package

Submodules

brew.generation.bagging module

`class` `brew.generation.bagging.`**`Bagging`**(*base_classifier=None, n_classifiers=100, combination_rule='majority_vote'*)

Bases: *`brew.generation.base.PoolGenerator`*

`fit`(*X, y*)

`predict`(*X*)

class brew.generation.bagging.**BaggingSK**(*base_classifier=None*, *n_classifiers=100*, *combination_rule='majority_vote'*)

Bases: *brew.generation.base.PoolGenerator*

" This class should not be used, use brew.generation.bagging.Bagging instead.

fit(*X*, *y*)

predict(*X*)

brew.generation.base module

class brew.generation.base.**PoolGenerator**

Bases: object

fit(*X*, *y*)

predict(*X*)

brew.generation.random_subspace module

class brew.generation.random_subspace.**RandomSubspace**(*base_classifier=None*, *n_classifiers=100*, *combination_rule='majority_vote'*, *max_features=0.5*)

Bases: *brew.generation.base.PoolGenerator*

fit(*X*, *y*)

predict(*X*)

brew.generation.smote_bagging module

class brew.generation.smote_bagging.**SmoteBagging**(*base_classifier=None*, *n_classifiers=100*, *combination_rule='majority_vote'*, *k=5*)

Bases: *brew.generation.base.PoolGenerator*

fit(*X*, *y*)

predict(*X*)

smote_bootstrap_sample(*X*, *y*, *b*, *k*)

class brew.generation.smote_bagging.**SmoteBaggingNew**(*base_classifier=None*, *n_classifiers=100*, *combination_rule='majority_vote'*, *k=5*)

Bases: *brew.generation.smote_bagging.SmoteBagging*

fit(*X*, *y*)

smote_bootstrap_sample(*X*, *y*, *b*, *k*)

Module contents

class brew.generation.**Bagging**(*base_classifier=None*, *n_classifiers=100*, *combination_rule='majority_vote'*)

Bases: *brew.generation.base.PoolGenerator*

fit(*X*, *y*)

predict(*X*)

class brew.generation.**SmoteBagging**(*base_classifier=None*, *n_classifiers=100*, *combination_rule='majority_vote'*, *k=5*)

Bases: *brew.generation.base.PoolGenerator*

fit(*X*, *y*)

predict(*X*)

smote_bootstrap_sample(*X*, *y*, *b*, *k*)

class brew.generation.**RandomSubspace**(*base_classifier=None*, *n_classifiers=100*, *combination_rule='majority_vote'*, *max_features=0.5*)

Bases: *brew.generation.base.PoolGenerator*

fit(*X*, *y*)

predict(*X*)

brew.metrics package

Subpackages

brew.metrics.diversity package

Submodules

brew.metrics.diversity.base module

class brew.metrics.diversity.base.**Diversity**(*metric=''*)

Bases: object

Ensemble Diversity Calculator.

The class calculates the diversity of ensemble of classifiers.

`metric`

function, receives the oracle output and returns float – Function used to calculate the metric.

Parameters metric (*{'e', 'kw', 'q', 'p', 'disagreement', 'agreement', 'df'}, optional*) – Metric used to compute the ensemble diversity:

- 'e' (Entropy Measure e) will use kuncheva_entropy_measure()
- 'kw' (Kohavi Wolpert Variance) will use kuncheva_kw()
- 'q' (Q Statistics) will use kuncheva_q_statistics()
- 'p' (Correlation Coefficient p) will use kuncheva_correlation_coefficient_p() # noqa

- 'disagreement' (Disagreement Measure) will use `kuncheva_disagreement_measure()` # noqa
- 'agreement' (Agreement Measure) will use `kuncheva_agreement_measure()` # noqa
- 'df' (Double Fault Measure) will use `kuncheva_double_fault_measure()` # noqa

Examples

```
>>> from brew.metrics.diversity.base import Diversity
>>> from brew.generation.bagging import Bagging
>>>
>>> from sklearn.tree import DecisionTreeClassifier
>>> import numpy as np
>>>
>>> X = np.array([[-1, 0], [-0.8, 1], [-0.8, -1], [-0.5, 0],
                  [0.5, 0], [1, 0], [0.8, 1], [0.8, -1]])
>>> y = np.array([1, 1, 1, 2, 1, 2, 2, 2])
>>> tree = DecisionTreeClassifier(max_depth=1, min_samples_leaf=1)
>>> bag = Bagging(base_classifier=tree, n_classifiers=10)
>>> bag.fit(X, y)
>>>
>>> div = Diversity(metric='q')
>>> q = div.calculate(bag.ensemble, Xtst, ytst)
>>> q < 1.01 and q > -1.01
True
```

See also:

brew.metrics.diversity.paired Paired diversity metrics.

brew.metrics.diversity.non_paired Non-paired diversity metrics.

References

Brown, Gavin, et al. "Diversity creation methods: a survey and categorisation." Information Fusion 6.1 (2005): 5-20.

Kuncheva, Ludmila I., and Christopher J. Whitaker. "Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy." Machine learning 51.2 (2003): 181-207.

Tang, E. Ke, Ponnuthurai N. Suganthan, and Xin Yao. "An analysis of diversity measures." Machine Learning 65.1 (2006): 247-271.

calculate(*ensemble*, *X*, *y*)

brew.metrics.diversity.non_paired module

brew.metrics.diversity.non_paired.**entropy_measure_e**(*ensemble*, *X*, *y*)

brew.metrics.diversity.non_paired.**kohavi_wolpert_variance**(*ensemble*, *X*, *y*)

brew.metrics.diversity.non_paired.**kuncheva_entropy_measure**(*oracle*)

brew.metrics.diversity.non_paired.**kuncheva_kw**(*oracle*) `brew.metrics.diversity.non_paired.`**`new_entropy`**`(`*ensemble, X, y*`)`

brew.metrics.diversity.paired module

`brew.metrics.diversity.paired.`**`agreement_measure`**`(`*y_true, y_pred_a, y_pred_b*`)`

`brew.metrics.diversity.paired.`**`correlation_coefficient_p`**`(`*y_true, y_pred_a, y_pred_b*`)`

`brew.metrics.diversity.paired.`**`disagreement_measure`**`(`*y_true, y_pred_a, y_pred_b*`)`

`brew.metrics.diversity.paired.`**`double_fault_measure`**`(`*y_true, y_pred_a, y_pred_b*`)`

`brew.metrics.diversity.paired.`**`kuncheva_agreement_measure`**`(`*oracle*`)`

`brew.metrics.diversity.paired.`**`kuncheva_correlation_coefficient_p`**`(`*oracle*`)`

`brew.metrics.diversity.paired.`**`kuncheva_disagreement_measure`**`(`*oracle*`)`

`brew.metrics.diversity.paired.`**`kuncheva_double_fault_measure`**`(`*oracle*`)`

`brew.metrics.diversity.paired.`**`kuncheva_q_statistics`**`(`*oracle*`)`

`brew.metrics.diversity.paired.`**`paired_metric_ensemble`**`(`*ensemble, X, y, paired_metric=<function q_statistics>*`)`

`brew.metrics.diversity.paired.`**`q_statistics`**`(`*y_true, y_pred_a, y_pred_b*`)`

Module contents

`class` `brew.metrics.diversity.`**`Diversity`**`(`*metric=''*`)`

Bases: `object`

Ensemble Diversity Calculator.

The class calculates the diversity of ensemble of classifiers.

`` `metric` ``

function, receives the oracle output and returns float – Function used to calculate the metric.

Parameters metric (*{'e', 'kw', 'q', 'p', 'disagreement', 'agreement', 'df'}, optional*) – Metric used to compute the ensemble diversity:

- ’e’ (Entropy Measure e) will use `kuncheva_entropy_measure()`
- ’kw’ (Kohavi Wolpert Variance) will use `kuncheva_kw()`
- ’q’ (Q Statistics) will use `kuncheva_q_statistics()`
- ’p’ (Correlation Coefficient p) will use `kuncheva_correlation_coefficient_p()` # noqa
- ’disagreement’ (Disagreement Measure) will use `kuncheva_disagreement_measure()` # noqa
- ’agreement’ (Agreement Measure) will use `kuncheva_agreement_measure()` # noqa
- ’df’ (Double Fault Measure) will use `kuncheva_double_fault_measure()` # noqa

Examples

```
>>> from brew.metrics.diversity.base import Diversity
>>> from brew.generation.bagging import Bagging
>>>
>>> from sklearn.tree import DecisionTreeClassifier
>>> import numpy as np
>>>
>>> X = np.array([[-1, 0], [-0.8, 1], [-0.8, -1], [-0.5, 0],
                  [0.5, 0], [1, 0], [0.8, 1], [0.8, -1]])
>>> y = np.array([1, 1, 1, 2, 1, 2, 2, 2])
>>> tree = DecisionTreeClassifier(max_depth=1, min_samples_leaf=1)
>>> bag = Bagging(base_classifier=tree, n_classifiers=10)
>>> bag.fit(X, y)
>>>
>>> div = Diversity(metric='q')
>>> q = div.calculate(bag.ensemble, Xtst, ytst)
>>> q < 1.01 and q > -1.01
True
```

See also:

brew.metrics.diversity.paired Paired diversity metrics.

brew.metrics.diversity.non_paired Non-paired diversity metrics.

References

Brown, Gavin, et al. “Diversity creation methods: a survey and categorisation.” Information Fusion 6.1 (2005): 5-20.

Kuncheva, Ludmila I., and Christopher J. Whitaker. “Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy.” Machine learning 51.2 (2003): 181-207.

Tang, E. Ke, Ponnuthurai N. Suganthan, and Xin Yao. “An analysis of diversity measures.” Machine Learning 65.1 (2006): 247-271.

calculate(*ensemble*, *X*, *y*)

Submodules

brew.metrics.evaluation module

class brew.metrics.evaluation.**Evaluator**(*metric='auc'*)

Bases: object

calculate(*y_true*, *y_pred*)

brew.metrics.evaluation.**acc_score**(*y_true*, *y_pred*, *positive_label=1*)

brew.metrics.evaluation.**auc_score**(*y_true*, *y_pred*, *positive_label=1*)

Module contents

brew.preprocessing package

Submodules

brew.preprocessing.smote module

`brew.preprocessing.smote.`**`smote`**(*T*, *N=100*, *k=1*)

T: minority class data N: percentage of oversampling k: number of neighbors used

Module contents

brew.selection package

Subpackages

brew.selection.dynamic package

Submodules

brew.selection.dynamic.base module

`class` `brew.selection.dynamic.base.`**`DCS`**(*Xval*, *yval*, *K=5*, *weighted=False*, *knn=None*)

Bases: `object`

`get_neighbors`(*x*, *return_distance=False*)

`select`(*ensemble*, *x*)

brew.selection.dynamic.knora module

`class` `brew.selection.dynamic.knora.`**`KNORA`**(*Xval*, *yval*, *K=5*, *weighted=False*, *knn=None*)

Bases: *`brew.selection.dynamic.base.DCS`*

`class` `brew.selection.dynamic.knora.`**`KNORA_ELIMINATE`**(*Xval*, *yval*, *K=5*, *weighted=False*, *knn=None*, *v2007=False*)

Bases: *`brew.selection.dynamic.knora.KNORA`*

K-nearest-oracles Eliminate.

The KNORA Eliminate reduces the neighborhood until finds an ensemble of classifiers that correctly classify all neighbors.

`` `Xval` ``

array-like, shape = [indeterminated, n_features] – Validation set.

`` `yval` ``

array-like, shape = [indeterminated] – Labels of the validation set.

`` `knn` ``

sklearn KNeighborsClassifier, – Classifier used to find neighborhood.

`weighted`
bool, (makes no difference in knora_eliminate) – Bool that defines if the classifiers uses weights or not

Examples

```
>>> from brew.selection.dynamic.knora import KNORA_ELIMINATE
>>> from brew.generation.bagging import Bagging
>>> from brew.base import EnsembleClassifier
>>>
>>> from sklearn.tree import DecisionTreeClassifier
>>> import numpy as np
>>>
>>> X = np.array([[-1, 0], [-0.8, 1], [-0.8, -1], [-0.5, 0],
                  [0.5, 0], [1, 0], [0.8, 1], [0.8, -1]])
>>> y = np.array([1, 1, 1, 2, 1, 2, 2, 2])
>>>
>>> dt = DecisionTreeClassifier(max_depth=1, min_samples_leaf=1)
>>> bag = Bagging(base_classifier=dt, n_classifiers=10)
>>> bag.fit(X, y)
>>>
>>> ke = KNORA_ELIMINATE(X, y, K=5)
>>>
>>> clf = EnsembleClassifier(bag.ensemble, selector=ke)
>>> clf.predict([-1.1,-0.5])
[1]
```

See also:

`brew.selection.dynamic.knora.KNORA_UNION` KNORA Union.

`brew.selection.dynamic.lca.LCA` Local Class Accuracy.

`brew.selection.dynamic.ola.OLA` Overall Local Accuracy.

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.

Hung-Ren Ko, A., Robert Sabourin, and A. de Souza Britto. "K-nearest oracle for dynamic ensemble selection." Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on. Vol. 1. IEEE, 2007

`select`(*ensemble*, *x*)

class `brew.selection.dynamic.knora.`**`KNORA_UNION`**(*Xval*, *yval*, *K=5*, *weighted=False*, *knn=None*)

Bases: *`brew.selection.dynamic.knora.KNORA`*

K-nearest-oracles Union.

The KNORA union reduces the neighborhood until finds an ensemble of classifiers that correctly classify all neighbors.

`Xval`
: *array-like, shape = [indeterminated, n_features]* – Validation set.

`yval`
: *array-like, shape = [indeterminated]* – Labels of the validation set.

`knn`
: *sklearn KNeighborsClassifier,* – Classifier used to find neighborhood.

`weighted`
: *bool, (makes no difference in knora_eliminate)* – Bool that defines if the classifiers uses weights or not

Examples

```
>>> from brew.selection.dynamic.knora import KNORA_UNION
>>> from brew.generation.bagging import Bagging
>>> from brew.base import EnsembleClassifier
>>>
>>> from sklearn.tree import DecisionTreeClassifier
>>> import numpy as np
>>>
>>> X = np.array([[-1, 0], [-0.8, 1], [-0.8, -1], [-0.5, 0],
                  [0.5, 0], [1, 0], [0.8, 1], [0.8, -1]])
>>> y = np.array([1, 1, 1, 2, 1, 2, 2, 2])
>>>
>>> dt = DecisionTreeClassifier(max_depth=1, min_samples_leaf=1)
>>> bag = Bagging(base_classifier=dt, n_classifiers=10)
>>> bag.fit(X, y)
>>>
>>> ku = KNORA_UNION(X, y, K=5)
>>>
>>> clf = EnsembleClassifier(bag.ensemble, selector=ku)
>>> clf.predict([-1.1,-0.5])
[1]
```

See also:

`brew.selection.dynamic.knora.KNORA_ELIMINATE` Knora Eliminate.

`brew.selection.dynamic.lca.LCA` Local Class Accuracy.

`brew.selection.dynamic.ola.OLA` Overall Local Accuracy.

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.

Hung-Ren Ko, A., Robert Sabourin, and A. de Souza Britto. "K-nearest oracle for dynamic ensemble selection." Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on. Vol. 1. IEEE, 2007.

`select` (*ensemble*, *x*)

brew.selection.dynamic.lca module

class brew.selection.dynamic.lca.**LCA**(*Xval*, *yval*, *K=5*, *weighted=False*, *knn=None*)

Bases: *brew.selection.dynamic.base.DCS*

select(*ensemble*, *x*)

class brew.selection.dynamic.lca.**LCA2**(*Xval*, *yval*, *K=5*, *weighted=False*, *knn=None*)

Bases: *brew.selection.dynamic.base.DCS*

Local Class Accuracy.

The Local Class Accuracy selects the best classifier for a sample using it's K nearest neighbors.

`Xval`
array-like, shape = [indeterminated, n_features] – Validation set.

`yval`
array-like, shape = [indeterminated] – Labels of the validation set.

`knn`
sklearn KNeighborsClassifier, – Classifier used to find neighborhood.

Examples

```
>>> from brew.selection.dynamic.lca import LCA
>>> from brew.generation.bagging import Bagging
>>> from brew.base import EnsembleClassifier
>>>
>>> from sklearn.tree import DecisionTreeClassifier
>>> import numpy as np
>>>
>>> X = np.array([[-1, 0], [-0.8, 1], [-0.8, -1], [-0.5, 0],
                  [0.5, 0], [1, 0], [0.8, 1], [0.8, -1]])
>>> y = np.array([1, 1, 1, 2, 1, 2, 2, 2])
>>> tree = DecisionTreeClassifier(max_depth=1, min_samples_leaf=1)
>>> bag = Bagging(base_classifier=tree, n_classifiers=10)
>>> bag.fit(X, y)
>>>
>>> lca = LCA(X, y, K=3)
>>>
>>> clf = EnsembleClassifier(bag.ensemble, selector=lca)
>>> clf.predict([-1.1,-0.5])
[1]
```

See also:

brew.selection.dynamic.ola.OLA Overall Local Accuracy.

References

Woods, Kevin, Kevin Bowyer, and W. Philip Kegelmeyer Jr. "Combination of multiple classifiers using local accuracy estimates." Computer Vision and Pattern Recognition, 1996. Proceedings CVPR'96, 1996 IEEE Computer Society Conference on. IEEE, 1996.

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.

select (*ensemble*, *x*)

brew.selection.dynamic.ola module

class brew.selection.dynamic.ola.**OLA** (*Xval*, *yval*, *K=5*, *weighted=False*, *knn=None*)
Bases: *brew.selection.dynamic.base.DCS*

select (*ensemble*, *x*)

class brew.selection.dynamic.ola.**OLA2** (*Xval*, *yval*, *K=5*, *weighted=False*, *knn=None*)
Bases: *brew.selection.dynamic.base.DCS*

Overall Local Accuracy.

The Overall Local Accuracy selects the best classifier for a sample using it's K nearest neighbors.

`Xval`
array-like, shape = [indeterminated, n_features] – Validation set.

`yval`
array-like, shape = [indeterminated] – Labels of the validation set.

`knn`
sklearn KNeighborsClassifier, – Classifier used to find neighborhood.

Examples

```
>>> from brew.selection.dynamic.ola import OLA
>>> from brew.generation.bagging import Bagging
>>> from brew.base import EnsembleClassifier
>>>
>>> from sklearn.tree import DecisionTreeClassifier
>>> import numpy as np
>>>
>>> X = np.array([[-1, 0], [-0.8, 1], [-0.8, -1], [-0.5, 0], [0.5, 0],
                  [1, 0], [0.8, 1], [0.8, -1]])
>>> y = np.array([1, 1, 1, 2, 1, 2, 2, 2])
>>> tree = DecisionTreeClassifier(max_depth=1, min_samples_leaf=1)
>>> bag = Bagging(base_classifier=tree, n_classifiers=10)
>>> bag.fit(X, y)
>>>
>>> ola = OLA(X, y, K=3)
>>>
>>> clf = EnsembleClassifier(bag.ensemble, selector=ola)
>>> clf.predict([-1.1,-0.5])
[1]
```

See also:

brew.selection.dynamic.lca.LCA Local Class Accuracy.

References

Woods, Kevin, Kevin Bowyer, and W. Philip Kegelmeyer Jr. "Combination of multiple classifiers using local accuracy estimates." Computer Vision and Pattern Recognition, 1996. Proceedings CVPR'96, 1996 IEEE Computer Society Conference on. IEEE, 1996.

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.

select (*ensemble*, *x*)

Module contents

class `brew.selection.dynamic.`**LCA** (*Xval*, *yval*, *K=5*, *weighted=False*, *knn=None*)

Bases: *`brew.selection.dynamic.base.DCS`*

select (*ensemble*, *x*)

class `brew.selection.dynamic.`**OLA** (*Xval*, *yval*, *K=5*, *weighted=False*, *knn=None*)

Bases: *`brew.selection.dynamic.base.DCS`*

select (*ensemble*, *x*)

class `brew.selection.dynamic.`**APriori** (*Xval*, *yval*, *K=5*, *weighted=False*, *knn=None*, *threshold=0.1*)

Bases: `brew.selection.dynamic.probabilistic.Probabilistic`

A Priori Classifier Selection.

The A Priori method is a dynamic classifier selection that uses a probabilistic-based measures for selecting the best classifier.

`Xval`
array-like, shape = [indeterminated, n_features] – Validation set.

`yval`
array-like, shape = [indeterminated] – Labels of the validation set.

`knn`
sklearn KNeighborsClassifier, – Classifier used to find neighborhood.

`threshold`
float, default = 0.1 – Threshold used to verify if there is a single best.

Examples

```
>>> from brew.selection.dynamic.probabilistic import APriori
>>> from brew.generation.bagging import Bagging
>>> from brew.base import EnsembleClassifier
>>>
>>> from sklearn.tree import DecisionTreeClassifier as Tree
>>> import numpy as np
>>>
>>> X = np.array([[-1, 0], [-0.8, 1], [-0.8, -1], [-0.5, 0] ,
                  [0.5, 0], [1, 0], [0.8, 1], [0.8, -1]])
>>> y = np.array([1, 1, 1, 2, 1, 2, 2, 2])
>>> tree = Tree(max_depth=1, min_samples_leaf=1)
>>> bag = Bagging(base_classifier=tree, n_classifiers=10)
>>> bag.fit(X, y)
>>>
>>> apriori = APriori(X, y, K=3)
>>>
>>> clf = EnsembleClassifier(bag.ensemble, selector=apriori)
>>> clf.predict([-1.1,-0.5])
[1]
```

See also:

`brew.selection.dynamic.probabilistic.APosteriori` A Posteriori DCS.

`brew.selection.dynamic.ola.OLA` Overall Local Accuracy.

`brew.selection.dynamic.lca.LCA` Local Class Accuracy.

References

Giacinto, Giorgio, and Fabio Roli. "Methods for dynamic classifier selection." Image Analysis and Processing, 1999. Proceedings. International Conference on. IEEE, 1999.

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.

`probabilities` (*clf*, *nn_X*, *nn_y*, *distances*, *x*)

`class` `brew.selection.dynamic.`**`APosteriori`** (*Xval*, *yval*, *K=5*, *weighted=False*, *knn=None*, *threshold=0.1*)

Bases: `brew.selection.dynamic.probabilistic.Probabilistic`

A Priori Classifier Selection.

The A Priori method is a dynamic classifier selection that uses a probabilistic-based measures for selecting the best classifier.

`` `Xval` ``
array-like, shape = [indeterminated, n_features] – Validation set.

`` `yval` ``
array-like, shape = [indeterminated] – Labels of the validation set.

`` `knn` ``
sklearn KNeighborsClassifier, – Classifier used to find neighborhood.

`` `threshold` ``
float, default = 0.1 – Threshold used to verify if there is a single best.

Examples

```
>>> from brew.selection.dynamic.probabilistic import APosteriori
>>> from brew.generation.bagging import Bagging
>>> from brew.base import EnsembleClassifier
>>>
>>> from sklearn.tree import DecisionTreeClassifier as Tree
>>> import numpy as np
>>>
>>> X = np.array([[-1, 0], [-0.8, 1], [-0.8, -1],
                  [-0.5, 0] , [0.5, 0], [1, 0],
                  [0.8, 1], [0.8, -1]])
>>> y = np.array([1, 1, 1, 2, 1, 2, 2, 2])
>>> tree = Tree(max_depth=1, min_samples_leaf=1)
>>> bag = Bagging(base_classifier=tree, n_classifiers=10)
>>> bag.fit(X, y)
>>>
>>> aposteriori = APosteriori(X, y, K=3)
>>>
```

(continues on next page)

(continued from previous page)

```
>>> clf = EnsembleClassifier(bag.ensemble, selector=aposteriori)
>>> clf.predict([-1.1,-0.5])
[1]
```

See also:

`brew.selection.dynamic.probabilistic.APriori` A Priori DCS.

`brew.selection.dynamic.ola.OLA` Overall Local Accuracy.

`brew.selection.dynamic.lca.LCA` Local Class Accuracy.

References

Giacinto, Giorgio, and Fabio Roli. “Methods for dynamic classifier selection.” Image Analysis and Processing, 1999. Proceedings. International Conference on. IEEE, 1999.

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.

probabilities(*clf*, *nn_X*, *nn_y*, *distances*, *x*)

class brew.selection.dynamic.**KNORA_UNION**(*Xval*, *yval*, *K=5*, *weighted=False*, *knn=None*)

Bases: *brew.selection.dynamic.knora.KNORA*

K-nearest-oracles Union.

The KNORA union reduces the neighborhood until finds an ensemble of classifiers that correctly classify all neighbors.

`Xval`
array-like, shape = [indeterminated, n_features] – Validation set.

`yval`
array-like, shape = [indeterminated] – Labels of the validation set.

`knn`
sklearn KNeighborsClassifier, – Classifier used to find neighborhood.

`weighted`
bool, (makes no difference in knora_eliminate) – Bool that defines if the classifiers uses weights or not

Examples

```
>>> from brew.selection.dynamic.knora import KNORA_UNION
>>> from brew.generation.bagging import Bagging
>>> from brew.base import EnsembleClassifier
>>>
>>> from sklearn.tree import DecisionTreeClassifier
>>> import numpy as np
>>>
>>> X = np.array([[-1, 0], [-0.8, 1], [-0.8, -1], [-0.5, 0],
                  [0.5, 0], [1, 0], [0.8, 1], [0.8, -1]])
>>> y = np.array([1, 1, 1, 2, 1, 2, 2, 2])
>>>
>>> dt = DecisionTreeClassifier(max_depth=1, min_samples_leaf=1)
```

(continues on next page)

(continued from previous page)

```
>>> bag = Bagging(base_classifier=dt, n_classifiers=10)
>>> bag.fit(X, y)
>>>
>>> ku = KNORA_UNION(X, y, K=5)
>>>
>>> clf = EnsembleClassifier(bag.ensemble, selector=ku)
>>> clf.predict([-1.1,-0.5])
[1]
```

See also:

brew.selection.dynamic.knora.KNORA_ELIMINATE Knora Eliminate.

brew.selection.dynamic.lca.LCA Local Class Accuracy.

brew.selection.dynamic.ola.OLA Overall Local Accuracy.

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.

Hung-Ren Ko, A., Robert Sabourin, and A. de Souza Britto. "K-nearest oracle for dynamic ensemble selection." Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on. Vol. 1. IEEE, 2007.

select (*ensemble*, *x*)

class brew.selection.dynamic.**KNORA_ELIMINATE** (*Xval*, *yval*, *K=5*, *weighted=False*, *knn=None*, *v2007=False*)

Bases: *brew.selection.dynamic.knora.KNORA*

K-nearest-oracles Eliminate.

The KNORA Eliminate reduces the neighborhood until finds an ensemble of classifiers that correctly classify all neighbors.

`Xval`
array-like, shape = [indeterminated, n_features] – Validation set.

`yval`
array-like, shape = [indeterminated] – Labels of the validation set.

`knn`
sklearn KNeighborsClassifier, – Classifier used to find neighborhood.

`weighted`
bool, (makes no difference in knora_eliminate) – Bool that defines if the classifiers uses weights or not

Examples

```
>>> from brew.selection.dynamic.knora import KNORA_ELIMINATE
>>> from brew.generation.bagging import Bagging
>>> from brew.base import EnsembleClassifier
```

(continues on next page)

(continued from previous page)

```
>>>
>>> from sklearn.tree import DecisionTreeClassifier
>>> import numpy as np
>>>
>>> X = np.array([[-1, 0], [-0.8, 1], [-0.8, -1], [-0.5, 0],
                  [0.5, 0], [1, 0], [0.8, 1], [0.8, -1]])
>>> y = np.array([1, 1, 1, 2, 1, 2, 2, 2])
>>>
>>> dt = DecisionTreeClassifier(max_depth=1, min_samples_leaf=1)
>>> bag = Bagging(base_classifier=dt, n_classifiers=10)
>>> bag.fit(X, y)
>>>
>>> ke = KNORA_ELIMINATE(X, y, K=5)
>>>
>>> clf = EnsembleClassifier(bag.ensemble, selector=ke)
>>> clf.predict([-1.1,-0.5])
[1]
```

See also:

`brew.selection.dynamic.knora.KNORA_UNION` KNORA Union.

`brew.selection.dynamic.lca.LCA` Local Class Accuracy.

`brew.selection.dynamic.ola.OLA` Overall Local Accuracy.

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.

Hung-Ren Ko, A., Robert Sabourin, and A. de Souza Britto. "K-nearest oracle for dynamic ensemble selection." Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on. Vol. 1. IEEE, 2007

select (*ensemble*, *x*)

class brew.selection.dynamic.**MCB** (*Xval*, *yval*, *K=5*, *weighted=False*, *knn=None*, *similarity_threshold=0.7*, *significance_threshold=0.3*)

Bases: `brew.selection.dynamic.base.DCS`

Multiple Classifier Behavior.

The Multiple Classifier Behavior (MCB) selects the best classifier using the similarity of the classifications on the K neighbors of the test sample in the validation set.

`Xval`
array-like, shape = [indeterminated, n_features] – Validation set.

`yval`
array-like, shape = [indeterminated] – Labels of the validation set.

`knn`
sklearn KNeighborsClassifier, – Classifier used to find neighborhood.

Examples

```
>>> from brew.selection.dynamic.mcb import MCB
>>> from brew.generation.bagging import Bagging
>>> from brew.base import EnsembleClassifier
>>>
>>> from sklearn.tree import DecisionTreeClassifier
>>> import numpy as np
>>>
>>> X = np.array([[-1, 0], [-0.8, 1], [-0.8, -1], [-0.5, 0],
                  [0.5, 0], [1, 0], [0.8, 1], [0.8, -1]])
>>> y = np.array([1, 1, 1, 2, 1, 2, 2, 2])
>>> tree = DecisionTreeClassifier(max_depth=1, min_samples_leaf=1)
>>> bag = Bagging(base_classifier=tree, n_classifiers=10)
>>> bag.fit(X, y)
>>>
>>> mcb = MCB(X, y, K=3)
>>>
>>> clf = EnsembleClassifier(bag.ensemble, selector=mcb)
>>> clf.predict([-1.1,-0.5])
[1]
```

See also:

`brew.selection.dynamic.lca.OLA` Overall Local Accuracy.

`brew.selection.dynamic.lca.LCA` Local Class Accuracy.

References

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

`select` (*ensemble*, *x*)

`class` `brew.selection.dynamic.`**`DSKNN`**(*Xval*, *yval*, *K=5*, *weighted=False*, *knn=None*, *n_1=0.7*, *n_2=0.3*)

Bases: *`brew.selection.dynamic.base.DCS`*

DS-KNN

The DS-KNN selects an ensemble of classifiers based on their accuracy and diversity in the neighborhood of the test sample.

`` `Xval` ``
array-like, shape = [indeterminated, n_features] – Validation set.

`` `yval` ``
array-like, shape = [indeterminated] – Labels of the validation set.

`` `knn` ``
sklearn KNeighborsClassifier, – Classifier used to find neighborhood.

Examples

```
>>> from brew.selection.dynamic import DSKNN
>>> from brew.generation.bagging import Bagging
>>> from brew.base import EnsembleClassifier
>>>
>>> from sklearn.tree import DecisionTreeClassifier
>>> import numpy as np
>>>
>>> X = np.array([[-1, 0], [-0.8, 1], [-0.8, -1], [-0.5, 0],
                  [0.5, 0], [1, 0], [0.8, 1], [0.8, -1]])
>>> y = np.array([1, 1, 1, 2, 1, 2, 2, 2])
>>> tree = DecisionTreeClassifier(max_depth=1, min_samples_leaf=1)
>>> bag = Bagging(base_classifier=tree, n_classifiers=10)
>>> bag.fit(X, y)
>>>
>>> sel = DSKNN(X, y, K=3)
>>>
>>> clf = EnsembleClassifier(bag.ensemble, selector=sel)
>>> clf.predict([-1.1,-0.5])
[1]
```

See also:

`brew.selection.dynamic.lca.OLA` Overall Local Accuracy.

`brew.selection.dynamic.lca.LCA` Local Class Accuracy.

References

Santana, Alixandre, et al. "A dynamic classifier selection method to build ensembles using accuracy and diversity." 2006 Ninth Brazilian Symposium on Neural Networks (SBRN'06). IEEE, 2006.

`select` (*ensemble*, *x*)

brew.selection.pruning package

Submodules

brew.selection.pruning.epic module

class `brew.selection.pruning.epic.`**`EPIC`**
Bases: `brew.selection.pruning.base.Prunner`

`fit` (*ensemble*, *X*, *y*)

`get` (*p=0.1*)

Module contents

class `brew.selection.pruning.`**`EPIC`**
Bases: `brew.selection.pruning.base.Prunner`

`fit` (*ensemble*, *X*, *y*)

`get` (*p=0.1*)

Module contents

brew.utils package

Submodules

brew.utils.data module

`brew.utils.data.`**`split_data`**`(`*X*, *y*, *t_size*`)`

Module contents

6.1.2 Submodules

6.1.3 brew.base module

`class` `brew.base.`**`BrewClassifier`**`(`*classifier=None*, *transformer=None*`)`

Bases: `object`

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

`predict``(`*X*`)`

`predict_proba``(`*X*`)`

`class` `brew.base.`**`Ensemble`**`(`*classifiers=None*`)`

Bases: `object`

Class that represents a collection of classifiers.

The Ensemble class serves as a wrapper for a list of classifiers, besides providing a simple way to calculate the output of all the classifiers in the ensemble.

`` `classifiers` ``

list – Stores all classifiers in the ensemble.

`` `yval` ``

array-like, shape = [indeterminated] – Labels of the validation set.

`` `knn` ``

sklearn KNeighborsClassifier, – Classifier used to find neighborhood.

Examples

```
>>> import numpy as np
>>> from sklearn.tree import DecisionTreeClassifier
>>>
>>> from brew.base import Ensemble
>>>
>>> X = np.array([[-1, 0], [-0.8, 1], [-0.8, -1], [-0.5, 0],
                  [0.5, 0], [1, 0], [0.8, 1], [0.8, -1]])
>>> y = np.array([1, 1, 1, 2, 1, 2, 2, 2])
>>>
>>> dt1 = DecisionTreeClassifier()
>>> dt2 = DecisionTreeClassifier()
```

(continues on next page)

(continued from previous page)

```
>>>
>>> dt1.fit(X, y)
>>> dt2.fit(X, y)
>>>
>>> ens = Ensemble(classifiers=[dt1, dt2])
```

add (*classifier*)

add_classifiers (*classifiers*)

add_ensemble (*ensemble*)

fit (*X*, *y*)
warning: this fit overrides previous generated base classifiers!

get_classes ()

in_agreement (*x*)

output (*X*, *mode='votes'*)
Returns the output of all classifiers packed in a numpy array.

This method calculates the output of each classifier, and stores them in a array-like shape. The specific shape and the meaning of each element is defined by argument *mode*.

(1) 'labels': each classifier will return a single label prediction for each sample in X, therefore the ensemble output will be a 2d-array of shape (n_samples, n_classifiers), with elements being the class labels.

(2) 'probs': each classifier will return the posterior probabilities of each class (i.e. instead of returning a single choice it will return the probabilities of each class label being the right one). The ensemble output will be a 3d-array with shape (n_samples, n_classes, n_classifiers), with each element being the probability of a specific class label being right on a given sample according to one the classifiers. This mode can be used with any combination rule.

(3) 'votes': each classifier will return votes for each class label i.e. a binary representation, where the chosen class label will have one vote and the other labels will have zero votes. The ensemble output will be a binary 3d-array with shape (n_samples, n_classes, n_classifiers), with the elements being the votes. This mode is mainly used in combining the classifiers output by using majority vote rule.

Parameters

- **X** (*array-like, shape = [n_samples, n_features]*) – The test input samples.
- **mode** (*string, optional (default='labels')*) – The type of output given by each classifier. 'labels' | 'probs' | 'votes'

output_simple (*X*)

class brew.base.**EnsembleClassifier** (*ensemble=None*, *selector=None*, *combiner=None*)
Bases: object

fit (*X*, *y*)

predict (*X*)

predict_proba (*X*)

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

class brew.base.**FeatureSubsamplingTransformer** (*features=None*)
Bases: *brew.base.Transformer*

apply (*X*)

class brew.base.**Transformer**

Bases: object

apply(*X*)

brew.base.**oracle**(*ensemble*, *X*, *y_true*, *metric=<function auc_score>*)

brew.base.**single_best**(*ensemble*, *X*, *y_true*, *metric=<function auc_score>*)

brew.base.**transform2votes**(*output*, *n_classes*)

6.1.4 Module contents

class brew.**Ensemble**(*classifiers=None*)

Bases: object

Class that represents a collection of classifiers.

The Ensemble class serves as a wrapper for a list of classifiers, besides providing a simple way to calculate the output of all the classifiers in the ensemble.

`classifiers`

list – Stores all classifiers in the ensemble.

`yval`

array-like, shape = [indeterminated] – Labels of the validation set.

`knn`

sklearn KNeighborsClassifier, – Classifier used to find neighborhood.

Examples

```
>>> import numpy as np
>>> from sklearn.tree import DecisionTreeClassifier
>>>
>>> from brew.base import Ensemble
>>>
>>> X = np.array([[-1, 0], [-0.8, 1], [-0.8, -1], [-0.5, 0],
                  [0.5, 0], [1, 0], [0.8, 1], [0.8, -1]])
>>> y = np.array([1, 1, 1, 2, 1, 2, 2, 2])
>>>
>>> dt1 = DecisionTreeClassifier()
>>> dt2 = DecisionTreeClassifier()
>>>
>>> dt1.fit(X, y)
>>> dt2.fit(X, y)
>>>
>>> ens = Ensemble(classifiers=[dt1, dt2])
```

add(*classifier*)

add_classifiers(*classifiers*)

add_ensemble(*ensemble*)

fit(*X*, *y*)

warning: this fit overrides previous generated base classifiers!

get_classes()

in_agreement (*x*)

output (*X*, *mode='votes'*)

Returns the output of all classifiers packed in a numpy array.

This method calculates the output of each classifier, and stores them in a array-like shape. The specific shape and the meaning of each element is defined by argument *mode*.

(1) 'labels': each classifier will return a single label prediction for each sample in X, therefore the ensemble output will be a 2d-array of shape (n_samples, n_classifiers), with elements being the class labels.

(2) 'probs': each classifier will return the posterior probabilities of each class (i.e. instead of returning a single choice it will return the probabilities of each class label being the right one). The ensemble output will be a 3d-array with shape (n_samples, n_classes, n_classifiers), with each element being the probability of a specific class label being right on a given sample according to one the classifiers. This mode can be used with any combination rule.

(3) 'votes': each classifier will return votes for each class label i.e. a binary representation, where the chosen class label will have one vote and the other labels will have zero votes. The ensemble output will be a binary 3d-array with shape (n_samples, n_classes, n_classifiers), with the elements being the votes. This mode is mainly used in combining the classifiers output by using majority vote rule.

Parameters

- **X** (*array-like, shape = [n_samples, n_features]*) – The test input samples.
- **mode** (*string, optional (default='labels')*) – The type of output given by each classifier. 'labels' | 'probs' | 'votes'

output_simple (*X*)

class brew.**EnsembleClassifier** (*ensemble=None*, *selector=None*, *combiner=None*)

Bases: `object`

fit (*X*, *y*)

predict (*X*)

predict_proba (*X*)

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

6.2 Installation

At the command line either via easy_install or pip:

```
$ easy_install brew
$ pip install brew
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv brew
$ pip install brew
```

6.3 Usage

To use brew in a project:

```python
import brew
from brew.base import Ensemble
from brew.base import EnsembleClassifier
from brew.combination import Combiner

# here, clf1 and clf2 are sklearn classifiers or brew ensemble classifiers
# already trained. Keep in mind that brew requires your labels = [0,1,2,...]
# numerical with no skips.
clfs = [clf1, clf2]
ens = Ensemble(classifiers = clfs)

# create your Combiner
# the rules can be 'majority_vote', 'max', 'min', 'mean' or 'median'
comb = Combiner(rule='mean')

# now create your ensemble classifier
ensemble_clf = EnsembleClassifier(ensemble=ens, combiner=comb)
y_pred = ensemble_clf.predict(X)

# there you go, y_pred is your prediction.
```

6.4 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

6.4.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/viisar/brew/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

brew could always use more documentation, whether as part of the official brew docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/viisar/brew/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

6.4.2 Get Started!

Ready to contribute? Here's how to set up *brew* for local development.

1. **Fork_** the *brew* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/brew.git
```

3. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

4. When you're done making changes, check that your changes pass style and unit tests, including testing other Python versions with tox:

```
$ tox
```

To get tox, just pip install it.

5. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

6.4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.

3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/viisar/brew under pull requests for active pull requests or run the `tox` command and make sure that the tests pass for all supported Python versions.

6.4.4 Tips

To run a subset of tests:

```
$ py.test test/test_brew.py
```

6.5 Credits

This project was started in 2014 by *Dayvid Victor* and *Thyago Porpino* as a project for the Multiple Classifier Systems class at Federal University of Pernambuco (UFPE).

Nowdays, this project is part of Dayvid's PhD thesis on Ensemble Learning.

6.5.1 Development Lead

- Dayvid Victor <victor.dvro@gmail.com>
- Thyago Porpino <thyago.porpino@gmail.com>

6.5.2 Contributors

None yet. Why not be the first?

6.6 History

6.6.1 0.1.0 (2014-11-12)

- First release on PyPI.

CHAPTER 7

Feedback

If you have any suggestions or questions about **brew** feel free to email me at victor.dvro@gmail.com.

If you encounter any errors or problems with **brew**, please let me know! Open an Issue at the GitHub http://github.com/dvro/brew main repository.

Python Module Index

Index