sklearn.ensemble.BaggingClassifier

» *class* `sklearn.ensemble.`**`BaggingClassifier`**(*base_estimator=None*, *n_estimators=10*, *max_samples=1.0*, *max_features=1.0*, *bootstrap=True*, *bootstrap_features=False*, *oob_score=False*, *warm_start=False*, *n_jobs=None*, *random_state=None*, *verbose=0*) ¶ [source]

A Bagging classifier.

A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it.

This algorithm encompasses several works from the literature. When random subsets of the dataset are drawn as random subsets of the samples, then this algorithm is known as Pasting [Rb1846455-d0e5-1]. If samples are drawn with replacement, then the method is known as Bagging [Rb1846455-d0e5-2]. When random subsets of the dataset are drawn as random subsets of the features, then the method is known as Random Subspaces [Rb1846455d0e5-3]. Finally, when base estimators are built on subsets of both samples and features, then the method is known as Random Patches [Rb1846455d0e5-4].

Read more in the User Guide.

Parameters:

base_estimator : *object or None, optional (default=None)*

The base estimator to fit on random subsets of the dataset. If None, then the base estimator is a decision tree.

n_estimators : *int, optional (default=10)*

The number of base estimators in the ensemble.

max_samples : *int or float, optional (default=1.0)*

The number of samples to draw from X to train each base estimator.

- If int, then draw `max_samples` samples.
- If float, then draw `max_samples * X.shape[0]` samples.

max_features : *int or float, optional (default=1.0)*

The number of features to draw from X to train each base estimator.

- If int, then draw `max_features` features.

- If float, then draw `max_features * X.shape[1]` features.

bootstrap : *boolean, optional (default=True)*

Whether samples are drawn with replacement. If False, sampling without replacement is performed.

bootstrap_features : *boolean, optional (default=False)*

Whether features are drawn with replacement.

oob_score : *bool, optional (default=False)*

Whether to use out-of-bag samples to estimate the generalization error.

warm_start : *bool, optional (default=False)*

When set to True, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just fit a whole new ensemble. See the Glossary.

New in version 0.17: warm_start constructor parameter.

n_jobs : *int or None, optional (default=None)*

The number of jobs to run in parallel for both **`fit`** and **`predict`**. `None` means 1 unless in a **`joblib.parallel_backend`** context. `-1` means using all processors. See Glossary for more details.

random_state : *int, RandomState instance or None, optional (default=None)*

If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by `np.random`.

verbose : *int, optional (default=0)*

Controls the verbosity when fitting and predicting.

Attributes:

base_estimator_ : *estimator*

The base estimator from which the ensemble is grown.

estimators_ : *list of estimators*

The collection of fitted base estimators.

`estimators_samples_` : *list of arrays*

The subset of drawn samples for each base estimator.

estimators_features_ : *list of arrays*

The subset of drawn features for each base estimator.

classes_ : *array of shape = [n_classes]*

The classes labels.

n_classes_ : *int or list*

The number of classes.

oob_score_ : *float*

Score of the training dataset obtained using an out-of-bag estimate.

oob_decision_function_ : *array of shape = [n_samples, n_classes]*

Decision function computed with out-of-bag estimate on the training set. If n_estimators is small it might be possible that a data point was never left out during the bootstrap. In this case, `oob_decision_function_` might contain NaN.

References

[Rb1846455d0e5-1] L. Breiman, “Pasting small votes for classification in large databases and on-line”, Machine Learning, 36(1), 85-103, 1999.

[Rb1846455d0e5-2] L. Breiman, “Bagging predictors”, Machine Learning, 24(2), 123-140, 1996.

[Rb1846455d0e5-3] T. Ho, “The random subspace method for constructing decision forests”, Pattern Analysis and Machine Intelligence, 20(8), 832-844, 1998.

[Rb1846455d0e5-4] G. Louppe and P. Geurts, “Ensembles on Random Patches”, Machine Learning and Knowledge Discovery in Databases, 346-361, 2012.

Methods

decision_function(self, X)	Average of the decision functions of the base classifiers.
fit(self, X, y[, sample_weight])	Build a Bagging ensemble of estimators from the training set (X, y).
get_params(self[, deep])	Get parameters for this estimator.
predict(self, X)	Predict class for X.
predict_log_proba(self, X)	Predict class log-probabilities for X.
predict_proba(self, X)	Predict class probabilities for X.
score(self, X, y[, sample_weight])	Returns the mean accuracy on the given test data and labels.
set_params(self, **params)	Set the parameters of this estimator.

__init__(*self*, *base_estimator=None*, *n_estimators=10*, *max_samples=1.0*, *max_features=1.0*, *bootstrap=True*, *bootstrap_features=False*, *oob_score=False*, *warm_start=False*, *n_jobs=None*, *random_state=None*, *verbose=0*) [source]

decision_function(*self*, *X*) [source]

Average of the decision functions of the base classifiers.

Parameters: **X** : *{array-like, sparse matrix} of shape = [n_samples, n_features]*

The training input samples. Sparse matrices are accepted only if they are supported by the base estimator.

Returns: **score** : *array, shape = [n_samples, k]*

The decision function of the input samples. The columns correspond to the classes in sorted order, as they appear in the attribute `classes_`. Regression and binary classification are special cases with `k == 1`, otherwise `k==n_classes`.

estimators_samples_

The subset of drawn samples for each base estimator.

Returns a dynamically generated list of indices identifying the samples used for fitting each member of the ensemble, i.e., the in-bag samples.

Note: the list is re-created at each call to the property in order to reduce the object memory footprint by not storing the sampling data. Thus fetching the property may be slower than expected.

fit(*self*, *X*, *y*, *sample_weight=None*) [source]

Build a Bagging ensemble of estimators from the training
set (X, y).

Parameters: **X** : *{array-like, sparse matrix} of shape = [n_samples, n_features]*

The training input samples. Sparse matrices are accepted only if they are supported by the base estimator.

y : *array-like, shape = [n_samples]*

The target values (class labels in classification, real numbers in regression).

sample_weight : *array-like, shape = [n_samples] or None*

Sample weights. If None, then samples are equally weighted. Note that this is supported only if the base estimator supports sample weighting.

Returns: **self** : *object*

get_params(*self*, *deep=True*) [source]

Get parameters for this estimator.

Parameters: **deep** : *boolean, optional*

If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns: **params** : *mapping of string to any*

Parameter names mapped to their values.

predict(*self*, *X*) [source]

Predict class for X.

The predicted class of an input sample is computed as the class with the highest mean predicted probability. If base estimators do not implement a `predict_proba` method, then it resorts to voting.

Parameters: **X** : *{array-like, sparse matrix} of shape = [n_samples, n_features]*

The training input samples. Sparse matrices are accepted only if they are supported by the base estimator.

Returns: **y** : *array of shape = [n_samples]*

The predicted classes.

predict_log_proba(*self*, *X*) [source]

Predict class log-probabilities for X.

The predicted class log-probabilities of an input sample is computed as the log of the mean predicted class probabilities of the base estimators in the ensemble.

Parameters: **X** : *{array-like, sparse matrix} of shape = [n_samples, n_features]*

The training input samples. Sparse matrices are accepted only if they are supported by the base estimator.

Returns: **p** : *array of shape = [n_samples, n_classes]*

The class log-probabilities of the input samples. The order of the classes corresponds to that in the attribute classes_.

predict_proba(*self*, *X*) [source]

Predict class probabilities for X.

The predicted class probabilities of an input sample is computed as the mean predicted class probabilities of the base estimators in the ensemble. If base estimators do not implement a `predict_proba` method, then it resorts to voting and the predicted class probabilities of an input sample represents the proportion of estimators predicting each class.

Parameters: **X** : *{array-like, sparse matrix} of shape = [n_samples, n_features]*

The training input samples. Sparse matrices are accepted only if they are supported by the base estimator.

Returns: **p** : *array of shape = [n_samples, n_classes]*

The class probabilities of the input samples. The order of the classes corresponds to that in the attribute classes_.

score(*self*, *X*, *y*, *sample_weight=None*) [source]

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.

set_params(*self*, ***params*) [source]

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Returns: self

Previous Next