sklearn.base (version 1.1.3)

index
/usr/local/python/3.10.1/lib/python3.10/site-packages/sklearn/base.py

```
Base classes for all estimators.
```

Modules

copy	numpy	re
inspect	platform	warnings

Classes

builtins.object
- BaseEstimator
- BiclusterMixin
- ClassifierMixin
- ClusterMixin
- DensityMixin
- MetaEstimatorMixin
- MultiOutputMixin
- OutlierMixin
- RegressorMixin
- TransformerMixin

class **BaseEstimator**(builtins.object)

```
Base class for all estimators in scikit-learn.

Notes
-----
All estimators should specify all the parameters that can be set
at the class level in their ``__init__`` as explicit keyword
arguments (no ``*args`` or ``**kwargs``).
```

Methods defined here:

__getstate__(self)

__repr__(self, N_CHAR_MAX=700)
```
Return repr(self).
```

__setstate__(self, state)

get_params(self, deep=True)
```
Get parameters for this estimator.

Parameters
----------
deep : bool, default=True
    If True, will return the parameters for this estimator and
    contained subobjects that are estimators.

Returns
-------
params : dict
```

```
        Parameter names mapped to their values.
```

set_params(self, **params)

```
    Set the parameters of this estimator.

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

    Parameters
    ----------
    **params : dict
        Estimator parameters.

    Returns
    -------
    self : estimator instance
        Estimator instance.
```

Data descriptors defined here:

__dict__

```
    dictionary for instance variables (if defined)
```

__weakref__

```
    list of weak references to the object (if defined)
```

class **BiclusterMixin**(builtins.object)

```
Mixin class for all bicluster estimators in scikit-learn.
```

Methods defined here:

get_indices(self, i)

```
    Row and column indices of the `i`'th bicluster.

    Only works if ``rows_`` and ``columns_`` attributes exist.

    Parameters
    ----------
    i : int
        The index of the cluster.

    Returns
    -------
    row_ind : ndarray, dtype=np.intp
        Indices of rows in the dataset that belong to the bicluster.
    col_ind : ndarray, dtype=np.intp
        Indices of columns in the dataset that belong to the bicluster.
```

get_shape(self, i)

```
    Shape of the `i`'th bicluster.

    Parameters
    ----------
    i : int
        The index of the cluster.

    Returns
```

```
        -------
        n_rows : int
            Number of rows in the bicluster.

        n_cols : int
            Number of columns in the bicluster.
```

get_submatrix(self, i, data)

```
        Return the submatrix corresponding to bicluster `i`.

        Parameters
        ----------
        i : int
            The index of the cluster.
        data : array-like of shape (n_samples, n_features)
            The data.

        Returns
        -------
        submatrix : ndarray of shape (n_rows, n_cols)
            The submatrix corresponding to bicluster `i`.

        Notes
        -----
        Works with sparse matrices. Only works if ``rows_`` and
        ``columns_`` attributes exist.
```

Readonly properties defined here:

biclusters_

```
        Convenient way to get row and column indicators together.

        Returns the ``rows_`` and ``columns_`` members.
```

Data descriptors defined here:

__dict__

```
        dictionary for instance variables (if defined)
```

__weakref__

```
        list of weak references to the object (if defined)
```

class **ClassifierMixin**(builtins.object)

```
 Mixin class for all classifiers in scikit-learn.
```

Methods defined here:

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

```
        Return 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 of shape (n_samples, n_features)
            Test samples.
```

```
y : array-like of shape (n_samples,) or (n_samples, n_outputs)
    True labels for `X`.

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

Returns
-------
score : float
    Mean accuracy of ``self.predict(X)`` wrt. `y`.
```

Data descriptors defined here:

__dict__

```
dictionary for instance variables (if defined)
```

__weakref__

```
list of weak references to the object (if defined)
```

class **ClusterMixin**(builtins.object)

```
Mixin class for all cluster estimators in scikit-learn.
```

Methods defined here:

fit_predict(self, X, y=None)

```
Perform clustering on `X` and returns cluster labels.

Parameters
----------
X : array-like of shape (n_samples, n_features)
    Input data.

y : Ignored
    Not used, present for API consistency by convention.

Returns
-------
labels : ndarray of shape (n_samples,), dtype=np.int64
    Cluster labels.
```

Data descriptors defined here:

__dict__

```
dictionary for instance variables (if defined)
```

__weakref__

```
list of weak references to the object (if defined)
```

class **DensityMixin**(builtins.object)

```
Mixin class for all density estimators in scikit-learn.
```

Methods defined here:

score(self, X, y=None)

```
        Return the score of the model on the data `X`.

        Parameters
        ----------
        X : array-like of shape (n_samples, n_features)
            Test samples.

        y : Ignored
            Not used, present for API consistency by convention.

        Returns
        -------
        score : float
```

Data descriptors defined here:

__dict__
```
        dictionary for instance variables (if defined)
```

__weakref__
```
        list of weak references to the object (if defined)
```

class **MetaEstimatorMixin**(builtins.object)

Data descriptors defined here:

__dict__
```
        dictionary for instance variables (if defined)
```

__weakref__
```
        list of weak references to the object (if defined)
```

class **MultiOutputMixin**(builtins.object)

```
Mixin to mark estimators that support multioutput.
```

Data descriptors defined here:

__dict__
```
        dictionary for instance variables (if defined)
```

__weakref__
```
        list of weak references to the object (if defined)
```

class **OutlierMixin**(builtins.object)

```
Mixin class for all outlier detection estimators in scikit-learn.
```

Methods defined here:

fit_predict(self, X, y=None)
```
        Perform fit on X and returns labels for X.

        Returns -1 for outliers and 1 for inliers.

        Parameters
        ----------
```

```
X : {array-like, sparse matrix} of shape (n_samples, n_features)
    The input samples.

y : Ignored
    Not used, present for API consistency by convention.

Returns
-------
y : ndarray of shape (n_samples,)
    1 for inliers, -1 for outliers.
```

Data descriptors defined here:

__dict__
```
dictionary for instance variables (if defined)
```

__weakref__
```
list of weak references to the object (if defined)
```

class **RegressorMixin**(builtins.object)

```
Mixin class for all regression estimators in scikit-learn.
```

Methods defined here:

score(self, X, y, sample_weight=None)
```
Return the coefficient of determination of the prediction.

The coefficient of determination :math:`R^2` is defined as
:math:`(1 - \frac{u}{v})`, where :math:`u` is the residual
sum of squares ``((y_true - y_pred)** 2).sum()`` and :math:`v`
is the total sum of squares ``((y_true - y_true.mean()) ** 2).sum()``.
The best possible score is 1.0 and it can be negative (because the
model can be arbitrarily worse). A constant model that always predicts
the expected value of `y`, disregarding the input features, would get
a :math:`R^2` score of 0.0.

Parameters
----------
X : array-like of shape (n_samples, n_features)
    Test samples. For some estimators this may be a precomputed
    kernel matrix or a list of generic objects instead with shape
    ``(n_samples, n_samples_fitted)``, where ``n_samples_fitted``
    is the number of samples used in the fitting for the estimator.

y : array-like of shape (n_samples,) or (n_samples, n_outputs)
    True values for `X`.

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

Returns
-------
score : float
    :math:`R^2` of ``self.predict(X)`` wrt. `y`.

Notes
-----
The :math:`R^2` score used when calling ``score`` on a regressor uses
``multioutput='uniform_average'`` from version 0.23 to keep consistent
```

```
with default value of :func:`~sklearn.metrics.r2_score`.
This influences the ``score`` method of all the multioutput
regressors (except for
:class:`~sklearn.multioutput.MultiOutputRegressor`).
```

Data descriptors defined here:

__dict__

```
dictionary for instance variables (if defined)
```

__weakref__

```
list of weak references to the object (if defined)
```

class **TransformerMixin**(builtins.object)

```
Mixin class for all transformers in scikit-learn.
```

Methods defined here:

fit_transform(self, X, y=None, **fit_params)

```
Fit to data, then transform it.

Fits transformer to `X` and `y` with optional parameters `fit_params`
and returns a transformed version of `X`.

Parameters
----------
X : array-like of shape (n_samples, n_features)
    Input samples.

y :  array-like of shape (n_samples,) or (n_samples, n_outputs),                default=None
    Target values (None for unsupervised transformations).

**fit_params : dict
    Additional fit parameters.

Returns
-------
X_new : ndarray array of shape (n_samples, n_features_new)
    Transformed array.
```

Data descriptors defined here:

__dict__

```
dictionary for instance variables (if defined)
```

__weakref__

```
list of weak references to the object (if defined)
```

Functions

clone(estimator, *, safe=True)

```
Construct a new unfitted estimator with the same parameters.

Clone does a deep copy of the model in an estimator
without actually copying attached data. It returns a new estimator
with the same parameters that has not been fitted on any data.
```

```
Parameters
----------
estimator : {list, tuple, set} of estimator instance or a single             estimator instance
    The estimator or group of estimators to be cloned.
safe : bool, default=True
    If safe is False, clone will fall back to a deep copy on objects
    that are not estimators.

Returns
-------
estimator : object
    The deep copy of the input, an estimator if input is an estimator.

Notes
-----
If the estimator's `random_state` parameter is an integer (or if the
estimator doesn't have a `random_state` parameter), an *exact clone* is
returned: the clone and the original estimator will give the exact same
results. Otherwise, *statistical clone* is returned: the clone might
return different results from the original estimator. More details can be
found in :ref:`randomness`.
```

is_classifier(estimator)

```
Return True if the given estimator is (probably) a classifier.

Parameters
----------
estimator : object
    Estimator object to test.

Returns
-------
out : bool
    True if estimator is a classifier and False otherwise.
```

is_outlier_detector(estimator)

```
Return True if the given estimator is (probably) an outlier detector.

Parameters
----------
estimator : estimator instance
    Estimator object to test.

Returns
-------
out : bool
    True if estimator is an outlier detector and False otherwise.
```

is_regressor(estimator)

```
Return True if the given estimator is (probably) a regressor.

Parameters
----------
estimator : estimator instance
    Estimator object to test.

Returns
-------
out : bool
    True if estimator is a regressor and False otherwise.
```