DummyClassifier

class **sklearn.dummy.DummyClassifier**(**, strategy='prior', random_state=None, constant=None*) [source]

DummyClassifier makes predictions that ignore the input features.

This classifier serves as a simple baseline to compare against other more complex classifiers.

The specific behavior of the baseline is selected with the `strategy` parameter.

All strategies make predictions that ignore the input feature values passed as the `X` argument to `fit` and `predict`. The predictions, however, typically depend on values observed in the `y` parameter passed to `fit`.

Note that the "stratified" and "uniform" strategies lead to non-deterministic predictions that can be rendered deterministic by setting the `random_state` parameter if needed. The other strategies are naturally deterministic and, once fit, always return the same constant prediction for any value of `X`.

Read more in the User Guide.

Added in version 0.13.

Parameters:

strategy : *{"most_frequent", "prior", "stratified", "uniform", "constant"}, default="prior"*

Strategy to use to generate predictions.

- "most_frequent": the `predict` method always returns the most frequent class label in the observed `y` argument passed to `fit`. The `predict_proba` method returns the matching one-hot encoded vector.
- "prior": the `predict` method always returns the most frequent class label in the observed `y` argument passed to `fit` (like "most_frequent"). `predict_proba` always returns the empirical class distribution of `y` also known as the empirical class prior distribution.
- "stratified": the `predict_proba` method randomly samples one-hot vectors from a multinomial distribution parametrized by the empirical class prior probabilities. The `predict` method returns the class label which got probability one in the one-hot

vector of `predict_proba`. Each sampled row of both methods is therefore independent and identically distributed.

- "uniform": generates predictions uniformly at random from the list of unique classes observed in `y`, i.e. each class has equal probability.
- "constant": always predicts a constant label that is provided by the user. This is useful for metrics that evaluate a non-majority class.

> *Changed in version 0.24:* The default value of `strategy` has changed to "prior" in version 0.24.

random_state : *int, RandomState instance or None, default=None*

Controls the randomness to generate the predictions when `strategy='stratified'` or `strategy='uniform'`. Pass an int for reproducible output across multiple function calls. See Glossary.

constant : *int or str or array-like of shape (n_outputs,), default=None*

The explicit constant as predicted by the "constant" strategy. This parameter is useful only for the "constant" strategy.

Attributes:

classes_ : *ndarray of shape (n_classes,) or list of such arrays*

Unique class labels observed in `y`. For multi-output classification problems, this attribute is a list of arrays as each output has an independent set of possible classes.

n_classes_ : *int or list of int*

Number of label for each output.

class_prior_ : *ndarray of shape (n_classes,) or list of such arrays*

Frequency of each class observed in `y`. For multioutput classification problems, this is computed independently for each output.

n_features_in_ : *int*

Number of features seen during fit.

feature_names_in_ : *ndarray of shape (`n_features_in_`,)*

Names of features seen during fit. Defined only when `X` has feature names that are all strings.

n_outputs_ : *int*

Number of outputs.

sparse_output_ : *bool*

True if the array returned from predict is to be in sparse CSC format. Is automatically set to True if the input `y` is passed in sparse format.

See also

`DummyRegressor`

Regressor that makes predictions using simple rules.

Examples

```
>>> import numpy as np
>>> from sklearn.dummy import DummyClassifier
>>> X = np.array([-1, 1, 1, 1])
>>> y = np.array([0, 1, 1, 1])
>>> dummy_clf = DummyClassifier(strategy="most_frequent")
>>> dummy_clf.fit(X, y)
DummyClassifier(strategy='most_frequent')
>>> dummy_clf.predict(X)
array([1, 1, 1, 1])
>>> dummy_clf.score(X, y)
0.75
```

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

Fit the baseline classifier.

Parameters:

X : *array-like of shape (n_samples, n_features)*

Training data.

y : *array-like of shape (n_samples,) or (n_samples, n_outputs)*

Target values.

sample_weight : *array-like of shape (n_samples,), default=None*

Sample weights.

Returns:

self : *object*

Returns the instance itself.

`get_metadata_routing()` [source]

Get metadata routing of this object.

Please check User Guide on how the routing mechanism works.

Returns:

routing : ***MetadataRequest***

A `MetadataRequest` encapsulating routing information.

`get_params(deep=True)` [source]

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.

`predict(X)` [source]

Perform classification on test vectors X.

Parameters:

X : ***array-like of shape (n_samples, n_features)***

Test data.

Returns:

y : ***array-like of shape (n_samples,) or (n_samples, n_outputs)***

Predicted target values for X.

`predict_log_proba(X)` [source]

Return log probability estimates for the test vectors X.

Parameters:

X : ***{array-like, object with finite length or shape}***

Training data.

Returns:

P : ***ndarray of shape (n_samples, n_classes) or list of such arrays***

Returns the log probability of the sample for each class in the model, where classes are ordered arithmetically for each output.

`predict_proba(X)` [source]

Return probability estimates for the test vectors X.

Parameters:

X : ***array-like of shape (n_samples, n_features)***

Test data.

Returns:

P : ***ndarray of shape (n_samples, n_classes) or list of such arrays***

Returns the probability of the sample for each class in the model, where classes are ordered arithmetically, for each output.

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

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 : ***None or array-like of shape (n_samples, n_features)***

Test samples. Passing None as test samples gives the same result as passing real test samples, since DummyClassifier operates independently of the sampled observations.

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) w.r.t. y.

`set_fit_request(*, sample_weight: bool | None | str = '$UNCHANGED$') → DummyClassifier` [source]

Request metadata passed to the `fit` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `fit` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `fit`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Note

This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters:

sample_weight : ***str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED***

Metadata routing for `sample_weight` parameter in `fit`.

Returns:

self : ***object***

The updated object.

`set_params(**params)` [source]

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as `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.

`set_score_request(*, sample_weight: bool | None | str = '$UNCHANGED$') → DummyClassifier` [source]

Request metadata passed to the `score` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `score` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `score`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Note

This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters:

sample_weight : *str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*

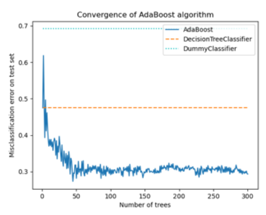
Metadata routing for `sample_weight` parameter in `score`.

Returns:

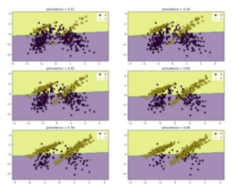
self : *object*

The updated object.

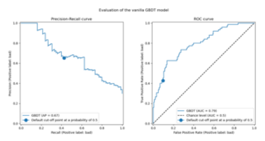
Gallery examples



Multi-class AdaBoosted Decision Trees



Class Likelihood Ratios to measure classification performance



Post-tuning the decision threshold for cost-sensitive learning

Previous
sklearn.dummy

Next
DummyRegressor