skfda.ml.classification.NearestCentroids

class **skfda.ml.classification.NearestCentroids**(*metric=<function lp_distance>*, *mean=<function mean>*) [source]

Nearest centroid classifier for functional data.

Each class is represented by its centroid, with test samples classified to the class with the nearest centroid.

Parameters:

- **metric** (*callable*, *(default)* – `lp_distance`) The metric to use when calculating distance between test samples and centroids. See the documentation of the metrics module for a list of available metrics. Defaults used L2 distance.
- **mean** (callable, (default `mean`)) – The centroids for the samples corresponding to each class is the point from which the sum of the distances (according to the metric) of all samples that belong to that particular class are minimized. By default it is used the usual mean, which minimizes the sum of L2 distance. This parameter allows change the centroid constructor. The function must accept a `FData` with the samples of one class and return a `FData` object with only one sample representing the centroid.

centroids_

`FDataGrid` – FDatagrid containing the centroid of each class

Examples

Firstly, we will create a toy dataset with 2 classes

```
>>> from skfda.datasets import make_sinusoidal_process
>>> fd1 = make_sinusoidal_process(phase_std=.25, random_state=0)
>>> fd2 = make_sinusoidal_process(phase_mean=1.8, error_std=0.,
...                               phase_std=.25, random_state=0)
>>> fd = fd1.concatenate(fd2)
>>> y = 15*[0] + 15*[1]
```

We will fit a Nearest centroids classifier

```
>>> from skfda.ml.classification import NearestCentroids
>>> neigh = NearestCentroids()
>>> neigh.fit(fd, y)
NearestCentroids(...)
```

We can predict the class of new samples

```
>>> neigh.predict(fd[::2]) # Predict labels for even samples
array([0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1])
```

See also

`KNeighborsClassifier`, `RadiusNeighborsClassifier`, `KNeighborsScalarRegressor`, `RadiusNeighborsScalarRegressor`, `NearestNeighbors`

`__init__`(*metric=<function lp_distance>, mean=<function mean>*) [source]

Initialize the classifier.

Methods

`__init__`([metric, mean])	Initialize the classifier.
`fit`(X, y)	Fit the model using X as training data and y as target values.
`get_params`([deep])	Get parameters for this estimator.
`predict`(X)	Predict the class labels for the provided data.
`score`(X, y[, sample_weight])	Returns the mean accuracy on the given test data and labels.
`set_params`(**params)	Set the parameters of this estimator.