skfda.ml.classification.KNeighborsClassifier

class `skfda.ml.classification.KNeighborsClassifier`*(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, metric=<function lp_distance>, metric_params=None, n_jobs=1, sklearn_metric=False)* [source]

Classifier implementing the k-nearest neighbors vote.

Parameters:

- **n_neighbors** (*int, optional (default = 5)*) – Number of neighbors to use by default for `kneighbors()` queries.
- **weights** (*str or callable, optional (default = 'uniform')*) –
  weight function used in prediction. Possible values:

  - 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
  - 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
  - [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

- **algorithm** (*{'auto', 'ball_tree', 'brute'}, optional*) –
  Algorithm used to compute the nearest neighbors:

  - 'ball_tree' will use `sklearn.neighbors.BallTree`.
  - 'brute' will use a brute-force search.
  - 'auto' will attempt to decide the most appropriate algorithm based on the values passed to `fit()` method.

- **leaf_size** (*int, optional (default = 30)*) – Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.
- **metric** (*string or callable, (default)* – `lp_distance`) the distance metric to use for the tree. The default metric is the Lp distance. See the documentation of the metrics module for a list of available metrics.
- **metric_params** (*dict, optional (default = None)*) – Additional keyword arguments for the metric function.
- **n_jobs** (*int or None, optional (default=None)*) – The number of parallel jobs to run for neighbors search. `None` means 1 unless in a `joblib.parallel_backend` context. `-1` means using all processors. Doesn't affect `fit()` method.
- **sklearn_metric** (*boolean, optional (default = False)*) – Indicates if the metric used is a sklearn distance between vectors (see `sklearn.neighbors.DistanceMetric`) or a functional metric of the module `skfda.misc.metrics`.

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 K-Nearest Neighbors classifier

```
>>> from skfda.ml.classification import KNeighborsClassifier
>>> neigh = KNeighborsClassifier()
>>> neigh.fit(fd, y)
KNeighborsClassifier(algorithm='auto', leaf_size=30,...)
```

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])
```

And the estimated probabilities.

```
>>> neigh.predict_proba(fd[0]) # Probabilities of sample 0
array([[ 1.,  0.]])
```

See also

`RadiusNeighborsClassifier`, `KNeighborsScalarRegressor`, `RadiusNeighborsScalarRegressor`, `NearestNeighbors`, `NearestCentroids`

Notes

See Nearest Neighbors in the sklearn online documentation for a discussion of the choice of `algorithm` and `leaf_size`.

This class wraps the sklearn classifier *sklearn.neighbors.KNeighborsClassifier*.

Warning

Regarding the Nearest Neighbors algorithms, if it is found that two neighbors, neighbor *k+1* and *k*, have identical distances but different labels, the results will depend on the ordering of the training data.

https://en.wikipedia.org/wiki/K-nearest_neighbor_algorithm

`__init__`*(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, metric=<function lp_distance>, metric_params=None, n_jobs=1, sklearn_metric=False)* [source]

Initialize the classifier.

Methods

`__init__`([n_neighbors, weights, algorithm, ...])	Initialize the classifier.
`fit`(X, y)	Fit the model using X as training data and y a
`get_params`([deep])	Get parameters for this estimator.
`kneighbors`([X, n_neighbors, return_distance])	Finds the K-neighbors of a point.
`kneighbors_graph`([X, n_neighbors, mode])	Computes the (weighted) graph of k-Neighbo
`predict`(X)	Predict the class labels for the provided data.
`predict_proba`(X)	Return probability estimates for the test data
`score`(X, y[, sample_weight])	Returns the mean accuracy on the given test
`set_params`(**params)	Set the parameters of this estimator.