skfda.ml.classification.NearestNeighbors

class `skfda.ml.classification.NearestNeighbors`(*n_neighbors=5*, *radius=1.0*, *algorithm='auto'*, *leaf_size=30*, *metric=<function lp_distance>*, *metric_params=None*, *n_jobs=1*, *sklearn_metric=False*) [source]

Unsupervised learner for implementing neighbor searches.

Parameters:

- **n_neighbors** (*int*, *optional (default = 5)*) – Number of neighbors to use by default for `kneighbors()` queries.
- **radius** (*float*, *optional (default = 1.0)*) – Range of parameter space to use by default for `radius_neighbors()` queries.
- **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)
```

We will fit a Nearest Neighbors estimator

```
>>> from skfda.ml.classification import NearestNeighbors
>>> neigh = NearestNeighbors(radius=.3)
>>> neigh.fit(fd)
NearestNeighbors(algorithm='auto', leaf_size=30,...)
```

Now we can query the k-nearest neighbors.

```
>>> distances, index = neigh.kneighbors(fd[:2])
>>> index # Index of k-neighbors of samples 0 and 1
array([[ 0,  7,  6, 11,  2],...)
```

```
>>> distances.round(2) # Distances to k-neighbors
array([[ 0.  ,  0.28,  0.29,  0.29,  0.3 ],
       [ 0.  ,  0.27,  0.28,  0.29,  0.3 ]])
```

We can query the neighbors in a given radius too.

```
>>> distances, index = neigh.radius_neighbors(fd[:2])
>>> index[0]
array([ 0,  2,  6,  7, 11]...)
```

```
>>> distances[0].round(2) # Distances to neighbors of the sample 0
array([ 0.  ,  0.3 ,  0.29,  0.28,  0.29])
```

See also

`KNeighborsClassifier`, `RadiusNeighborsClassifier`, `KNeighborsScalarRegressor`, `RadiusNeighborsScalarRegressor`, `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*.

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

`__init__`(*n_neighbors=5*, *radius=1.0*, *algorithm='auto'*, *leaf_size=30*, *metric=<function lp_distance>*, *metric_params=None*, *n_jobs=1*, *sklearn_metric=False*) [source]

Initialize the nearest neighbors searcher.

Methods

`__init__`([n_neighbors, radius, algorithm, ...])	Initialize the nearest neighbors searcher.
`fit`(X[, y])	Fit the model using X as training data.
`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-Neighbor
`radius_neighbors`([X, radius, return_distance])	Finds the neighbors within a given radius of a
`radius_neighbors_graph`([X, radius, mode])	Computes the (weighted) graph of Neighbors
`set_params`(**params)	Set the parameters of this estimator.