sklearn.neighbors.NearestNeighbors

class sklearn.neighbors.**NearestNeighbors**(*, *n_neighbors=5*, *radius=1.0*, *algorithm='auto'*, *leaf_size=30*, *metric='minkowski'*, *p=2*, *metric_params=None*, *n_jobs=None*) [source]

Unsupervised learner for implementing neighbor searches.

Read more in the User Guide.

New in version 0.9.

Parameters:

n_neighbors : *int, default=5*
Number of neighbors to use by default for `kneighbors` queries.

radius : *float, default=1.0*
Range of parameter space to use by default for `radius_neighbors` queries.

algorithm : *{'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'*
Algorithm used to compute the nearest neighbors:

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

Note: fitting on sparse input will override the setting of this parameter, using brute force.

leaf_size : *int, 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 : *str or callable, default='minkowski'*
the distance metric to use for the tree. The default metric is minkowski, and with p=2 is equivalent to the standard Euclidean metric. See the documentation of `DistanceMetric` for a list of available metrics. If metric is "precomputed", X is assumed to be a distance matrix and must be square during fit. X may be a sparse graph, in which case only "nonzero" elements may be considered neighbors.

p : *int, default=2*
Parameter for the Minkowski metric from sklearn.metrics.pairwise.pairwise_distances. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.

metric_params : *dict, default=None*
Additional keyword arguments for the metric function.

n_jobs : *int, 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. See Glossary for more details.

Attributes:

effective_metric_ : *str*
Metric used to compute distances to neighbors.

effective_metric_params_ : *dict*
Parameters for the metric used to compute distances to neighbors.

n_samples_fit_ : *int*
Number of samples in the fitted data.

See also:
`KNeighborsClassifier`
`RadiusNeighborsClassifier`
`KNeighborsRegressor`
`borsRegressor`

Toggle Menu

BallTree

Notes

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

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

Examples

```
>>> import numpy as np
>>> from sklearn.neighbors import NearestNeighbors
>>> samples = [[0, 0, 2], [1, 0, 0], [0, 0, 1]]
```

```
>>> neigh = NearestNeighbors(n_neighbors=2, radius=0.4)
>>> neigh.fit(samples)
NearestNeighbors(...)
```

```
>>> neigh.kneighbors([[0, 0, 1.3]], 2, return_distance=False)
array([[2, 0]]...)
```

```
>>> nbrs = neigh.radius_neighbors(
...     [[0, 0, 1.3]], 0.4, return_distance=False
... )
>>> np.asarray(nbrs[0][0])
array(2)
```

Methods

fit(X[, y])	Fit the nearest neighbors estimator from the training dataset.
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-Neighbors for points in X
radius_neighbors([X, radius, ...])	Finds the neighbors within a given radius of a point or points.
radius_neighbors_graph([X, radius, mode, ...])	Computes the (weighted) graph of Neighbors for points in X
set_params(**params)	Set the parameters of this estimator.

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

Fit the nearest neighbors estimator from the training dataset.

Parameters:

X : *{array-like, sparse matrix} of shape (n_samples, n_features) or (n_samples, n_samples) if metric='precomputed'*
Training data.

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

Returns:

self : *NearestNeighbors*
The fitted nearest neighbors estimator.

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.

Toggle Menu

kneighbors(*X=None*, *n_neighbors=None*, *return_distance=True*) [source]

Finds the K-neighbors of a point.

Returns indices of and distances to the neighbors of each point.

Parameters:

X : *array-like, shape (n_queries, n_features), or (n_queries, n_indexed) if metric == 'precomputed', default=None*
The query point or points. If not provided, neighbors of each indexed point are returned. In this case, the query point is not considered its own neighbor.

n_neighbors : *int, default=None*
Number of neighbors required for each sample. The default is the value passed to the constructor.

return_distance : *bool, default=True*
Whether or not to return the distances.

Returns:

neigh_dist : *ndarray of shape (n_queries, n_neighbors)*
Array representing the lengths to points, only present if return_distance=True

neigh_ind : *ndarray of shape (n_queries, n_neighbors)*
Indices of the nearest points in the population matrix.

Examples

In the following example, we construct a NearestNeighbors class from an array representing our data set and ask who's the closest point to [1,1,1]

```
>>> samples = [[0., 0., 0.], [0., .5, 0.], [1., 1., .5]]
>>> from sklearn.neighbors import NearestNeighbors
>>> neigh = NearestNeighbors(n_neighbors=1)
>>> neigh.fit(samples)
NearestNeighbors(n_neighbors=1)
>>> print(neigh.kneighbors([[1., 1., 1.]]))
(array([[0.5]]), array([[2]]))
```

As you can see, it returns [[0.5]], and [[2]], which means that the element is at distance 0.5 and is the third element of samples (indexes start at 0). You can also query for multiple points:

```
>>> X = [[0., 1., 0.], [1., 0., 1.]]
>>> neigh.kneighbors(X, return_distance=False)
array([[1],
       [2]]...)
```

kneighbors_graph(*X=None*, *n_neighbors=None*, *mode='connectivity'*) [source]

Computes the (weighted) graph of k-Neighbors for points in X

Parameters:

X : *array-like of shape (n_queries, n_features), or (n_queries, n_indexed) if metric == 'precomputed', default=None*
The query point or points. If not provided, neighbors of each indexed point are returned. In this case, the query point is not considered its own neighbor. For `metric='precomputed'` the shape should be (n_queries, n_indexed). Otherwise the shape should be (n_queries, n_features).

n_neighbors : *int, default=None*
Number of neighbors for each sample. The default is the value passed to the constructor.

mode : *{'connectivity', 'distance'}, default='connectivity'*
Type of returned matrix: 'connectivity' will return the connectivity matrix with ones and zeros, in 'distance' the edges are Euclidean distance between points.

Returns:

A : *sparse-matrix of shape (n_queries, n_samples_fit)*
`n_samples_fit` is the number of samples in the fitted data `A[i, j]` is assigned the weight of edge that connects `i` to `j`. The matrix is of CSR format.

Toggle Menu

See also:

NearestNeighbors.radius_neighbors_graph

Examples

```
>>> X = [[0], [3], [1]]
>>> from sklearn.neighbors import NearestNeighbors
>>> neigh = NearestNeighbors(n_neighbors=2)
>>> neigh.fit(X)
NearestNeighbors(n_neighbors=2)
>>> A = neigh.kneighbors_graph(X)
>>> A.toarray()
array([[1., 0., 1.],
       [0., 1., 1.],
       [1., 0., 1.]])
```

radius_neighbors(*X=None*, *radius=None*, *return_distance=True*, *sort_results=False*) [source]

Finds the neighbors within a given radius of a point or points.

Return the indices and distances of each point from the dataset lying in a ball with size `radius` around the points of the query array. Points lying on the boundary are included in the results.

The result points are *not* necessarily sorted by distance to their query point.

Parameters:

X : *array-like of (n_samples, n_features), default=None*
The query point or points. If not provided, neighbors of each indexed point are returned. In this case, the query point is not considered its own neighbor.

radius : *float, default=None*
Limiting distance of neighbors to return. The default is the value passed to the constructor.

return_distance : *bool, default=True*
Whether or not to return the distances.

sort_results : *bool, default=False*
If True, the distances and indices will be sorted by increasing distances before being returned. If False, the results may not be sorted. If `return_distance=False`, setting `sort_results=True` will result in an error.

New in version 0.22.

Returns:

neigh_dist : *ndarray of shape (n_samples,) of arrays*
Array representing the distances to each point, only present if `return_distance=True`. The distance values are computed according to the `metric` constructor parameter.

neigh_ind : *ndarray of shape (n_samples,) of arrays*
An array of arrays of indices of the approximate nearest points from the population matrix that lie within a ball of size `radius` around the query points.

Notes

Because the number of neighbors of each point is not necessarily equal, the results for multiple query points cannot be fit in a standard data array. For efficiency, `radius_neighbors` returns arrays of objects, where each object is a 1D array of indices or distances.

Examples

In the following example, we construct a NeighborsClassifier class from an array representing our data set and ask who’s the closest point to [1, 1, 1]:

Toggle Menu

```
>>> import numpy as np
>>> samples = [[0., 0., 0.], [0., .5, 0.], [1., 1., .5]]
>>> from sklearn.neighbors import NearestNeighbors
>>> neigh = NearestNeighbors(radius=1.6)
>>> neigh.fit(samples)
NearestNeighbors(radius=1.6)
>>> rng = neigh.radius_neighbors([[1., 1., 1.]])
>>> print(np.asarray(rng[0][0]))
[1.5 0.5]
>>> print(np.asarray(rng[1][0]))
[1 2]
```

The first array returned contains the distances to all points which are closer than 1.6, while the second array returned contains their indices. In general, multiple points can be queried at the same time.

radius_neighbors_graph(*X=None*, *radius=None*, *mode='connectivity'*, *sort_results=False*) [source]

Computes the (weighted) graph of Neighbors for points in X

Neighborhoods are restricted the points at a distance lower than radius.

Parameters:

X : *array-like of shape (n_samples, n_features), default=None*
The query point or points. If not provided, neighbors of each indexed point are returned. In this case, the query point is not considered its own neighbor.

radius : *float, default=None*
Radius of neighborhoods. The default is the value passed to the constructor.

mode : *{'connectivity', 'distance'}, default='connectivity'*
Type of returned matrix: 'connectivity' will return the connectivity matrix with ones and zeros, in 'distance' the edges are Euclidean distance between points.

sort_results : *bool, default=False*
If True, in each row of the result, the non-zero entries will be sorted by increasing distances. If False, the non-zero entries may not be sorted. Only used with mode='distance'.

New in version 0.22.

Returns:

A : *sparse-matrix of shape (n_queries, n_samples_fit)*
`n_samples_fit` is the number of samples in the fitted data `A[i, j]` is assigned the weight of edge that connects `i` to `j`. The matrix if of format CSR.

See also:

kneighbors_graph

Examples

```
>>> X = [[0], [3], [1]]
>>> from sklearn.neighbors import NearestNeighbors
>>> neigh = NearestNeighbors(radius=1.5)
>>> neigh.fit(X)
NearestNeighbors(radius=1.5)
>>> A = neigh.radius_neighbors_graph(X)
>>> A.toarray()
array([[1., 0., 1.],
       [0., 1., 0.],
       [1., 0., 1.]])
```

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:

Toggle Menu

****params : *dict***

Estimator parameters.

Returns:

self : *estimator instance*

Estimator instance.