# sklearn.neighbors.RadiusNeighborsClassifier

» *class* `sklearn.neighbors.`**`RadiusNeighborsClassifier`**(*radius=1.0*, *weights='uniform'*, *algorithm='auto'*, *leaf_size=30*, *p=2*, *metric='minkowski'*, *outlier_label=None*, *metric_params=None*, ***kwargs*) [source]

Classifier implementing a vote among neighbors within a given radius

Read more in the User Guide.

**Parameters:** **radius** : float, optional (default = 1.0)

Range of parameter space to use by default for `radius_neighbors` queries.

**weights** : str or callable

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.

Uniform weights are used by default.

**algorithm** : {'auto', 'ball_tree', 'kd_tree', 'brute'}, optional

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, 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.

**p** : integer, optional (default = 2)

> Power parameter for the Minkowski metric. 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** : string 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 the DistanceMetric class for a list of available metrics.

**outlier_label** : int, optional (default = None)

> Label, which is given for outlier samples (samples with no neighbors on given radius). If set to None, ValueError is raised, when outlier is detected.

**metric_params** : dict, optional (default = None)

> Additional keyword arguments for the metric function.

**See also:** `KNeighborsClassifier`, `RadiusNeighborsRegressor`, `KNeighborsRegressor`, `NearestNeighbors`

### 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

```
>>> X = [[0], [1], [2], [3]]
>>> y = [0, 0, 1, 1]
>>> from sklearn.neighbors import RadiusNeighborsClassifier
>>> neigh = RadiusNeighborsClassifier(radius=1.0)
>>> neigh.fit(X, y)
RadiusNeighborsClassifier(...)
>>> print(neigh.predict([[1.5]]))
[0]
```

### Methods

| | |
|---|---|
| `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 |
| `radius_neighbors`([X, radius, return_distance]) | 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 |
| `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. |

**`__init__`**(*radius=1.0*, *weights='uniform'*, *algorithm='auto'*, *leaf_size=30*, *p=2*, *metric='minkowski'*, *outlier_label=None*, *metric_params=None*, ***kwargs*) [source]

**`fit`**(*X*, *y*) [source]

»

Fit the model using X as training data and y as target values

**Parameters:** **X** : {array-like, sparse matrix, BallTree, KDTree}

Training data. If array or matrix, shape [n_samples, n_features], or [n_samples, n_samples] if metric='precomputed'.

**y** : {array-like, sparse matrix}

Target values of shape = [n_samples] or [n_samples, n_outputs]

**`get_params`**(*deep=True*) [source]

Get parameters for this estimator.

**Parameters:** **deep** : boolean, optional

If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns:** **params** : mapping of string to any

Parameter names mapped to their values.

**`predict`**(*X*) [source]

Predict the class labels for the provided data

**Parameters:** **X** : array-like, shape (n_query, n_features), or (n_query, n_indexed) if metric == 'precomputed'

Test samples.

**Returns:** **y** : array of shape [n_samples] or [n_samples, n_outputs]

Class labels for each data sample.

**`radius_neighbors`**(*X=None*, *radius=None*, *return_distance=True*) [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, (n_samples, n_features), optional

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

Limiting distance of neighbors to return. (default is the value passed to the constructor).

**return_distance** : boolean, optional. Defaults to True.

If False, distances will not be returned

**Returns:** **dist** : array, 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.

**ind** : array, 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]:

```
>>> 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(algorithm='auto', leaf_size=30, ...)
>>> 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'*) [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, shape = [n_samples, n_features], optional

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

Radius of neighborhoods. (default is the value passed to the constructor).

**mode** : {'connectivity', 'distance'}, optional

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 in CSR format, shape = [n_samples, n_samples]

A[i, j] is assigned the weight of edge that connects i to j.

**See also:** kneighbors_graph

**Examples**

```
>>> X = [[0], [3], [1]]
>>> from sklearn.neighbors import NearestNeighbors
>>> neigh = NearestNeighbors(radius=1.5)
>>> neigh.fit(X)
NearestNeighbors(algorithm='auto', leaf_size=30, ...)
>>> A = neigh.radius_neighbors_graph(X)
>>> A.toarray()
array([[ 1.,  0.,  1.],
       [ 0.,  1.,  0.],
       [ 1.,  0.,  1.]])
```

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

Returns 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** : array-like, shape = (n_samples, n_features)<br>Test samples.<br>**y** : array-like, shape = (n_samples) or (n_samples, n_outputs)<br>True labels for X.<br>**sample_weight** : array-like, shape = [n_samples], optional<br>Sample weights. |
| **Returns:** | **score** : float<br>Mean accuracy of self.predict(X) wrt. y. |

»

`set_params`(***params*) [source]

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

| | |
|---|---|
| **Returns:** | **self** : |