sklearn.neighbors.KNeighborsRegressor

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

Regression based on k-nearest neighbors.

The target is predicted by local interpolation of the targets associated of the nearest neighbors in the training set.

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.

weights : *{'uniform', 'distance'} or callable, 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.

Uniform weights are used by default.

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.

p : *int, 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 : *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.

metric_params : *dict, default=None*
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. Doesn't affect `fit` method.

Attributes:

effective_metric_ : *str or callable*

The distance metric to use. It will be same as the `metric` parameter or a synonym of it, e.g. 'euclidean' if the `metric` parameter set to 'minkowski' and `p` parameter set to 2.

effective_metric_params_ : *dict*

Additional keyword arguments for the metric function. For most metrics will be same with `metric_params` parameter, but may also contain the `p` parameter value if the `effective_metric_` attribute is set to 'minkowski'.

n_features_in_ : *int*

Number of features seen during fit.

New in version 0.24.

feature_names_in_ : *ndarray of shape (`n_features_in_`,)*

Names of features seen during fit. Defined only when `X` has feature names that are all strings.

New in version 1.0.

n_samples_fit_ : *int*

Number of samples in the fitted data.

See also:

`NearestNeighbors`
Unsupervised learner for implementing neighbor searches.

`RadiusNeighborsRegressor`
Regression based on neighbors within a fixed radius.

`KNeighborsClassifier`
Classifier implementing the k-nearest neighbors vote.

`RadiusNeighborsClassifier`
Classifier implementing a vote among neighbors within a given radius.

Notes

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

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_neighbors_algorithm

Examples

Hide prompts and outputs

```
>>> X = [[0], [1], [2], [3]]
>>> y = [0, 0, 1, 1]
>>> from sklearn.neighbors import KNeighborsRegressor
>>> neigh = KNeighborsRegressor(n_neighbors=2)
>>> neigh.fit(X, y)
KNeighborsRegressor(...)
>>> print(neigh.predict([[1.5]]))
[0.5]
```

Methods

`fit`(X, y)	Fit the k-nearest neighbors regressor from the training dataset.
`get_params`([deep])	Get parameters for this estimator.
`kneighbors`([X, n_neighbors, return_distance])	Find the K-neighbors of a point.
`kneighbors_graph`([X, n_neighbors, mode])	Compute the (weighted) graph of k-Neighbors for points in X.
`predict`(X)	Predict the target for the provided data.
`score`(X, y[, sample_weight])	Return the coefficient of determination of the prediction.
`set_params`(**params)	Set the parameters of this estimator.

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

Fit the k-nearest neighbors regressor 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 : *{array-like, sparse matrix} of shape (n_samples,) or (n_samples, n_outputs)*
Target values.

Returns:

self : *KNeighborsRegressor*
The fitted k-nearest neighbors regressor.

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.

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

Find 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]

Compute 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 distances between points, type of distance depends on the selected metric parameter in NearestNeighbors class.

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]` gives the weight of the edge connecting `i` to `j`. The matrix is of CSR format.

See also:

ghbors.radius_neighbors_graph

Compute the (weighted) graph of Neighbors for points in X.

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.]])
```

predict(*X*) [source]

Predict the target for the provided data.

Parameters:

X : *array-like of shape (n_queries, n_features), or (n_queries, n_indexed) if metric == 'precomputed'*
Test samples.

Returns:

y : *ndarray of shape (n_queries,) or (n_queries, n_outputs), dtype=int*
Target values.

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

Return the coefficient of determination of the prediction.

The coefficient of determination R^2 is defined as $(1 - \frac{u}{v})$, where u is the residual sum of squares `((y_true - y_pred)** 2).sum()` and v is the total sum of squares `((y_true - y_true.mean()) ** 2).sum()`. The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of `y`, disregarding the input features, would get a R^2 score of 0.0.

Parameters:

X : *array-like of shape (n_samples, n_features)*
Test samples. For some estimators this may be a precomputed kernel matrix or a list of generic objects instead with shape `(n_samples, n_samples_fitted)`, where `n_samples_fitted` is the number of samples used in the fitting for the estimator.

y : *array-like of shape (n_samples,) or (n_samples, n_outputs)*
True values for `X`.

sample_weight : *array-like of shape (n_samples,), default=None*
Sample weights.

Returns:

score : *float*
R^2 of `self.predict(X)` wrt. `y`.

Notes

The R^2 score used when calling `score` on a regressor uses `multioutput='uniform_average'` from version 0.23 to keep consistent with default value of `r2_score`. This influences the `score` method of all the multioutput regressors (except for `MultiOutputRegressor`).

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:

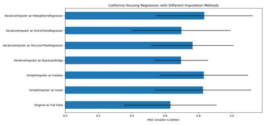
****params : *dict***
Estimator parameters.

Returns:

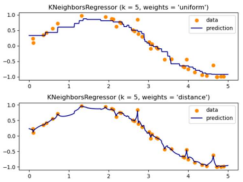
self : *estimator instance*
Estimator instance.

Examples using `sklearn.neighbors.KNeighborsRegressor`

Face completion with a multi-output estimators



Imputing missing values with variants of IterativeImputer



Nearest Neighbors regression