

sklearn.neighbors.LocalOutlierFactor

```
class sklearn.neighbors.LocalOutlierFactor(n_neighbors=20, *, algorithm='auto', leaf_size=30, metric='minkowski', p=2, metric_params=None,
contamination='auto', novelty=False, n_jobs=None)
```

[source]

Unsupervised Outlier Detection using the Local Outlier Factor (LOF).

The anomaly score of each sample is called the Local Outlier Factor. It measures the local deviation of the density of a given sample with respect to its neighbors. It is local in that the anomaly score depends on how isolated the object is with respect to the surrounding neighborhood. More precisely, locality is given by k-nearest neighbors, whose distance is used to estimate the local density. By comparing the local density of a sample to the local densities of its neighbors, one can identify samples that have a substantially lower density than their neighbors. These are considered outliers.

New in version 0.19.

Parameters:

n_neighbors : int, default=20
Number of neighbors to use by default for [kneighbors](#) queries. If n_neighbors is larger than the number of samples provided, all samples will be used.

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 is 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'
Metric to use for distance computation. Default is "minkowski", which results in the standard Euclidean distance when p = 2. See the documentation of [scipy.spatial.distance](#) and the metrics listed in [distance_metrics](#) for valid metric values.

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.

If metric is a callable function, it takes two arrays representing 1D vectors as inputs and must return one value indicating the distance between those vectors. This works for Scipy's metrics, but is less efficient than passing the metric name as a string.

p : float, default=2
Parameter for the Minkowski metric from [sklearn.metrics.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.

contamination : 'auto' or float, default='auto'
The amount of contamination of the data set, i.e. the proportion of outliers in the data set. When fitting this is used to define the threshold on the scores of the samples.

- if 'auto', the threshold is determined as in the original paper,
- if a float, the contamination should be in the range (0, 0.5].

Changed in version 0.22: The default value of `contamination` changed from 0.1 to 'auto'.

novelty : bool, default=False
By default, LocalOutlierFactor is only meant to be used for outlier detection (novelty=False). Set novelty to True if you want to use LocalOutlierFactor for novelty detection. In this case be aware that you should only use predict, decision_function and score_samples on new unseen data and not on the training set; and note that the results obtained this way may differ from the standard LOF results.

New in version 0.20.

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:

negative_outlier_factor_ : ndarray of shape (n_samples,)
The opposite LOF of the training samples. The higher, the more normal. Inliers tend to have a LOF score close to 1 (negative_outlier_factor_ close to -1), while outliers tend to have a larger LOF score.

The local outlier factor (LOF) of a sample captures its supposed ‘degree of abnormality’. It is the average of the ratio of the local reachability density of a sample and those of its k-nearest neighbors.

n_neighbors_ : int

The actual number of neighbors used for [kneighbors](#) queries.

offset_ : float

Offset used to obtain binary labels from the raw scores. Observations having a negative_outlier_factor smaller than offset_ are detected as abnormal. The offset is set to -1.5 (inliers score around -1), except when a contamination parameter different than “auto” is provided. In that case, the offset is defined in such a way we obtain the expected number of outliers in training.

New in version 0.20.

effective_metric_ : str

The effective metric used for the distance computation.

effective_metric_params_ : dict

The effective additional keyword arguments for the metric function.

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

It is the number of samples in the fitted data.

See also:
[sklearn.svm.OneClassSVM](#)
Unsupervised Outlier Detection using Support Vector Machine.

References

[1]
Breunig, M. M., Kriegel, H. P., Ng, R. T., & Sander, J. (2000, May). LOF: identifying density-based local outliers. In ACM sigmod record.

Examples

```
>>> import numpy as np
>>> from sklearn.neighbors import LocalOutlierFactor
>>> X = [[-1.1], [0.2], [101.1], [0.3]]
>>> clf = LocalOutlierFactor(n_neighbors=2)
>>> clf.fit_predict(X)
array([ 1,  1, -1,  1])
>>> clf.negative_outlier_factor_
array([-0.9821..., -1.0370..., -73.3697..., -0.9821...])
```

Methods

decision_function (X)	Shifted opposite of the Local Outlier Factor of X.
fit (X[, y])	Fit the local outlier factor detector from the training dataset.
fit_predict (X[, y])	Fit the model to the training set X and return the labels.
get_metadata_routing ()	Get metadata routing of this object.
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 labels (1 inlier, -1 outlier) of X according to LOF.
score_samples (X)	Opposite of the Local Outlier Factor of X.
set_params (**params)	Set the parameters of this estimator.

decision_function(X)

[source]

Shifted opposite of the Local Outlier Factor of X.

Bigger is better, i.e. large values correspond to inliers.

Only available for novelty detection (when novelty is set to True). The shift offset allows a zero threshold for being an outlier. The argument X is supposed to contain *new data*: if X contains a point from training, it considers the later in its own neighborhood. Also, the samples in X are not considered in the neighborhood of any point.

Parameters:
X : {array-like, sparse matrix} of shape (n_samples, n_features)

The query sample or samples to compute the Local Outlier Factor w.r.t. the training samples.

<div><div>shifted_opposite_lof_scores : ndarray of shape (n_samples,)</div><div>The shifted opposite of the Local Outlier Factor of each input samples. The lower, the more abnormal. Negative scores represent outliers, positive scores represent inliers.</div></div>	
<div><div>fit(X, y=None)</div><div>[source]</div></div>	
<div>Fit the local outlier factor detector from the training dataset.</div>	
<div><div>Parameters:</div><div><div><div>X : {array-like, sparse matrix} of shape (n_samples, n_features) or (n_samples, n_samples) if metric='precomputed'</div><div>Training data.</div></div><div><div>y : Ignored</div><div>Not used, present for API consistency by convention.</div></div></div></div>	
<div><div>Returns:</div><div><div>self : LocalOutlierFactor</div><div>The fitted local outlier factor detector.</div></div></div>	
<div><div>fit_predict(X, y=None)</div><div>[source]</div></div>	
<div>Fit the model to the training set X and return the labels.</div>	
<div>Not available for novelty detection (when novelty is set to True). Label is 1 for an inlier and -1 for an outlier according to the LOF score and the contamination parameter.</div>	
<div><div>Parameters:</div><div><div><div>X : {array-like, sparse matrix} of shape (n_samples, n_features), default=None</div><div>The query sample or samples to compute the Local Outlier Factor w.r.t. the training samples.</div></div><div><div>y : Ignored</div><div>Not used, present for API consistency by convention.</div></div></div></div>	
<div><div>Returns:</div><div><div>is_inlier : ndarray of shape (n_samples,)</div><div>Returns -1 for anomalies/outliers and 1 for inliers.</div></div></div>	
<div><div>get_metadata_routing()</div><div>[source]</div></div>	
<div>Get metadata routing of this object.</div>	
<div>Please check User Guide on how the routing mechanism works.</div>	
<div><div>Returns:</div><div><div>routing : MetadataRequest</div><div>A MetadataRequest encapsulating routing information.</div></div></div>	
<div><div>get_params(deep=True)</div><div>[source]</div></div>	
<div>Get parameters for this estimator.</div>	
<div><div>Parameters:</div><div><div><div>deep : bool, default=True</div><div>If True, will return the parameters for this estimator and contained subobjects that are estimators.</div></div></div></div>	
<div><div>Returns:</div><div><div>params : dict</div><div>Parameter names mapped to their values.</div></div></div>	
<div><div>kneighbors(X=None, n_neighbors=None, return_distance=True)</div><div>[source]</div></div>	
<div>Find the K-neighbors of a point.</div>	
<div>Returns indices of and distances to the neighbors of each point.</div>	
<div><div>Parameters:</div><div><div><div>X : {array-like, sparse matrix}, shape (n_queries, n_features), or (n_queries, n_indexed) if metric == 'precomputed', default=None</div><div>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.</div></div><div><div>n_neighbors : int, default=None</div><div>Number of neighbors required for each sample. The default is the value passed to the constructor.</div></div><div><div>distance : bool, default=True</div></div></div></div>	

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, sparse matrix} 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:

[NearestNeighbors.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=None)

[\[source\]](#)

Predict the labels (1 inlier, -1 outlier) of X according to LOF.

Only available for novelty detection (when novelty is set to True). This method allows to generalize prediction to *new observations* (not in the training set). Note that the result of clf.fit(X) then clf.predict(X) with novelty=True may differ from the result obtained by clf.fit_predict(X) with novelty=False.

Parameters:

X : {array-like, sparse matrix} of shape (n_samples, n_features)

The query sample or samples to compute the Local Outlier Factor w.r.t. the training samples.

is_inlier : *ndarray of shape (n_samples,)*

Returns -1 for anomalies/outliers and +1 for inliers.

score_samples(X)

[\[source\]](#)

Opposite of the Local Outlier Factor of X.

It is the opposite as bigger is better, i.e. large values correspond to inliers.

Only available for novelty detection (when novelty is set to True). The argument X is supposed to contain *new data*: if X contains a point from training, it considers the later in its own neighborhood. Also, the samples in X are not considered in the neighborhood of any point. Because of this, the scores obtained via `score_samples` may differ from the standard LOF scores. The standard LOF scores for the training data is available via the `negative_outlier_factor_` attribute.

Parameters:

X : *{array-like, sparse matrix} of shape (n_samples, n_features)*

The query sample or samples to compute the Local Outlier Factor w.r.t. the training samples.

Returns:

opposite_lof_scores : *ndarray of shape (n_samples,)*

The opposite of the Local Outlier Factor of each input samples. The lower, the more abnormal.

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*

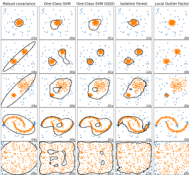
Estimator parameters.

Returns:

self : *estimator instance*

Estimator instance.

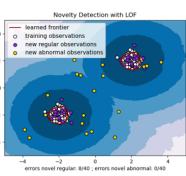
Examples using `sklearn.neighbors.LocalOutlierFactor`



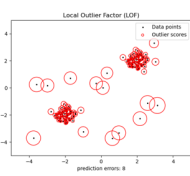
Comparing anomaly detection algorithms for outlier detection on toy datasets



Evaluation of outlier detection estimators



Novelty detection with Local Outlier Factor (LOF)



Outlier detection with Local Outlier Factor (LOF)