pyriemann.clustering.Kmeans

class pyriemann.clustering. **Kmeans** (*n_clusters=2, max_iter=100, metric='riemann', random_state=None, init='random', n_init=10, n_jobs=1, tol=0.0001*) [source]
(../_modules/pyriemann/clustering.html#Kmeans)

Kmean clustering using Riemannian geometry.

Find clusters that minimize the sum of squared distance to their centroid. This is a direct implementation of the kmean algorithm with a riemanian metric.

Parameters:

n_cluster: int (default: 2)

number of clusters.

max_iter : int (default: 100)

The maximum number of iteration to reach convergence.

metric : string (default: 'riemann')

The type of metric used for centroid and distance estimation.

random_state : integer or numpy.RandomState, optional

The generator used to initialize the centers. If an integer is given, it fixes the seed. Defaults to the global numpy random number generator.

init : 'k-means++', 'random' or an ndarray (default 'random')

Method for initialization of centers. 'k-means++' : selects initial cluster centers for k-mean clustering in a smart way to speed up convergence. See section Notes in k_init for more details. 'random': choose k observations (rows) at random from data for the initial centroids. If an ndarray is passed, it should be of shape (n_clusters, n_features) and gives the initial centers.

n_init : int, (default: 10)

Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia.

n_jobs : int, (default: 1)

> The number of jobs to use for the computation. This works by computing each of the n_init runs in parallel. If -1 all CPUs are used. If 1 is given, no parallel computing code is used at all, which is useful for debugging. For n_jobs below -1, (n_cpus + 1 + n_jobs) are used. Thus for n_jobs = -2, all CPUs but one are used.

tol: float, (default: 1e-4)

> the stopping criterion to stop convergence, representing the minimum amount of change in labels between two iterations.

See also

`Kmeans` , `MDM`

Notes

New in version 0.2.2.

Attributes

mdm	(MDM instance.) MDM instance containing the centroids.
labels :	Labels of each point
inertia	(float) Sum of distances of samples to their closest cluster center.

`__init__` (*n_clusters=2, max_iter=100, metric='riemann', random_state=None, init='random', n_init=10, n_jobs=1, tol=0.0001*) **[source]**
(../_modules/pyriemann/clustering.html#Kmeans.__init__)

Init.

`centroids` () **(../_modules/pyriemann/clustering.html#Kmeans.centroids)** **[source]**

helper for fast access to the centroid.

Returns: **centroids** : list of SPD matrices, len (n_cluster)

> Return a list containing the centroid of each cluster.

`fit` (*X, y=None*) **(../_modules/pyriemann/clustering.html#Kmeans.fit)** **[source]**

Fit (estimates) the clusters.

Parameters: **X** : ndarray, shape (n_trials, n_channels, n_channels)

ndarray of SPD matrices.

y : ndarray | None (default None)

Not used, here for compatibility with sklearn API.

Returns: **self** : Kmeans instance

The Kmean instance.

`fit_predict` (*X*, *y=None*)

Performs clustering on X and returns cluster labels.

Parameters: **X** : ndarray, shape (n_samples, n_features)

Input data.

Returns: **y** : ndarray, shape (n_samples,)

cluster labels

`fit_transform` (*X*, *y=None*, ***fit_params*)

Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters: **X** : numpy array of shape [n_samples, n_features]

Training set.

y : numpy array of shape [n_samples]

Target values.

Returns: **X_new** : numpy array of shape [n_samples, n_features_new]

Transformed array.

`get_params` (*deep=True*)

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*) (../_modules/pyriemann/clustering.html#Kmeans.predict) [source]

get the predictions.

Parameters: **X** : ndarray, shape (n_trials, n_channels, n_channels)

ndarray of SPD matrices.

Returns: **pred** : ndarray of int, shape (n_trials, 1)

the prediction for each trials according to the closest centroid.

`score` (*X*, *y*, *sample_weight=None*)

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)

Test samples.

y : array-like, shape = (n_samples) or (n_samples, n_outputs)

True labels for X.

sample_weight : array-like, shape = [n_samples], optional

Sample weights.

Returns: **score** : float

Mean accuracy of self.predict(X) wrt. y.

`set_params` (***params*)

Set the parameters of this estimator.

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

Returns: self

`transform` (*X*) (../_modules/pyriemann/clustering.html#Kmeans.transform) [source]

get the distance to each centroid.

Parameters: **X** : ndarray, shape (n_trials, n_channels, n_channels)

ndarray of SPD matrices.

Returns: **dist** : ndarray, shape (n_trials, n_cluster)

the distance to each centroid according to the metric.

Source (../_sources/generated/pyriemann.clustering.Kmeans.txt) Back to top