

fx-fit(k-means)

Link

https://github.com/scikit-learn/scikit-learn/blob/92c9b1866/sklearn/cluster/_kmeans.py#L1213

▼ Code

```
def fit(self, X, y=None, sample_weight=None):

    random_state = check_random_state(self.random_state)

    self.cluster_centers_, self.labels_, self.inertia_, s
        k_means(
            X, n_clusters=self.n_clusters, sample_weight=
            init=self.init, n_init=self.n_init,
            max_iter=self.max_iter, verbose=self.verbose,
            precompute_distances=self.precompute_distance
            tol=self.tol, random_state=random_state, copy
            n_jobs=self.n_jobs, algorithm=self.algorithm,
            return_n_iter=True)
    return self
```

▼ Documentation

```
"""Compute k-means clustering.
```

```
Parameters
```

```
-----
```

```
X : array-like or sparse matrix, shape=(n_samples, n_
    Training instances to cluster. It must be noted t
```

```

        will be converted to C ordering, which will cause
        copy if the given data is not C-contiguous.

    y : Ignored
        not used, present here for API consistency by con

    sample_weight : array-like, shape (n_samples,), optio
        The weights for each observation in X. If None, a
        are assigned equal weight (default: None)

    """

```

Prompt

As a code documentation assistant, you are programmed to document at the function level of the given code snippet. Your approach involves placing comments directly under the def statement of the function. The output should be the entire code along with the documentation of the function written. Code is as follows:

```

"""
def fit(self, X, y=None, sample_weight=None):

    random_state = check_random_state(self.random_state)

    self.cluster_centers_, self.labels_, self.inertia_, self.n_iter_
= \
    k_means(
        X, n_clusters=self.n_clusters,

```

```

sample_weight=sample_weight,
        init=self.init, n_init=self.n_init,
        max_iter=self.max_iter, verbose=self.verbose,
        precompute_distances=self.precompute_distances,
        tol=self.tol, random_state=random_state,
copy_x=self.copy_x,
        n_jobs=self.n_jobs, algorithm=self.algorithm,
        return_n_iter=True)
    return self
"""

```

Responses

ChatGPT

```

def fit(self, X, y=None, sample_weight=None):
    """
    Fit the KMeans clustering model to the given data.

    Parameters:
    -----
    X : array-like or pd.DataFrame, shape (n_samples, n_features)
        The input data to be clustered.

    y : None, default=None
        Ignored. This parameter is only kept for compatibility.

    sample_weight : array-like, shape (n_samples,), optional (default=None)
        The weights of samples. If None, all samples are given equal weight.

    Returns:
    -----

```

```

self : object
    Returns the instance itself.

Notes:
-----
This method fits the KMeans clustering model to the input data,
assigns labels to each data point, calculates the inertia (variance)
and tracks the number of iterations performed during the fitting.

References:
-----
- `scikit-learn KMeans documentation <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>`
"""
random_state = check_random_state(self.random_state)

self.cluster_centers_, self.labels_, self.inertia_, self.n_iter_ = \
    k_means(
        X, n_clusters=self.n_clusters, sample_weight=sample_weight,
        init=self.init, n_init=self.n_init,
        max_iter=self.max_iter, verbose=self.verbose,
        precompute_distances=self.precompute_distances,
        tol=self.tol, random_state=random_state, copy_x=self.copy_x,
        n_jobs=self.n_jobs, algorithm=self.algorithm,
        return_n_iter=True)
return self

```

GPT4

```

def fit(self, X, y=None, sample_weight=None):
    """
    Fits the model to the data in X.

    This function computes the centroids of the clusters (k-means)
    and assigns each data point to the cluster it is closest to.

    Parameters:

```

- X : array-like or sparse matrix, shape (n_samples, n_features)
The input samples to cluster.
- y : Ignored
Not used, present here for API consistency by convention.
- sample_weight : array-like, shape (n_samples,), optional
The weights for each observation in X. If None, all observations are given equal weight.

Attributes updated:

- cluster_centers_ : array, [n_clusters, n_features]
Coordinates of cluster centers.
- labels_ : array, [n_samples,]
Labels of each point.
- inertia_ : float
Sum of squared distances of samples to their closest cluster center.
- n_iter_ : int
Number of iterations run.

Other parameters like 'init', 'n_init', 'max_iter', 'verbose' are not updated.

Returns:

```
self : object
    Fitted estimator.
"""
```

```
random_state = check_random_state(self.random_state)
```

```
self.cluster_centers_, self.labels_, self.inertia_, self.n_iter_ = \
    k_means(
        X, n_clusters=self.n_clusters, sample_weight=sample_weight,
        init=self.init, n_init=self.n_init,
        max_iter=self.max_iter, verbose=self.verbose,
        precompute_distances=self.precompute_distances,
        tol=self.tol, random_state=random_state, copy_x=self.copy_x,
        n_jobs=self.n_jobs, algorithm=self.algorithm,
```

```

        return_n_iter=True)
    return self

```

Bard

```

"""
Fits the model to the data.

Parameters:
    X : ndarray, shape (n_samples, n_features)
        Input data to be clustered.

    y : ndarray, shape (n_samples,) or None (default)
        Target labels for the data. This parameter is ignored with
        sample_weight.

    sample_weight : ndarray, shape (n_samples,) or None (default)
        Weights for each sample. If None, all samples are given
        equal weight.

Returns:
    self : KMeans instance
        Returns the instance itself.
"""

def fit(self, X, y=None, sample_weight=None):

    random_state = check_random_state(self.random_state)

    self.cluster_centers_, self.labels_, self.inertia_, self.n_iter_ = \
        k_means(
            X, n_clusters=self.n_clusters, sample_weight=sample_weight,
            init=self.init, n_init=self.n_init,
            max_iter=self.max_iter, verbose=self.verbose,
            precompute_distances=self.precompute_distances,
            tol=self.tol, random_state=random_state, copy_x=self.copy_x)

```

```

        n_jobs=self.n_jobs, algorithm=self.algorithm,
        return_n_iter=True)
    return self

```

LLama2

```

def fit(self, X, y=None, sample_weight=None):
    """
    Fits the model to the data using the K-Means clustering algorithm.

    Parameters
    -----
    X : array-like, shape (n_samples, n_features)
        Data matrix.
    y : array-like, shape (n_samples,)
        Target vector. If None, then it will be ignored.
    sample_weight : array-like, shape (n_samples,)
        Sample weights. If None, then each sample will have equal weight.

    Returns
    -----
    self

    Notes
    -----
    This method uses the K-Means++ initialization method by default.
    The number of iterations can be controlled with the 'max_iter' parameter.
    The convergence tolerance can be set with the 'tol' parameter.
    The algorithm used can be selected from ['auto', 'full', 'elkan'].
    The distance metric used can be selected from ['euclidean', 'manhattan', 'minkowski'].
    The random state can be controlled with the 'random_state' parameter.
    The number of jobs that can be used for parallel computation can be set with the 'n_jobs' parameter.
    The inertia of the clusters can be accessed through the 'inertia_' attribute.
    The cluster centers can be accessed through the 'cluster_centers_' attribute.
    The labels of the samples can be accessed through the 'labels_' attribute.

```

```
The number of iterations taken to reach convergence can be a
"""
```

StarChat

```
def fit(self, X, y=None, sample_weight=None):
    """Performs K-Means clustering on the dataset X.

    Parameters
    -----
    X : array-like or sparse matrix, shape=(n_samples, n_features)
        Training data where each row represents a point in the feature space.

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

    sample_weight : array-like of shape [n_samples], default=None
        Individual weights for each sample. If None, then all samples are equally weighted.

    Returns
    -----
    self : object
        Fitted estimator.
    """

    # Some code here...

    # More code here...
```