**\_clusters** : int, optional, default: 8

The number of clusters to form as well as the number of centroids to generate.

**init** : {‘k-means++’, ‘random’ or an ndarray}

Method for initialization, defaults to ‘k-means++’:

‘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.

**max\_iter** : int, default: 300

Maximum number of iterations of the k-means algorithm for a single run.

**tol** : float, default: 1e-4

Relative tolerance with regards to inertia to declare convergence

**precompute\_distances** : {‘auto’, True, False}

Precompute distances (faster but takes more memory).

‘auto’ : do not precompute distances if n\_samples \* n\_clusters > 12 million. This corresponds to about 100MB overhead per job using double precision.

True : always precompute distances

False : never precompute distances

**verbose** : int, default 0

Verbosity mode.

**random\_state** : int, RandomState instance or None, optional, default: None

If int, random\_state is the seed used by the random number generator; If RandomState instance, random\_state is the random number generator; If None, the random number generator is the RandomState instance used by *np.random*.

**copy\_x** : boolean, default True

When pre-computing distances it is more numerically accurate to center the data first. If copy\_x is True, then the original data is not modified. If False, the original data is modified, and put back before the function returns, but small numerical differences may be introduced by subtracting and then adding the data mean.

**n\_jobs** : int

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.

**algorithm** : “auto”, “full” or “elkan”, default=”auto”

K-means algorithm to use. The classical EM-style algorithm is “full”. The “elkan” variation is more efficient by using the triangle inequality, but currently doesn’t support sparse data. “auto” chooses “elkan” for dense data and “full” for sparse data.