

# K-Means Clustering

Learn about the K-Means clustering algorithm and how it works.

## Chapter Goals:

- Learn about K-means clustering and how it works
- Understand why mini-batch clustering is used for large datasets

### A. K-means algorithm

The idea behind clustering data is pretty simple: partition a dataset into groups of similar data observations. How we go about finding these clusters is a bit more complex, since there are a number of different methods for clustering datasets.

The most well-known clustering method is K-means clustering ([https://en.wikipedia.org/wiki/K-means\\_clustering](https://en.wikipedia.org/wiki/K-means_clustering)). The K-means clustering algorithm will separate the data into  $K$  clusters (the number of clusters is chosen by the user) using cluster means, also known as *centroids*.

These centroids represent the "centers" of each cluster. Specifically, a cluster's centroid is equal to the average of all the data observations within the cluster.

```
1 cluster = np.array([
2     [ 1.2, 0.6],
3     [ 2.4, 0.8],
4     [-1.6, 1.4],
5     [ 0. , 1.2]])
6 print('Cluster:\n{}\n'.format(repr(cluster)))
7
8 centroid = cluster.mean(axis=0)
9 print('Centroid:\n{}\n'.format(repr(centroid)))
```



×

Output

Cluster:  
array([[ 1.2, 0.6],  
 [ 2.4, 0.8],  
 [-1.6, 1.4],  
 [ 0. , 1.2]])

Centroid:  
array([0.5, 1. ])

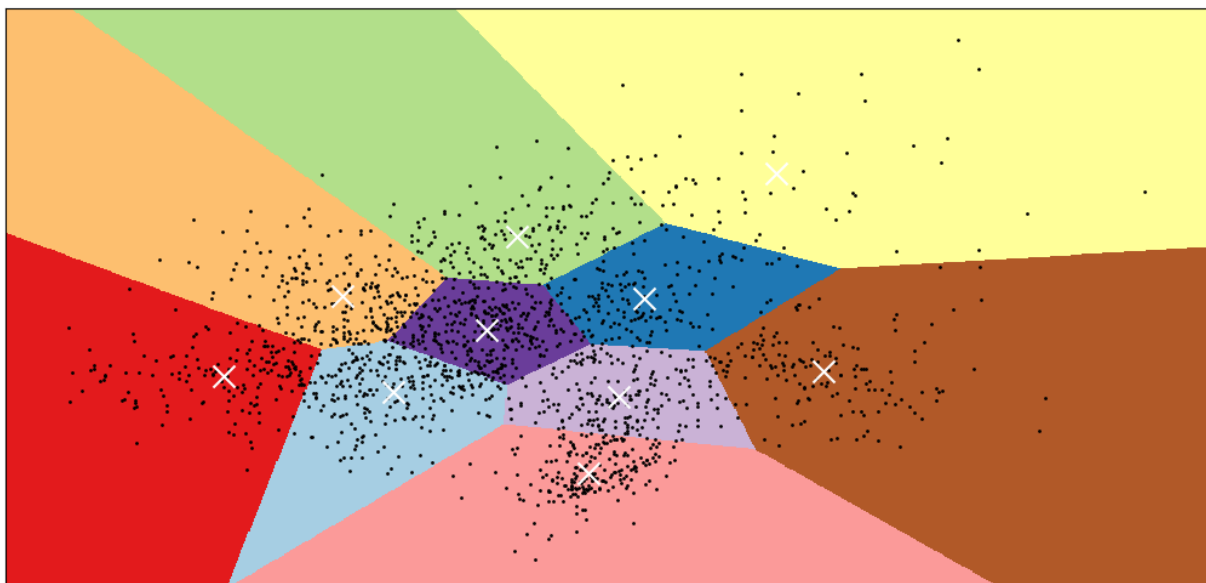
⚙️

0.903s

📄

The K-means clustering algorithm is an iterative process. Each iteration, the algorithm will assign each data observation to the cluster with the closest centroid to the observation (using the regular distance (<http://mathworld.wolfram.com/Distance.html>) metric).

Then it updates each centroid to be equal to the new average of the data observations in the cluster. Note that at the beginning of the algorithm, the cluster centroids are either randomly initialized or (better) initialized using the K-means++ (<https://en.wikipedia.org/wiki/K-means%2B%2B>) algorithm. The clustering process stops when there are no more changes in cluster assignment for any data observation.



An example of K-means clustering on a dataset with 10 clusters chosen ( $K = 10$ ). Clusters are

distinguished by color. The white crosses represent the centroids of each cluster.



In scikit-learn, K-means clustering is implemented using the `KMeans` (<https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans>) object (part of the `cluster` module).

The code below demonstrates how to use the `KMeans` object (with 3 clusters).

```
1 from sklearn.cluster import KMeans
2 kmeans = KMeans(n_clusters=3)
3 # predefined data
4 kmeans.fit(data)
5
6 # cluster assignments
7 print('{}\n'.format(repr(kmeans.labels_)))
8
9 # centroids
10 print('{}\n'.format(repr(kmeans.cluster_centers_)))
11
12 new_obs = np.array([
13     [5.1, 3.2, 1.7, 1.9],
14     [6.9, 3.2, 5.3, 2.2]])
15 # predict clusters
16 print('{}\n'.format(repr(kmeans.predict(new_obs))))
```



Output

1.999s

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2,
       2, 2, 2, 0, 0, 2, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 0, 2, 2, 2, 2,
       2, 0, 2, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 0], dtype=int32)
```

```
array([[5.9016129 , 2.7483871 , 4.39354839, 1.43387097],
```



```

14     [6.9, 3.2, 5.3, 2.2]])
15 # predict clusters
16 print('{}\n'.format(repr(kmeans.predict(new_obs))))

```



Output

1.627s

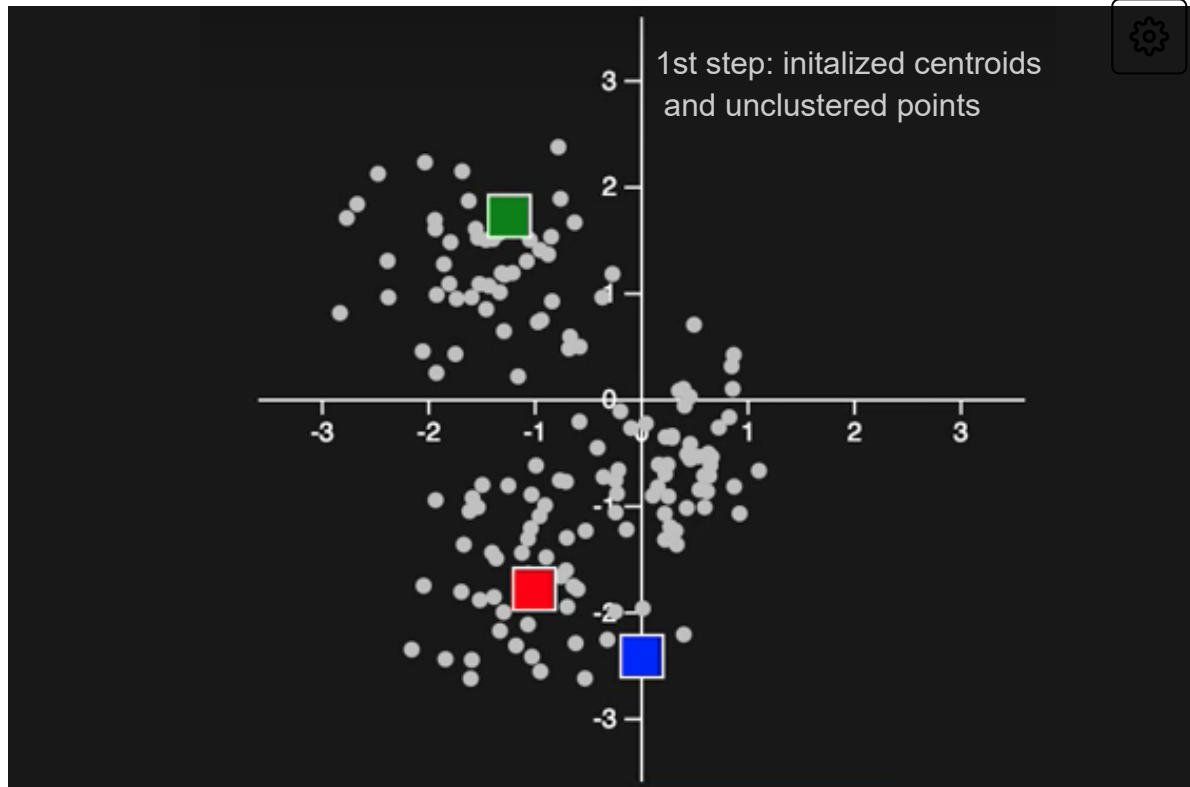
```

array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 2, 2, 2, 0, 2,
       2, 2, 2, 0, 0, 2, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 0, 2, 2,
       2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 0], dtype=int32)

array([[5.86153846, 2.74153846, 4.36923077, 1.45230769],

```

Note that the clusterings can have different permutations, i.e. different cluster labelings (0, 1, 2 vs. 1, 2, 0). Otherwise, the cluster assignments for both `kMeans` and `MiniBatchKMeans` should be relatively the same.



Example of the K-Means algorithm's multiple steps.

1 of 2

< > ▷ ⏪ + ⌂

## Time to Code!

The coding exercise for this chapter will be to complete the `kmeans_clustering` function, which will use either `KMeans` or `MiniBatchKMeans` for clustering data.

Which object we use for clustering depends on the value of `batch_size`.

**If `batch_size` is `None`, set `kmeans` equal to `KMeans`, initialized with `n_clusters` for the `n_clusters` argument. Otherwise, set `kmeans` equal to `MiniBatchKMeans`, initialized with `n_clusters` for the `n_clusters` argument and `batch_size` for the `batch_size` argument.**

After setting up `kmeans`, we fit it on the `data` and return it.

**Call `kmeans.fit` with `data` as the only argument. Then return `kmeans`.**

```
1 def kmeans_clustering(data, n_clusters, batch_size):
2     # CODE HERE
```





## Solution



```
1 def kmeans_clustering(data, n_clusters, batch_size):
2     if batch_size is None:
3         kmeans = KMeans(n_clusters=n_clusters)
4     else:
5         kmeans = MiniBatchKMeans(n_clusters=n_clusters,
6                                   batch_size=batch_size)
7     kmeans.fit(data)
8     return kmeans
9
10
```

[← Back](#)

Nearest Neighbors

[Next →](#)

Hierarchical Clustering

☒ **Mark as Completed**58% completed, meet the [criteria](#) and claim your course certificate![Buy Certificate](#)Report an  
Issue

Ask a Question

(https://discuss.educative.io/tag/k-means-clustering\_\_clustering-with-scikit-learn\_\_machine-learning-for-software-engineers)

