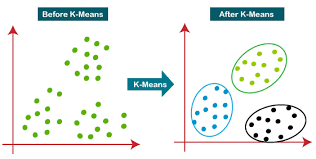
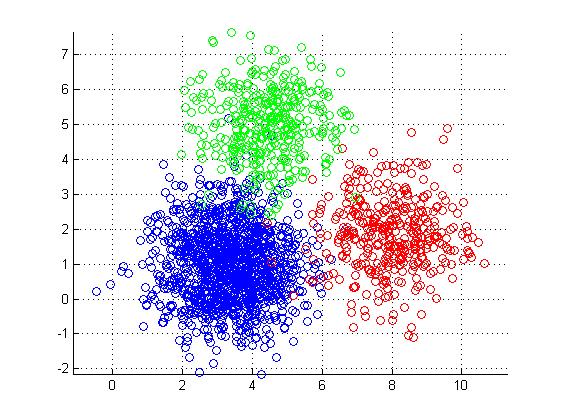
K-means clustering

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar to each other than to those in other groups (clusters). Clustering is one of the main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval.

**K-means clustering**

K-means is **a centroid-based clustering algorithm, where we calculate the distance between each data point and a centroid to assign it to a cluster**. The goal is to identify the K number of groups in the dataset.

****



What is K in K means clustering?

**The number of clusters found from data by the method** is denoted by the letter 'K' in K-means. In this method, data points are assigned to clusters in such a way that the sum of the squared distances between the data points and the centroid is as small as possible.

Why is K means clustering best?

**Guarantees convergence**. Can warm-start the positions of centroids. Easily adapts to new examples. Generalizes to clusters of different shapes and sizes, such as elliptical clusters.

**Working of K-Means Algorithm**

The following stages will help us understand how the K-Means clustering technique works-

* ***Step\_1:*** First, we need to provide the number of clusters, K, that need to be generated by this algorithm.
* ***Step\_2:*** Next, choose K data points at random and assign each to a cluster. Briefly, categorize the data based on the number of data points.
* ***Step\_3:*** The cluster centroids will now be computed.
* ***Step\_4:*** Iterate the steps below until we find the ideal centroid, which is the assigning of data points to clusters that do not vary.

1. The sum of squared distances between data points and centroids would be calculated first.
2. At this point, we need to allocate each data point to the cluster that is closest to the others (centroid).
3. Finally, compute the centroids for the clusters by averaging all of the cluster’s data points.

K-means implements the **Expectation-Maximization** strategy to solve the problem. The Expectation-step is used to assign data points to the nearest cluster, and the Maximization-step is used to compute the centroid of each cluster.

**When using the K-means algorithm, we must keep the following points in mind:**

* It is suggested to normalize the data while dealing with clustering algorithms such as K-Means since such algorithms employ distance-based measurement to identify the similarity between data points.
* Because of the iterative nature of K-Means and the random initialization of centroids, K-Means may become stuck in a local optimum and fail to converge to the global optimum. As a result, it is advised to employ distinct centroids’ initializations.

**Implementation in Python:**

we will import the essential packages.

|  |  |
| --- | --- |
| 1  2  3  4  5 | %matplotlib inline  **import** **matplotlib.pyplot** **as** **plt**  **import** **seaborn** **as** **sns**; sns.set()  **import** **numpy** **as** **np**  **from** **sklearn.cluster** **import** KMeans |

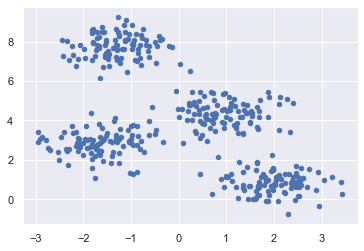
build a 2D dataset with four blobs.

|  |  |
| --- | --- |
| 1  2 | **from** **sklearn.datasets.samples\_generator** **import** make\_blobs  X, y\_true = make\_blobs(n\_samples=400, centers=4, cluster\_std=0.60, random\_state=0 |

visualizing the dataset.

|  |  |
| --- | --- |
| 1  2 | plt.scatter(X[:, 0], X[:, 1], s=20);  plt.show() |

**Out\_put:**

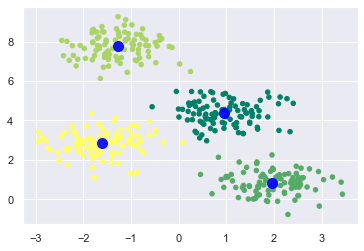
****

* create a K – means object while specifying the number of clusters, train the model, and estimate

|  |  |
| --- | --- |
| 1  2  3  4  5 | *#create a K – means object while specifying the number of clusters, train the model, and estimate*  kmeans = KMeans(n\_clusters=4)  kmeans.fit(X)  y\_kmeans = kmeans.predict(X) |

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | *# we can plot and visualize the cluster’s centers as determined by the k-means Python estimator-*  plt.scatter(X[:, 0], X[:, 1], c=y\_kmeans, s=20, cmap='summer')  centers = kmeans.cluster\_centers\_  plt.scatter(centers[:, 0], centers[:, 1], c='blue', s=100, alpha=0.9);  plt.show() |

**Out\_put:**



**K-Means Clustering Algorithm Applications**

The performance of K-means clustering is sufficient to achieve the given goals. When it comes to the following scenarios, it is useful:

* To get relevant insights from the data we’re dealing with.
* Distinct models will be created for different subgroups in a cluster-then-predict approach.
* Market segmentation
* Document Clustering
* Image segmentation
* Image compression
* Customer segmentation
* Analyzing the trend on dynamic data

**Advantages and Disadvantages**

***Advantages***: The below are some of the features of K-Means clustering algorithms:

* It is simple to grasp and put into practice.
* K-means would be faster than Hierarchical clustering if we had a high number of variables.
* An instance’s cluster can be changed when centroids are re-computation.
* When compared to Hierarchical clustering, K-means produces tighter clusters.

***Disadvantages:*** Some of the drawbacks of K-Means clustering techniques are as follows:

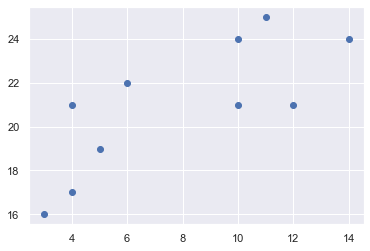
* The number of clusters, i.e., the value of k, is difficult to estimate.
* A major effect on output is exerted by initial inputs such as the number of clusters in a network (value of k).
* The sequence in which the data is entered has a considerable impact on the final output.
* It’s quite sensitive to rescaling. If we rescale our data using normalization or standards, the outcome will be drastically different. ultimate result
* It is not advisable to do clustering tasks if the clusters have a sophisticated geometric shape

**#**K-means clustering requires us to select K, the number of clusters we want to group the data into. The elbow method lets us graph the inertia (a distance-based metric) and visualize the point at which it starts decreasing linearly. This point is referred to as the "eblow" and is a good estimate for the best value for K based on our data.

**Start by visualizing some data points:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | *#Start by visualizing some data points:*  **import** **matplotlib.pyplot** **as** **plt**  x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]  y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]  plt.scatter(x, y)  plt.show() |

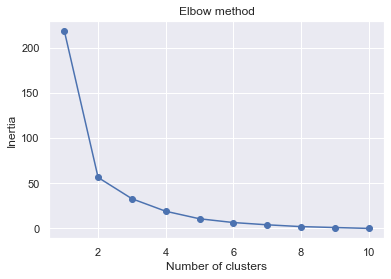
**Out\_put:**



**Now we utilize the elbow method to visualize the intertia for different values of K:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | *#Now we utilize the elbow method to visualize the intertia for different values of K:*  **from** **sklearn.cluster** **import** KMeans  data = list(zip(x, y))  inertias = []  **for** i **in** range(1,11):  kmeans = KMeans(n\_clusters=i)  kmeans.fit(data)  inertias.append(kmeans.inertia\_)  plt.plot(range(1,11), inertias, marker='o')  plt.title('Elbow method')  plt.xlabel('Number of clusters')  plt.ylabel('Inertia')  plt.show() |

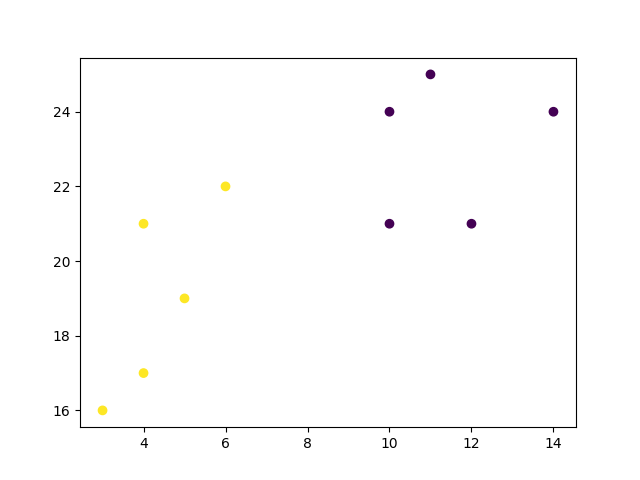
**Out\_put:**



**The elbow method shows that 2 is a good value for K, so we retrain and visualize the result:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | *#The elbow method shows that 2 is a good value for K, so we retrain and visualize the result:*  kmeans = KMeans(n\_clusters=2)  kmeans.fit(data)  plt.scatter(x, y, c=kmeans.labels\_)  plt.show() |

**Out\_put:**



{\displaystyle {\underset {\mathbf {S} }{\operatorname {arg\,min} }}\sum \_{i=1}^{k}\sum \_{\mathbf {x} \in S\_{i}}\left\|\mathbf {x} -{\boldsymbol {\mu }}\_{i}\right\|^{2}={\underset {\mathbf {S} }{\operatorname {arg\,min} }}\sum \_{i=1}^{k}|S\_{i}|\operatorname {Var} S\_{i}}