K-Means Clustering with Scikit-Learn

K-means clustering is one of the most widely used unsupervised machine learning algorithms that forms clusters of data based on the similarity between data instances. For this particular algorithm to work, the number of clusters has to be defined beforehand. The K in the K-means refers to the number of clusters.

The K-means algorithm starts by randomly choosing a centroid value for each cluster. After that the algorithm iteratively performs three steps:

(i) Find the Euclidean distance between each data instance and centroids of all the clusters;

(ii) Assign the data instances to the cluster of the centroid with nearest distance;

(iii) Calculate new centroid values based on the mean values of the coordinates of all the data instances from the corresponding cluster.

A Simple Example

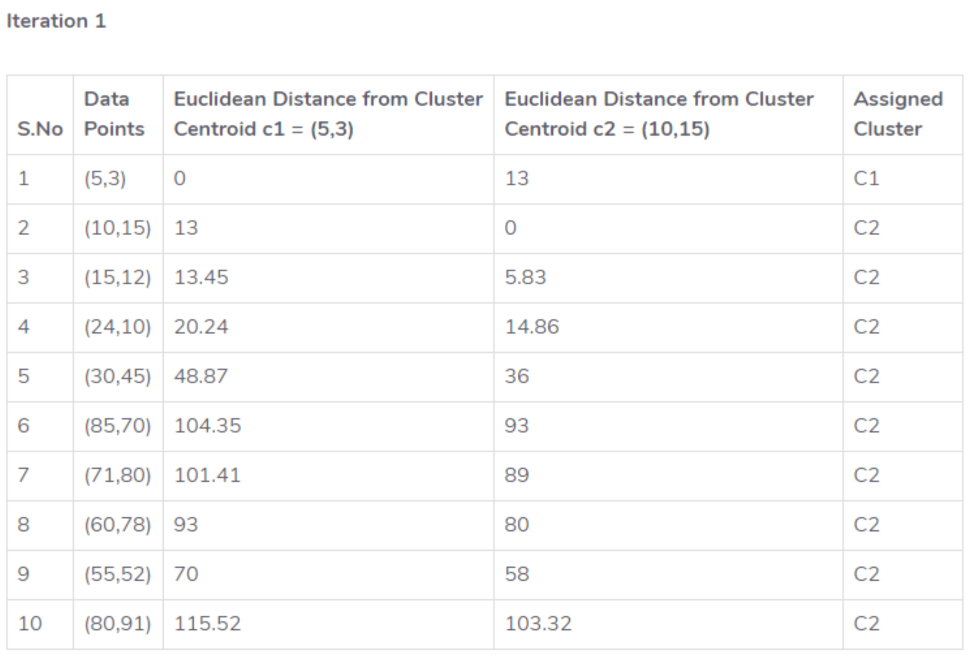
Let's try to see how the K-means algorithm works with the help of a handcrafted example, before implementing the algorithm in Scikit-Learn.

We have a set of the following two dimensional data instances named D.

In [1]:
```
D = {(5, 3), (10, 15), (15, 12), (24, 10), (30, 45),
     (85, 70), (71, 80), (60, 78), (55, 52), (80, 91)}

# We want to divide this data into two clusters,
# C1 and C2 based on the similarity between the data points.
```

The first step is to randomly initialize values for the centroids of both clusters. Let's name centroids of clusters C1 and C2 as c1 and c2 and initialize them with the values of the first two data points i.e. (5, 3) and (10, 15).



Iteration 1

S.No	Data Points	Euclidean Distance from Cluster Centroid c1 = (5,3)	Euclidean Distance from Cluster Centroid c2 = (10,15)	Assigned Cluster
1	(5,3)	0	13	C1
2	(10,15)	13	0	C2
3	(15,12)	13.45	5.83	C2
4	(24,10)	20.24	14.86	C2
5	(30,45)	48.87	36	C2
6	(85,70)	104.35	93	C2
7	(71,80)	101.41	89	C2
8	(60,78)	93	80	C2
9	(55,52)	70	58	C2
10	(80,91)	115.52	103.32	C2

After assigning data points to the corresponding clusters, the next step is to calculate the new centroid values. These values are calculated by finding the means of the coordinates of the data points that belong to a particular cluster.

For cluster C1, there is currently only one point i.e. (5,3), therefore the mean of the coordinates remain same and the new centroid value for c1 will also be (5,3).

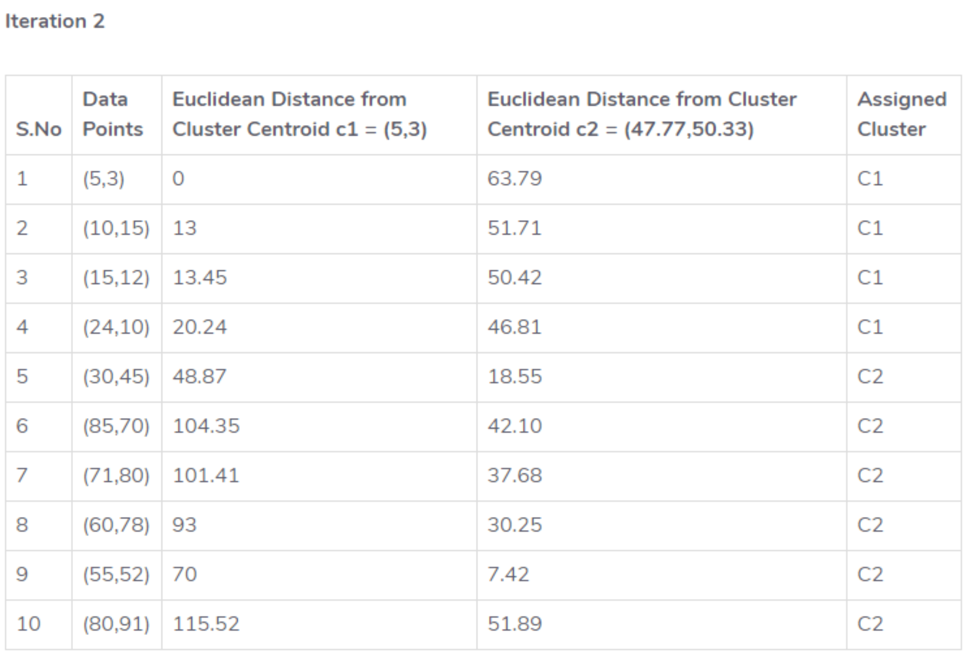
For C2, there are currently 9 data points. We name the coordinates of data points as x and y. The new value for x coordinate of centroid c2 can be calculated by determining the mean of x coordinates of all 9 points that belong to cluster C2 as given below:

c2(x) = (10 + 15 + 24 + 30 + 85 + 71 + 60 + 55 + 80) / 9 = 47.77
The new value for y coordinate of centroid c2 can be calculated by determining the mean of all y coordinates of all 9 points that belong to cluster C2.

c2(y) = (15 + 12 + 10 + 45 + 70 + 80 + 78 + 52 + 91) / 9 = 50.33

**The updated centroid value for c2 will now be {47.77, 50.33}.

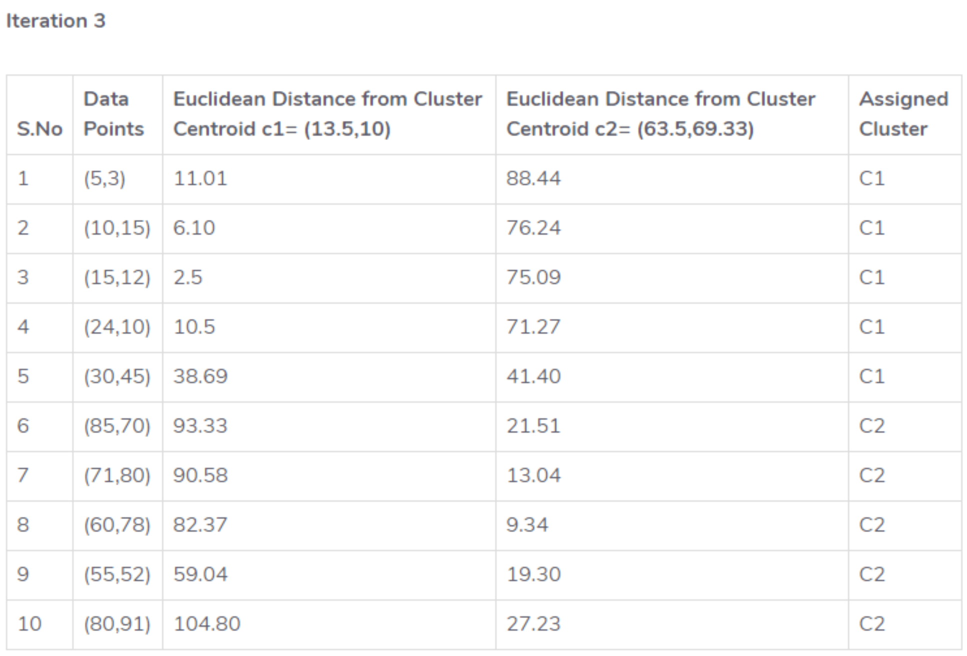
For the next iteration, the new centroid values for c1 and c2 will be used and the whole process will be repeated. The iterations continue until the centroid values stop updating.


Iteration 2

S.No	Data Points	Euclidean Distance from Cluster Centroid c1 = (5,3)	Euclidean Distance from Cluster Centroid c2 = (47.77,50.33)	Assigned Cluster
1	(5,3)	0	63.79	C1
2	(10,15)	13	51.71	C1
3	(15,12)	13.45	50.42	C1
4	(24,10)	20.24	46.81	C1
5	(30,45)	48.87	18.55	C2
6	(85,70)	104.35	42.10	C2
7	(71,80)	101.41	37.68	C2
8	(60,78)	93	30.25	C2
9	(55,52)	70	7.42	C2
10	(80,91)	115.52	51.89	C2

c1(x) = (5, 10, 15, 24) / 4 = 13.5
c1(y) = (3, 15, 12, 10) / 4 = 10.0 **Updated c1 to be (13.5, 10.0).

c2(x) = (30 + 85 + 71 + 60 + 55 + 80) / 6 = 63.5
c2(y) = (45 + 70 + 80 + 78 + 52 +91) / 6 = 69.33
**Updated c2 to be (63.5, 69.33).


Iteration 3

S.No	Data Points	Euclidean Distance from Cluster Centroid c1= (13.5,10)	Euclidean Distance from Cluster Centroid c2= (63.5,69.33)	Assigned Cluster
1	(5,3)	11.01	88.44	C1
2	(10,15)	6.10	76.24	C1
3	(15,12)	2.5	75.09	C1
4	(24,10)	10.5	71.27	C1
5	(30,45)	38.69	41.40	C1
6	(85,70)	93.33	21.51	C2
7	(71,80)	90.58	13.04	C2
8	(60,78)	82.37	9.34	C2
9	(55,52)	59.04	19.30	C2
10	(80,91)	104.80	27.23	C2

c1(x) = (5, 10, 15, 24, 30) / 5 = 16.8
c1(y) = (3, 15, 12, 10, 45) / 5 = 17.0
**Updated c1 to be (16.8, 17.0).

c2(x) = (85 + 71 + 60 + 55 + 80) / 5 = 70.2
c2(y) = (70 + 80 + 78 + 52 + 91) / 5 = 74.2
**Updated c2 to be (70.2, 74.2).

Iteration 4

S.No	Data Points	Euclidean Distance from Cluster Centroid c1 = (16.8,17)	Euclidean Distance from Cluster Centroid c2 = (70.2,74.2)	Assigned Cluster
1	(5,3)	18.30	96.54	C1
2	(10,15)	7.08	84.43	C1
3	(15,12)	5.31	83.16	C1
4	(24,10)	10.04	79.09	C1
5	(30,45)	30.95	49.68	C1
6	(85,70)	86.37	15.38	C2
7	(71,80)	83.10	5.85	C2
8	(60,78)	74.74	10.88	C2
9	(55,52)	51.80	26.90	C2
10	(80,91)	97.31	19.44	C2

At the end of fourth iteration, the updated values of C1 and C2 are same as they were at the end of the third iteration. This means that data cannot be clustered any further. c1 and c2 are the centroids for C1 and C2. To classify a new data point, the distance between the data point and the centroids of the clusters is calculated. Data point is assigned to the cluster whose centroid is closest to the data point.

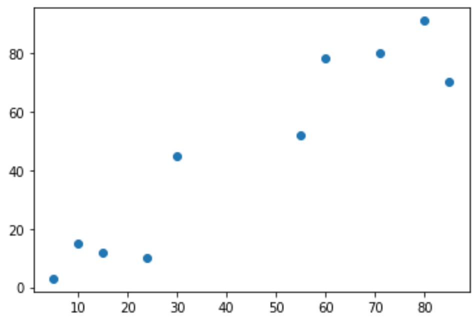
K-means Clustering with Scikit-Learn

Now that we know how the K-means clustering algorithm actually works, let's see how we can implement it with Scikit-Learn.

In [2]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
from sklearn.cluster import KMeans

# prepare the data that we want to cluster
X = np.array([[5,3],
     [10,15],
     [15,12],
     [24,10],
     [30,45],
     [85,70],
     [71,80],
     [60,78],
     [55,52],
     [80,91],])

# Visualize the Data  .. type your code..
plt.scatter(X[:,0],X[:,1], label='True Position')
plt.show()
```



In [3]:
```
# Create Clusters. Assume with no. of clusters = 2
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)

# see what centroid values the algorithm generated for the final clusters
print(kmeans.cluster_centers_)
```

```
[[16.8 17. ]
 [70.2 74.2]]
```

Here the first row contains values for the coordinates of the first centroid i.e. (16.8 , 17) and the second row contains values for the coordinates of the other centroid i.e. (70.2, 74.2). You can see that these values are similar to what we calculated manually for centroids c1 and c2 in the last section.

In [4]:
```
# To see the labels for the data point, -> use labels_ attribute
print(kmeans.labels_)


# The output is a one dimensional array of 10 elements corresponding
# to the clusters assigned to our 10 data points.
```

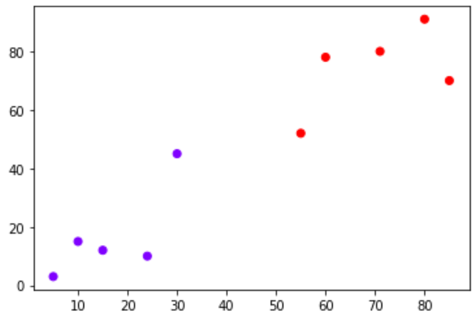
```
[0 0 0 0 0 1 1 1 1 1]
```

Here the first five points have been clustered together and the last five points have been clustered. Here 0 and 1 are merely used to represent cluster IDs and have no mathematical significance.

In [5]:
```
# Let's plot the data points again on the graph and
# visualize how the data has been clustered.

plt.scatter(X[:,0],X[:,1], c=kmeans.labels_, cmap='rainbow')
```

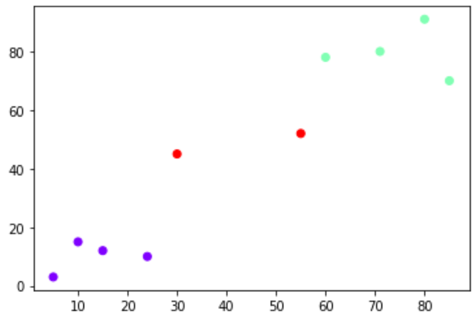
Out[5]:
```
<matplotlib.collections.PathCollection at 0xf40ea70>
```



In [6]:
```
# execute K-means algorithm with three clusters and see the output graph
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)

plt.scatter(X[:,0],X[:,1], c=kmeans.labels_, cmap='rainbow')
```

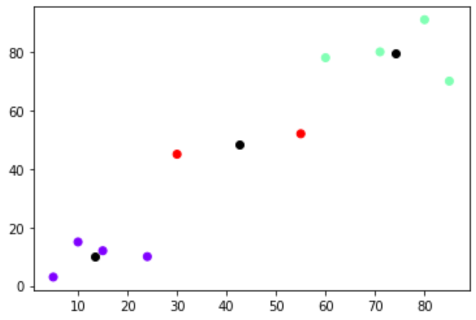
Out[6]:
```
<matplotlib.collections.PathCollection at 0xf44fd90>
```



In [7]:
```
# let's plot the points along with the centroid coordinates
# of each cluster to see how the centroid positions effects clustering.

plt.scatter(X[:,0], X[:,1], c=kmeans.labels_, cmap='rainbow')
plt.scatter(kmeans.cluster_centers_[:,0] ,kmeans.cluster_centers_[:,1], color='black')
```

Out[7]:
```
<matplotlib.collections.PathCollection at 0x5ed1a30>
```



Recommended Reading

Clustering with Gaussian Mixture Models

Clustering is an essential part of any data analysis. Using an algorithm such as K-Means leads to hard assignments, meaning that each point is definitively assigned a cluster center. This leads to some interesting problems: what if the true clusters actually overlap? What about data that is more spread out; how do we assign clusters then?

For further reading of concepts related to Expectation-Maximization Algorithm and GMM, please see this blog article. https://pythonmachinelearning.pro/clustering-with-gaussian-mixture-models/ (https://pythonmachinelearning.pro/clustering-with-gaussian-mixture-models/)