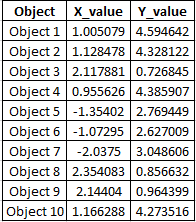
**Experiment-11:** Apply EM algorithm to cluster a Heart Disease Data Set. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

**K-means**

K-means is an unsupervised learning method for clustering data points. The algorithm iteratively divides data points into K clusters by minimizing the variance in each cluster.

**#Analysis**

**make\_blobs** from **sklearn.datasets** module for doing this.



# Imports

from sklearn.datasets.samples\_generator import make\_blobs

# Generate 2D data points

X, \_ = make\_blobs(n\_samples=10, centers=3, n\_features=2, cluster\_std=0.2, random\_state=0)

# Convert the data points into a pandas DataFrame

import pandas as pd

# Generate indicators for the data points

obj\_names = []

for i in range(1, 11):

obj = "Object " + str(i)

obj\_names.append(obj)

# Create a pandas DataFrame with the names and (x, y) coordinates

data = pd.DataFrame({

'Object': obj\_names,

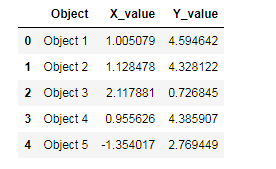
'X\_value': X[:, 0],

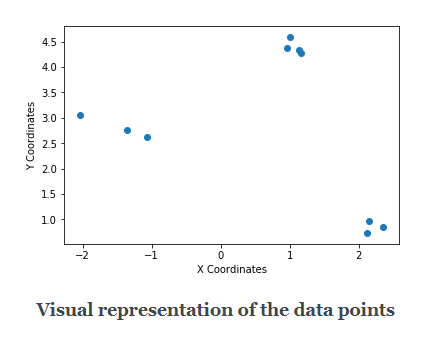
'Y\_value': X[:, -1]

})

# Preview the data

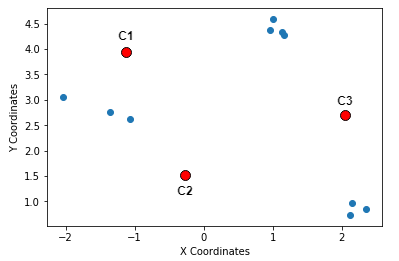
print(data.head())





* **Initialize random centroids**

You start the process by taking three(as we decided K to be 3) random points (in the form of (x, y)). These points are called **centroids**which is just a fancy name for denoting *centers*. Let’s name these three points - **C1, C2,** and **C3**so that you can refer them later.

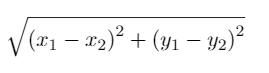


**Step 1 in K-Means: Random centroids**

* **Calculate distances between the centroids and the data points**

 Next, you measure the distances of the data points from these three randomly             chosen points. A very popular choice of distance measurement function, in this         case, is the **[Euclidean distance](https://en.wikipedia.org/wiki/Euclidean_distance).**

     Briefly, if there are n points on a 2D space(just like the above figure) and their           coordinates are denoted by (x\_i, y\_i), then the Euclidean distance between any         two points (**(x1, y1)** and**(x2, y2)**) on this space is given by:



**Equation for Euclidean distance**

       Suppose the coordinates of C1, C2 and C3 are - **(-1, 4)**, **(-0.2, 1.5)** and **(2, 2.5)**         respectively. Let’s now write a few lines of Python code which will calculate the           Euclidean distances between the data-points and these randomly chosen                     centroids. We start by initializing the centroids.

# Initialize the centroids

c1 = (-1, 4)

c2 = (-0.2, 1.5)

c3 = (2, 2.5)

   Next, we write a small helper function to calculate the Euclidean distances between the data points and centroids.

# A helper function to calculate the Euclidean diatance between the data

# points and the centroids

def calculate\_distance(centroid, X, Y):

distances = []

# Unpack the x and y coordinates of the centroid

c\_x, c\_y = centroid

# Iterate over the data points and calculate the distance using the

# given formula

for x, y in list(zip(X, Y)):

root\_diff\_x = (x - c\_x) \*\* 2

root\_diff\_y = (y - c\_y) \*\* 2

distance = np.sqrt(root\_diff\_x + root\_diff\_y)

distances.append(distance)

return distances

 We can now apply this function to the data points and assign the results in the  DataFrame accordingly.

# Calculate the distance and assign them to the DataFrame accordingly

data['C1\_Distance'] = calculate\_distance(c1, data.X\_value, data.Y\_value)

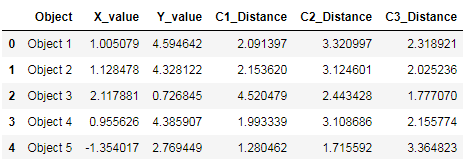
data['C2\_Distance'] = calculate\_distance(c2, data.X\_value, data.Y\_value)

data['C3\_Distance'] = calculate\_distance(c3, data.X\_value, data.Y\_value)

# Preview the data

print(data.head())

   The output should be like the following:



**Snapshot of the Euclidean distances between the data points and the centroids**

  Time to study the next step in the algorithm.

* **Compare, assign, mean and repeat**

This is fundamentally the last step of the K-Means clustering algorithm. Once             you have the distances between the data points and the centroids, you compare         the distances and take the *smallest ones*. The centroid to which the distance for         a particular data point is the smallest, that centroid gets assigned as the cluster         for that particular data point. Let’s do this programmatically.

# Get the minimum distance centroids

data['Cluster'] = data[['C1\_Distance', 'C2\_Distance', 'C3\_Distance']].apply(np.argmin, axis =1)

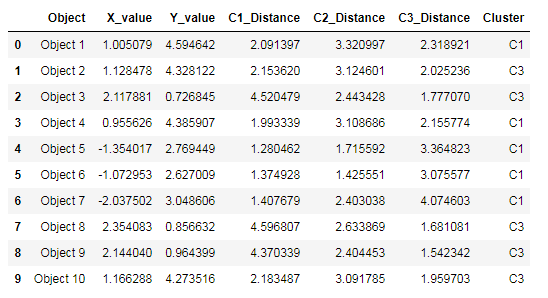
# Map the centroids accordingly and rename them

data['Cluster'] = data['Cluster'].map({'C1\_Distance': 'C1', 'C2\_Distance': 'C2', 'C3\_Distance': 'C3'})

# Get a preview of the data

print(data.head(10))

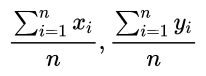
You get a nicely formatted output:



**Clusters after one iteration of K-means**

With this step, we complete an iteration of the K-Means cloistering algorithm.            Take a closer look at the output - **there’s no C2 in there**.

    Now comes the most interesting part of *updating the centroids*by determining          the **mean** values of the coordinates of the data points (which should be                        belonging to some centroid by now). Hence the name K-**Means**. This is how the        mean calculation looks like:



**Mean update in K-Means (n denotes the number of data points belonging in a cluster)**

 The following lines of code does this for you:

# Calculate the coordinates of the new centroid from cluster 1

x\_new\_centroid1 = data[data['Cluster']=='C1']['X\_value'].mean()

y\_new\_centroid1 = data[data['Cluster']=='C1']['Y\_value'].mean()

# Calculate the coordinates of the new centroid from cluster 2

x\_new\_centroid2 = data[data['Cluster']=='C3']['X\_value'].mean()

y\_new\_centroid2 = data[data['Cluster']=='C3']['Y\_value'].mean()

# Print the coordinates of the new centroids

print('Centroid 1 ({}, {})'.format(x\_new\_centroid1, y\_new\_centroid1))

print('Centroid 2 ({}, {})'.format(x\_new\_centroid2, y\_new\_centroid2))

You get:

https://paper-attachments.dropbox.com/s_B2D4C7E216B003565119EB315150BA36BC0129AD3B4159870ADDFFD84D67B5E4_1555225218664_image.png

**Coordinates of the new centroids**

Notice that the algorithm, in its first iteration, grouped the data points into two     clusters although we specified this number to be 3. The following animation gives     you a pretty good overview of how centroid updates take place in the K-Means algorithm.

