7. Apply **EM algorithm** to cluster a set of data stored in a .CSV file. 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.

**Theory:**

**Expectation Maximization algorithm**

* The basic approach and logic of this clustering method is as follows.
* Suppose we measure a single continuous variable in a large sample of observations. Further, suppose that the sample consists of two clusters of observations with different means (and perhaps different standard deviations); within each sample, the distribution of values for the continuous variable follows the normal distribution.
* The goal of EM clustering is to estimate the means and standard deviations for each cluster so as to maximize the likelihood of the observed data (distribution).
* Put another way, the EM algorithm attempts to approximate the observed distributions of values based on mixtures of different distributions in different clusters. The results of EM clustering are different from those computed by k-means clustering.
* The latter will assign observations to clusters to maximize the distances between clusters.
* The EM algorithm does not compute actual assignments of observations to clusters, but classification probabilities.
* In other words, each observation belongs to each cluster with a certain probability. Of course, as a final result we can usually review an actual assignment of observations to clusters, based on the (largest) classification probability.
* K means Clustering
* The algorithm will categorize the items into k groups of similarity. To calculate that similarity, we will use the euclidean distance as measurement.
* **The algorithm works as follows:**

1. First, we initialize k points, called means, randomly.
2. We categorize each item to its closest mean and we update the mean’s coordinates, which are the averages of the items categorized in that mean so far.
3. We repeat the process for a given number of iterations and at the end, we have our clusters.

* The “points” mentioned above are called means, because they hold the mean values of the items categorized in it.
* To initialize these means, we have a lot of options.
* An intuitive method is to initialize the means at random items in the data set.
* Another method is to initialize the means at random values between the boundaries of the data set (if for a feature x the items have values in [0,3], we will initialize the means with values for x at [0,3]).

**Dataset:**

This is the "Iris" dataset. Originally published at UCI Machine Learning Repository: Iris Data Set, this small dataset from 1936 is often used for testing out machine learning algorithms and visualizations

The Iris dataset includes three iris species with 50 samples each as well as some properties about each flower. One flower species is linearly separable from the other two, but the other two are not linearly separable from each other.

**Attribute information:**

The columns in this dataset are:

* Id
* SepalLengthCm
* SepalWidthCm
* PetalLengthCm
* PetalWidthCm
* Species

Each row of the table represents an iris flower, including its species and dimensions of its botanical parts, sepal and petal, in centimetres.

**Some instances from the dataset:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Id | SepalLengthCm | SepalWidthCm | PetalLengthCm | PetalWidthCm | Species |
| 1 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 2 | 4.9 | 3 | 1.4 | 0.2 | Iris-setosa |
| 3 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 4 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 5 | 5 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| 6 | 5.4 | 3.9 | 1.7 | 0.4 | Iris-setosa |
| 7 | 4.6 | 3.4 | 1.4 | 0.3 | Iris-setosa |
| 8 | 5 | 3.4 | 1.5 | 0.2 | Iris-setosa |
| 9 | 4.4 | 2.9 | 1.4 | 0.2 | Iris-setosa |

**Program:**

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.cluster import KMeans

import pandas as pd

import numpy as np

# import some data to play with

iris = datasets.load\_iris()

X = pd.DataFrame(iris.data)

X.columns = ['Sepal\_Length','Sepal\_Width','Petal\_Length','Petal\_Width']

y = pd.DataFrame(iris.target)

y.columns = ['Targets']

# Build the K Means Model

model = KMeans(n\_clusters=3)

model.fit(X) # model.labels\_ : Gives cluster no for which samples belongs to

# # Visualise the clustering results

plt.figure(figsize=(14,14))

colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications using Petal features

plt.subplot(2, 2, 1)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[y.Targets], s=40)

plt.title('Real Clusters')

plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

# Plot the Models Classifications

plt.subplot(2, 2, 2)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[model.labels\_], s=40)

plt.title('K-Means Clustering')

plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

# General EM for GMM

from sklearn import preprocessing

# transform your data such that its distribution will have a

# mean value 0 and standard deviation of 1.

scaler = preprocessing.StandardScaler()

scaler.fit(X)

xsa = scaler.transform(X)

xs = pd.DataFrame(xsa, columns = X.columns)

from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n\_components=3)

gmm.fit(xs)

gmm\_y = gmm.predict(xs)

plt.subplot(2, 2, 3)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[gmm\_y], s=40)

plt.title('GMM Clustering')

plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

print('Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.')

**Output:**

Observation: The GMM using EM algorithm-based clustering matched the true labels more closely than the Kmeans.



