**Program 9: 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. Add Python ML library classes/API in the program.**

**Unsupervised Learning:**

In machine learning, unsupervised learning is a class of problems in which one seeks to determine how the data are organized. It is distinguished from supervised learning (and reinforcement learning) is that the learner is given only unlabeled examples.

**Dataset:**

➢ Iris dataset

➢ Number of Attributes:2 1. sepal length 2. sepal width

➢ Number of instances:150

**Clustering Algorithms -**

1. **K-means clustering:**

* It is a type of unsupervised learning, which is used when you have unlabeled data (i.e., data without defined categories or groups).
* The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable K.
* Data points are clustered based on feature similarity.
* The results of the K-means clustering algorithm are:
* The centroids of the K clusters, which can be used to label new data
* Labels for the training data (each data point is assigned to a single cluster)

Each centroid of a cluster is a collection of feature values which define the resulting groups. Examining the centroid feature weights can be used to qualitatively interpret what kind of group each cluster represents.

The k-means is a partitional clustering algorithm.

Let the set of data points (or instances) be as follows:

D = {x1, x2, …, xn}, where

x = (xi1, xi2, …, xir), is a vector in a real-valued space X ⊆ Rr, and r is the number of attributes in the data.

The k-means algorithm partitions the given data into k clusters with each cluster having a center called a centroid.

k is specified by the user.

Given k, the k-means algorithm works as follows:

**Algorithm K-means( k, D )**

1. Identify the k data points as the initial centroids (cluster centers).
2. Repeat step 1.
3. For each data point x ϵ D do.
4. Compute the distance from x to the centroid.
5. Assign x to the closest centroid (a centroid represents a cluster).
6. Re-compute the centroids using the current cluster memberships until the stopping criterion is met.
7. **Expectation–maximization**

➢ EM algorithm is an iterative method to find maximum likelihood estimates of parameters in

statistical models, where the model depends on unobserved latent variables.

➢ Iteratively learn probabilistic categorization model from unsupervised data.

➢ Initially assume random assignment of examples to categories “Randomly label” data

➢ Learn initial probabilistic model by estimating model parameters θ from randomly labeled data

➢ Iterate until convergence:

* **Expectation (E-step):** Compute P(ci | E) for each example given the current model, and probabilistically re-label the examples based on these posterior probability estimates.
* **Maximization (M-step):** Re-estimate the model parameters, θ, from the probabilistically relabeled data.

**The EM Algorithm for Gaussian Mixtures**

➢ The probability density function for multivariate\_normal is where mu is the mean, Sigma the covariance matrix, and k is the dimension of the space where x takes values.

**Algorithm:**

An arbitrary initial hypothesis h=<μ1, μ2 ,.., μk> is chosen.

**The EM Algorithm iterates over two steps:**

**Step 1 (Estimation, E):** Calculate the expected value E[zij] of each hidden variable zij, assuming that the current hypothesis h=<μ1, μ2 ,.., μk> holds.

**Step 2 (Maximization, M):** Calculate a new maximum likelihood hypothesis h‟=<μ1‟, μ2‟ ,.., μk‟>, assuming the value taken on by each hidden variable zij is its expected value E[zij] calculated in step 1. Then replace the hypothesis h=<μ1, μ2 ,.., μk> by the new hypothesis h‟=<μ1‟, μ2‟ ,.., μk‟> and iterate.

**EM ALGORITHM**

import numpy as np

import math

import matplotlib.pyplot as plt

import csv

def get\_binomial\_log\_likelihood(obs,probs):

N = sum(obs);

k = obs[0]

# number of heads 12

binomial\_coeff = math.factorial(N) / (math.factorial(N-k) \*math.factorial(k))

prod\_probs = obs[0]\*math.log(probs[0]) + obs[1]\*math.log(1-probs[0])

log\_lik = binomial\_coeff + prod\_probs

return log\_lik

data=[]

with open("cluster.csv") as tsv:

for line in csv.reader(tsv):

data=[int(i) for i in line]

head\_counts = np.array(data)

tail\_counts = 10-head\_counts

#print(tail\_counts)

experiments = list(zip(head\_counts,tail\_counts))

pA\_heads = np.zeros(100); pA\_heads[0] = 0.60

pB\_heads = np.zeros(100); pB\_heads[0] = 0.50

#print(pA\_heads)

#print(pB\_heads)

delta = 0.001

j = 0

improvement = float('inf')

print(improvement)

while (improvement>delta):

expectation\_A = np.zeros((len(experiments),2), dtype=float)

expectation\_B = np.zeros((len(experiments),2), dtype=float)

for i in range(0,len(experiments)):

e = experiments[i]

ll\_A =get\_binomial\_log\_likelihood(e,np.array([pA\_heads[j],1-pA\_heads[j]]))

ll\_B = get\_binomial\_log\_likelihood(e,np.array([pB\_heads[j],1-pB\_heads[j]]))

weightA = math.exp(ll\_A) / ( math.exp(ll\_A) + math.exp(ll\_B) )

weightB = math.exp(ll\_B) / ( math.exp(ll\_A) + math.exp(ll\_B) )

expectation\_A[i] = np.dot(weightA, e)

expectation\_B[i] = np.dot(weightB, e)

pA\_heads[j+1] = sum(expectation\_A)[0] / sum(sum(expectation\_A));

pB\_heads[j+1] = sum(expectation\_B)[0] / sum(sum(expectation\_B));

improvement = ( max( abs(np.array([pA\_heads[j+1],pB\_heads[j+1]])-

np.array([pA\_heads[j],pB\_heads[j]]) )) )

print(np.array([pA\_heads[j+1],pB\_heads[j+1]])-np.array([pA\_heads[j],pB\_heads[j]]) )

j = j+1

plt.figure();

plt.plot(range(0,j),pA\_heads[0:j])

plt.plot(range(0,j),pB\_heads[0:j])

plt.show()

**K - Mean ALGORITHM**

from sklearn.cluster import KMeans

from sklearn import metrics

import numpy as np

import matplotlib.pyplot as plt

import csv

data=[]

ydata=[]

with open("cluster.csv") as tsv:

for line in csv.reader(tsv):

data=[int(i) for i in line]

print(data)

ydata=[10-int(i) for i in line]

print(ydata)

x1 = np.array(data)

x2 = np.array(ydata)

print(x1)

print(x2)

plt.plot()

plt.xlim([0, 10])

plt.ylim([0, 10])

plt.title('Dataset')

plt.scatter(x1, x2)

plt.show()

plt.plot()

X = np.array(list(zip(x1, x2))).reshape(len(x1), 2)

print(X)

colors = ['b', 'g','r']

markers = ['o', 'v', 's']

K = 3

kmeans\_model = KMeans(n\_clusters=K).fit(X)

#print(kmeans\_model)

plt.plot()

for i, l in enumerate(kmeans\_model.labels\_):

print(i,l)

plt.plot(x1[i], x2[i], color=colors[l], marker=markers[l],ls='None')

plt.xlim([0, 10])

plt.ylim([0, 10])

**OUTPUT**

[ 0.00151004 -0.12639463]

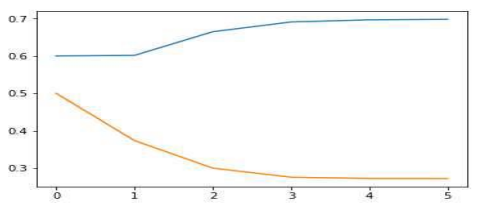
[ 0.06329684 -0.07364559]

[ 0.02612144 -0.02430469]

[ 0.00573256 -0.00349713]

[ 0.00137583 -0.00012734]

[0.00044257 0.00015722]



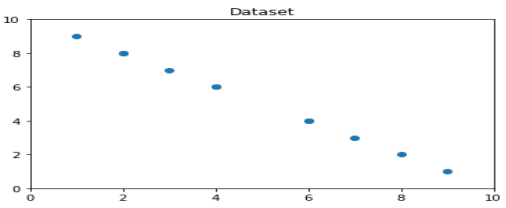
Program 8KM

[2, 9, 8, 4, 7, 2, 3, 1, 6, 4, 6]

[8, 1, 2, 6, 3, 8, 7, 9, 4, 6, 4]

[2 9 8 4 7 2 3 1 6 4 6]

[8 1 2 6 3 8 7 9 4 6 4]



[[2 8]

[9 1]

[8 2]

[4 6]

[7 3]

[2 8]

[3 7]

[1 9]

[6 4]

[4 6]

[6 4]]

0 1

1 0

2 0

3 1

4 0

5 1

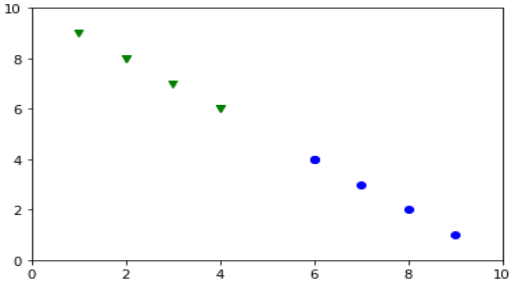
6 1

7 1

8 0

9 1

10 0



**Result:**