**EE 511**

**PROJECT # 3**

BY

PRANAV GUNDEWAR

USC ID: 4463612994

EMAIL: [gundewar@usc.edu](mailto:gundewar@usc.edu)

**Problem 1 Testing Faith**

**Summary:**

* Old Faithful data set contains 2-D random variables- first dimension is the duration of the old faithful geyser eruptions. The second is the waiting time between eruptions.
* We have to run K means clustering for k = 2.
* I have used K-means function from sklearn library and Pandas library to read data.

**Approach:**

* Clustering is a type of **Unsupervised learning**. This is very often used when you don’t have labelled data. **K-Means Clustering** is one of the popular clustering algorithm. The goal of this algorithm is to find groups(clusters) in the given data.
* Algorithm works as follows assuming we have inputs x1, x2, x3, …, xn and value of **K.**

1. We randomly pick **K** cluster centres (centroids). Let’s assume these are c1, c2,…,ck and we can say that;

C=c1,c2,…,ck

C is the set of all centroids.

1. In this step we assign each input value to closest centres. This is done by calculating Euclidean(L2) distance between the point and each centroid.

Argmin ci∈C dist(ci,x)2

Where dist(.)dist(.) is the Euclidean distance.

1. In this step, we find the new centroid by taking the average of all the points assigned to that cluster.

ci=1|Si|∑xi∈Si xi

Si is the set of all points assigned to the ith cluster.

1. Repeat Step 2 and 3 until none of the cluster assignments change.

**Code:**

"""

@author: Pranav Gundewar

Project 3

Q1- K Means Clustering

Dataset: old faithful

"""

# Importing Libraries

import numpy as np

import pandas as pd

from matplotlib import pyplot as plt

from sklearn.cluster import KMeans

# Reading the data using

data = pd.read\_table('old\_faithful\_data.txt', delim\_whitespace = True)

print('Data Size:',data.shape)

f1 = data['d2'].values

f2 = data['d3'].values

X = np.array(list(zip(f1, f2)))

# Number of clusters

kmeans = KMeans(n\_clusters=2,n\_init=1,init='random',max\_iter = 500)

# Fitting the input data

kmeans = kmeans.fit(X)

# Getting the cluster labels

labels = kmeans.predict(X)

# Centroid values

centroids = kmeans.cluster\_centers\_

#Plotting the scatter plot

colors = ['g', 'b', 'y', 'c', 'm']

fig, ax = plt.subplots()

for i in range (2):

points = np.array([X[j] for j in range(len(X)) if labels[j] == i])

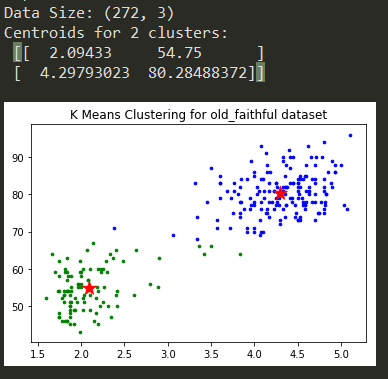
ax.scatter(points[:, 0], points[:, 1], s=7, c=colors[i])

plt.scatter(centroids[:,0],centroids[:,1], marker='\*', s=200, c='red')

plt.title('K Means Clustering for old\_faithful dataset')

print('Centroids for 2 clusters: \n',centroids)

**Result and Analysis:**



**Figure 1. K means Clustering for oldfaithful data set**

From figure 1, we can observe the scatter plot for given dataset. It has been divided using 2 clusters and cluster plot of data along with centroids can be observed. For 2 clusters, 2 centroids have been displayed in the figure.

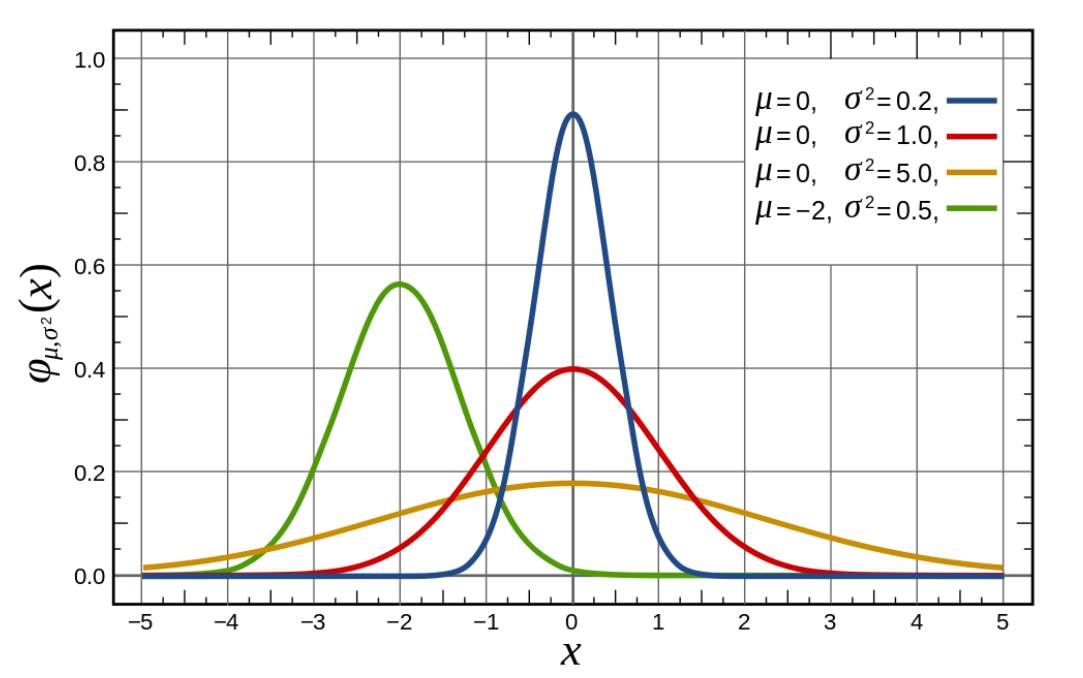
From plot, we can observe some misclassified points. That happens sometimes due to random choice of centres.

There are certain disadvantages of K means clustering. It has hard decision and fails to classify when the data has overlapping clusters

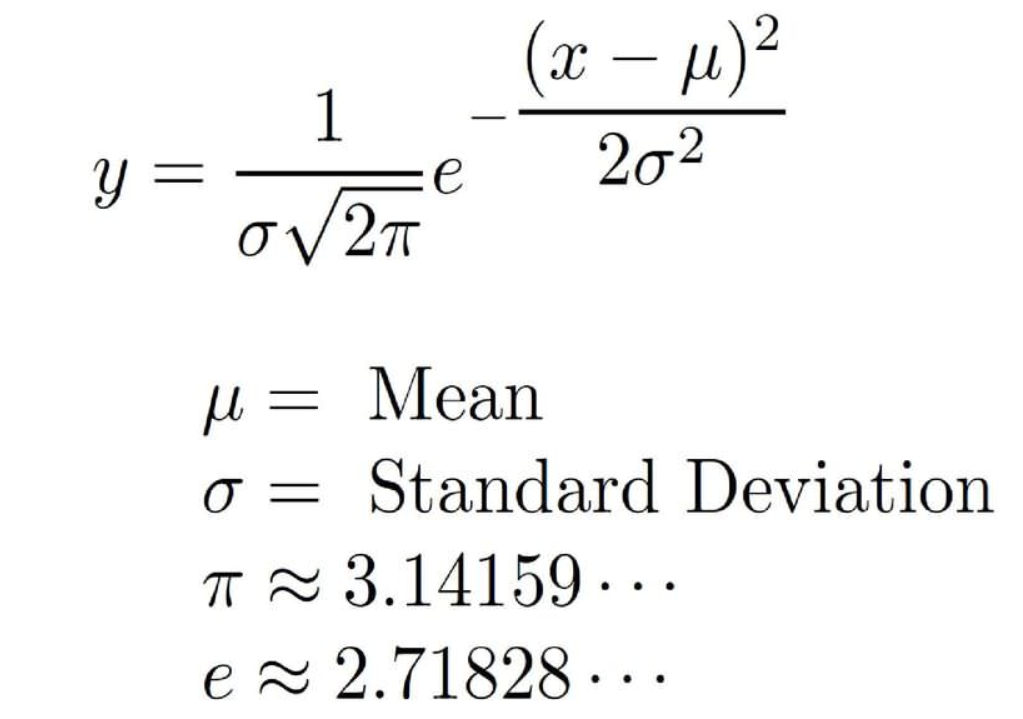
**Problem 2 Expectation Maximization**

**Summary:**

* We have to write a routine got GMM RNG for 2 sub-populations. We are supposed to implement the expectation maximization (EM) algorithm for estimating the pdf parameters of 2-D GMMs from samples
* I have used sklearn mixture model to generate GMM model.
* Gaussian Mixture model is a probability distribution that consists of multiple probability distributions.

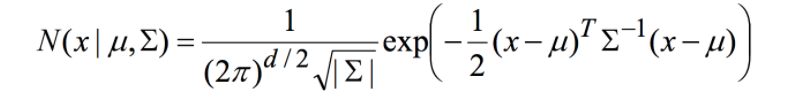


**Figure 2. Gaussian distribution - the probability density**



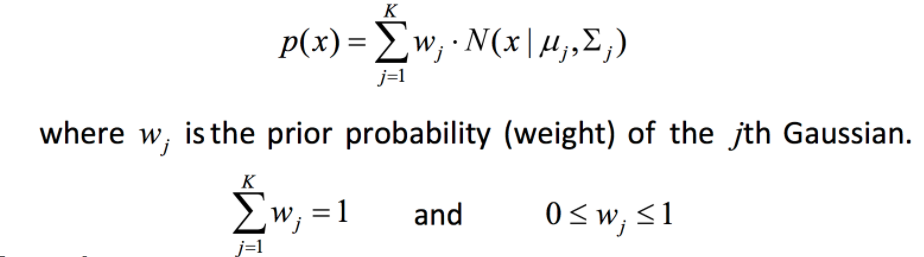
**Figure 3. Gaussian distribution mathematical formula**

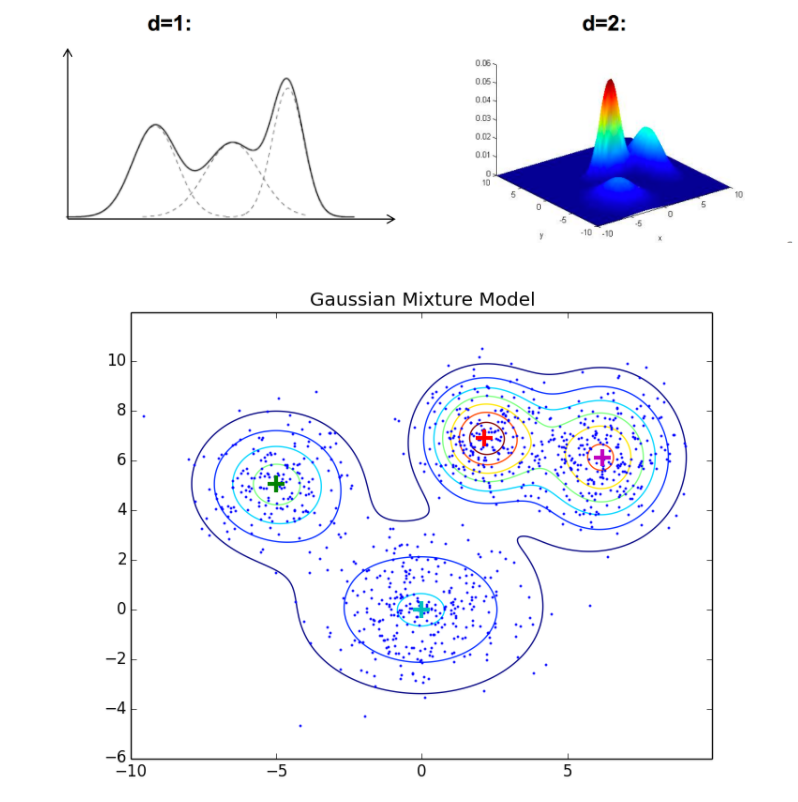
* For Gaussian mixture model, for d dimensions, distribution of a vector x = (x1, .. , xd)T is defined by



Where mu and sigma are mean and covariance matrix of Gaussian.

* Covariance is a measure of how changes in one variable are associated with changes in a second variable. Specifically, covariance measures the degree to which two variables are linearly associated. However, it is also often used informally as a general measure of how monotonically related two variables are.
* The probability given in a mixture of K gaussians is



 **Figure 4. Gaussian Mixture Model**

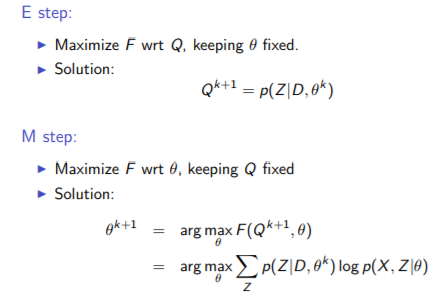
* Expectation–maximization (EM) algorithm is an [iterative method](https://en.wikipedia.org/wiki/Iterative_method) to find [maximum likelihood](https://en.wikipedia.org/wiki/Maximum_likelihood) of [parameters](https://en.wikipedia.org/wiki/Parameter) in [statistical models](https://en.wikipedia.org/wiki/Statistical_model), where the model depends on unobserved [latent variables](https://en.wikipedia.org/wiki/Latent_variable).
* The EM iteration alternates between performing an expectation (E) step, which creates a function for the expectation of the [log-likelihood](https://en.wikipedia.org/wiki/Likelihood_function#Log-likelihood) evaluated using the current estimate for the parameters, and a maximization (M) step, which computes parameters maximizing the expected log-likelihood found on the *E* step.
* These parameter-estimates are then used to determine the distribution of the latent variables in the next E step**.**

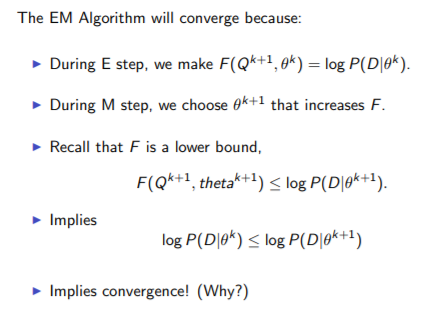
**Approach:**

* We will generate gaussian distributed random data using GMM module from sklearn and fit the data.
* The Expectation Maximization (EM) Algorithm:

1. Expectation: Fix model and estimate missing labels.

2. Maximization: Fix missing labels (or a distribution over the missing labels) and find the model that maximizes the expected log-likelihood of the data.





Part A of problem asks us to generate 2-dimensional RNG for a Gaussian mixture model (GMM) pdf with 2 subpopulations.

**Code:**

"""

@author: Pranav Gundewar

Project 3

Q2 Part A - EM Clustering

Dataset: old faithful

"""

from scipy.stats import multivariate\_normal as mvn

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.colors import LogNorm

from sklearn import mixture

import pandas as pd

# Part A- 2-dimensional RNG for a Gaussian mixture model (GMM) pdf with 2 subpopulations

n\_samples = 300

# generate random sample, two components

np.random.seed(10)

# generate spherical data centered on (20, 20)

#C1 = np.array([[1, 0], [0, 1]])

shifted\_gaussian = np.random.randn(n\_samples, 2) + np.array([20, 20])

# generate zero centered stretched Gaussian data

C2 = np.array([[2.0, -1.0], [3.5, .7]])

stretched\_gaussian = np.dot(np.random.randn(n\_samples, 2), C2) + np.array([2, 2])

# concatenate the two datasets into the final training set

X\_train = np.vstack([shifted\_gaussian, stretched\_gaussian])

# fit a Gaussian Mixture Model with two components

clf = mixture.GaussianMixture(n\_components=2, covariance\_type='full')

clf.fit(X\_train)

# display predicted scores by the model as a contour plot

x = np.linspace(-20., 40.)

y = np.linspace(-20., 40.)

X, Y = np.meshgrid(x, y)

XX = np.array([X.ravel(), Y.ravel()]).T

Z = -clf.score\_samples(XX)

Z = Z.reshape(X.shape)

CS = plt.contour(X, Y, Z, norm=LogNorm(vmin=1.0, vmax=1000.0),

levels=np.logspace(0, 3, 10))

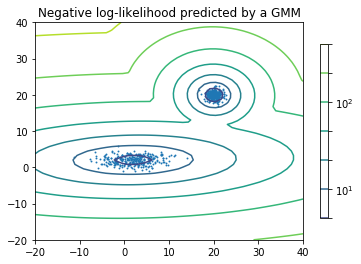
CB = plt.colorbar(CS, shrink=0.8, extend='both')

plt.scatter(X\_train[:, 0], X\_train[:, 1], .8)

plt.title('Negative log-likelihood predicted by a GMM')

plt.axis('tight')

plt.show()

 **Figure 5. Gaussian Mixture Model RNG for 2 components**

**Code:**

"""

@author: Pranav Gundewar

Project 3

Q2 Part - EM Clustering

Dataset: old faithful

"""

#Importing Libraries

import numpy as np

import pandas as pd

import time

start = time.clock()

class GMM:

def \_\_init\_\_(self, k = 3, eps = 0.0001):

self.k = k ## number of clusters

self.eps = eps ## threshold to stop `epsilon`

# All parameters from fitting/learning are kept in a named tuple

# from collections import namedtuple

def fit\_EM(self, X, max\_iters = 1000):

# n = number of data-points, d = dimension of data points

n, d = X.shape

# randomly choose the starting centroids/means

## as 3 of the points from datasets

mu = X[np.random.choice(n, self.k, False), :]

# initialize the covariance matrices for each gaussians

Sigma= [np.eye(d)] \* self.k

# initialize the probabilities/weights for each gaussians

w = [1./self.k] \* self.k

# responsibility matrix is initialized to all zeros

# we have responsibility for each of n points for eack of k gaussians

R = np.zeros((n, self.k))

### log\_likelihoods

log\_likelihoods = []

P = lambda mu, s: np.linalg.det(s) \*\* -.5 \*\* (2 \* np.pi) \*\* (-X.shape[1]/2.) \

\* np.exp(-.5 \* np.einsum('ij, ij -> i',\

X - mu, np.dot(np.linalg.inv(s) , (X - mu).T).T ) )

# Iterate till max\_iters iterations

while len(log\_likelihoods) < max\_iters:

# E - Step

## Vectorized implementation of e-step equation to calculate the

## membership for each of k -gaussians

for k in range(self.k):

R[:, k] = w[k] \* P(mu[k], Sigma[k])

### Likelihood computation

log\_likelihood = np.sum(np.log(np.sum(R, axis = 1)))

log\_likelihoods.append(log\_likelihood)

## Normalize so that the responsibility matrix is row stochastic

R = (R.T / np.sum(R, axis = 1)).T

## The number of datapoints belonging to each gaussian

N\_ks = np.sum(R, axis = 0)

# M Step

## calculate the new mean and covariance for each gaussian by

## utilizing the new responsibilities

for k in range(self.k):

## means

mu[k] = 1. / N\_ks[k] \* np.sum(R[:, k] \* X.T, axis = 1).T

x\_mu = np.matrix(X - mu[k])

## covariances

Sigma[k] = np.array(1 / N\_ks[k] \* np.dot(np.multiply(x\_mu.T, R[:, k]), x\_mu))

## and finally the probabilities

w[k] = 1. / n \* N\_ks[k]

# check for onvergence

if len(log\_likelihoods) < 2 : continue

if np.abs(log\_likelihood - log\_likelihoods[-2]) < self.eps: break

## bind all results together

from collections import namedtuple

self.params = namedtuple('params', ['mu', 'Sigma', 'w', 'log\_likelihoods', 'num\_iters'])

self.params.mu = mu

self.params.Sigma = Sigma

self.params.w = w

self.params.log\_likelihoods = log\_likelihoods

self.params.num\_iters = len(log\_likelihoods)

return self.params

def plot\_log\_likelihood(self):

import pylab as plt

plt.plot(self.params.log\_likelihoods)

plt.title('Log Likelihood vs iteration plot')

plt.xlabel('Iterations')

plt.ylabel('log likelihood')

plt.show()

def predict(self, x):

p = lambda mu, s : np.linalg.det(s) \*\* - 0.5 \* (2 \* np.pi) \*\*\

(-len(x)/2) \* np.exp( -0.5 \* np.dot(x - mu , \

np.dot(np.linalg.inv(s) , x - mu)))

probs = np.array([w \* p(mu, s) for mu, s, w in \

zip(self.params.mu, self.params.Sigma, self.params.w)])

return probs/np.sum(probs)

def demo\_2d():

### generate the random data

np.random.seed(3)

m1, cov1 = [16, 9], [[3, 0], [0, 5]] #Ellipsoidal Covarince Matrix

data1 = np.random.multivariate\_normal(m1, cov1, 90)

m2, cov2 = [6, 13], [[2, 0], [0, 2]] #Spherical Covarince Matrix

data2 = np.random.multivariate\_normal(m2, cov2, 45)

m3, cov3 = [4, 7], [[3.5, 0.5], [0.5, 2.5]] # poorly-separated subpopulations

data3 = np.random.multivariate\_normal(m3, cov3, 65)

X = np.vstack((data1,np.vstack((data2,data3))))

# X = np.vstack((data1,data2))

np.random.shuffle(X)

# np.savetxt('sample.csv', X, fmt = "%.4f", delimiter = ",")

####

# data = pd.read\_table('old\_faithful\_data.txt', delim\_whitespace = True)

# print('Data Size:',data.shape)

# f1 = data['d2'].values

# f2 = data['d3'].values

# X = np.array(list(zip(f1, f2)))

gmm = GMM(3, 0.000001)

params = gmm.fit\_EM(X, max\_iters= 300)

# print (params.log\_likelihoods)

import pylab as plt

from matplotlib.patches import Ellipse

def plot\_ellipse(pos, cov, nstd=2, ax=None, \*\*kwargs):

def eigsorted(cov):

vals, vecs = np.linalg.eigh(cov)

order = vals.argsort()[::-1]

return vals[order], vecs[:,order]

if ax is None:

ax = plt.gca()

vals, vecs = eigsorted(cov)

theta = np.degrees(np.arctan2(\*vecs[:,0][::-1]))

# Width and height are "full" widths, not radius

width, height = 2 \* nstd \* np.sqrt(abs(vals))

ellip = Ellipse(xy=pos, width=width, height=height, angle=theta, \*\*kwargs)

ax.add\_artist(ellip)

return ellip

def show(X, mu, cov):

plt.cla()

K = len(mu) # number of clusters

colors = ['g', 'c', 'b', 'k', 'm', 'y', 'r']

plt.plot(X.T[0], X.T[1],'.', color = 'r')

for k in range(K):

plot\_ellipse(mu[k], cov[k], alpha=0.6, color = colors[k % len(colors)])

fig = plt.figure(figsize = (13, 6))

fig.add\_subplot(121)

show(X, params.mu, params.Sigma)

fig.add\_subplot(122)

plt.plot(np.array(params.log\_likelihoods))

plt.title('Log Likelihood vs iteration plot')

plt.xlabel('Iterations')

plt.ylabel('log likelihood')

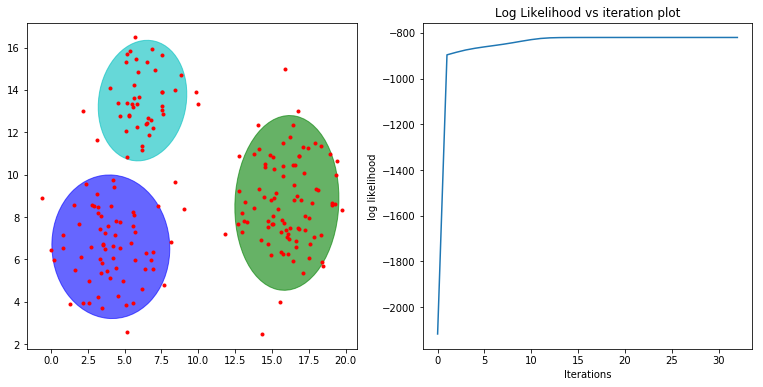
plt.show()

# print (gmm.predict(np.array([2, 1])))

if \_\_name\_\_ == "\_\_main\_\_":

demo\_2d()

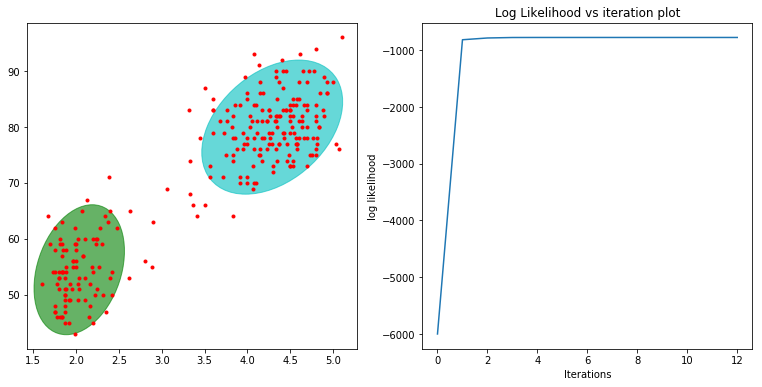
print ('Time required to run EM algorithm for 300 iterations: ', time.clock() - start)



**Figure 6. GMM EM distribution for poorly distributed, ellipsoidal and spherical covariance matrices**



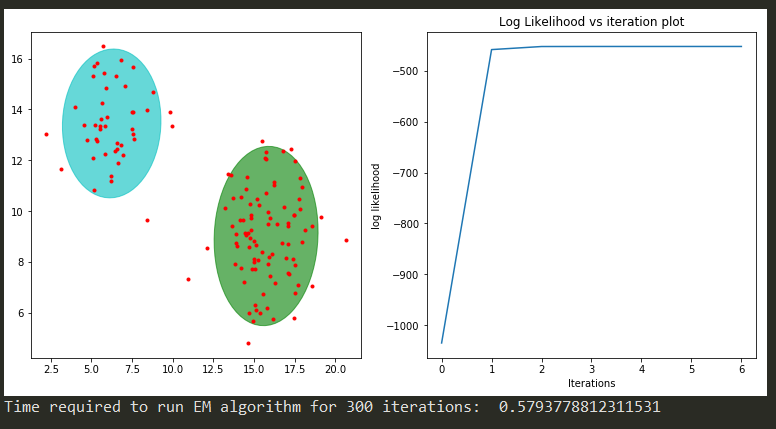
**Figure 7. Time required for EM algorithm**



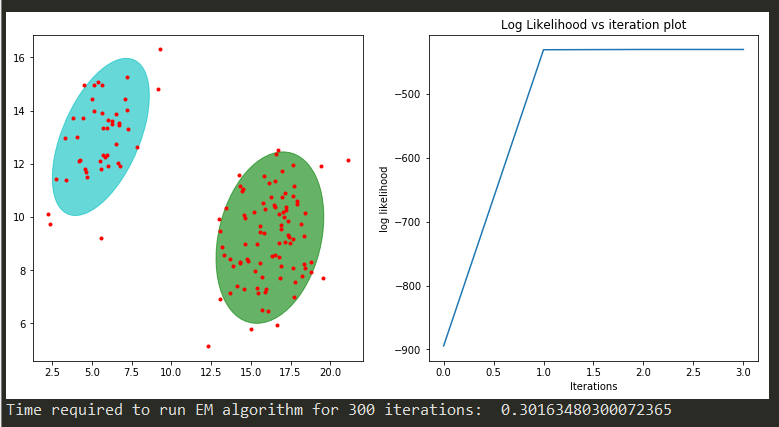
**Figure 8. EM clustering for Faithful Dataset**



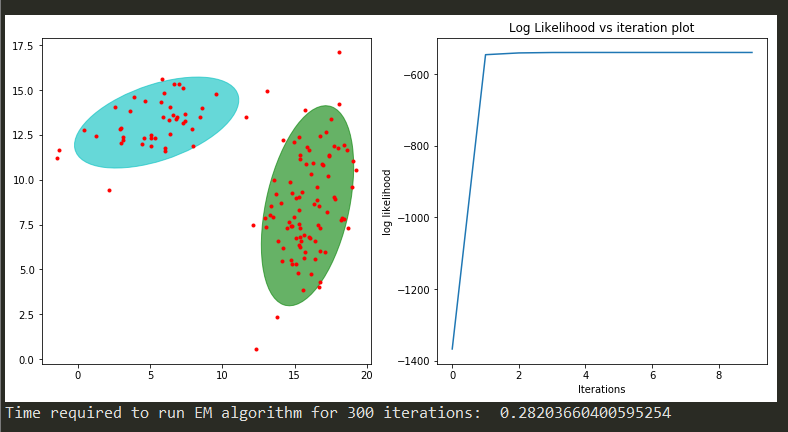
**Figure 9. EM clustering for Faithful Dataset**



**Figure 9. Spherical Covariance Matrix**



**Figure 9. Ellipsoidal Covariance Matrix**



**Figure 9. Poorly Distributed Covariance Matrix**

From above figures, we can observe that poorly distributed data took least time whereas spherical data took most time to run.

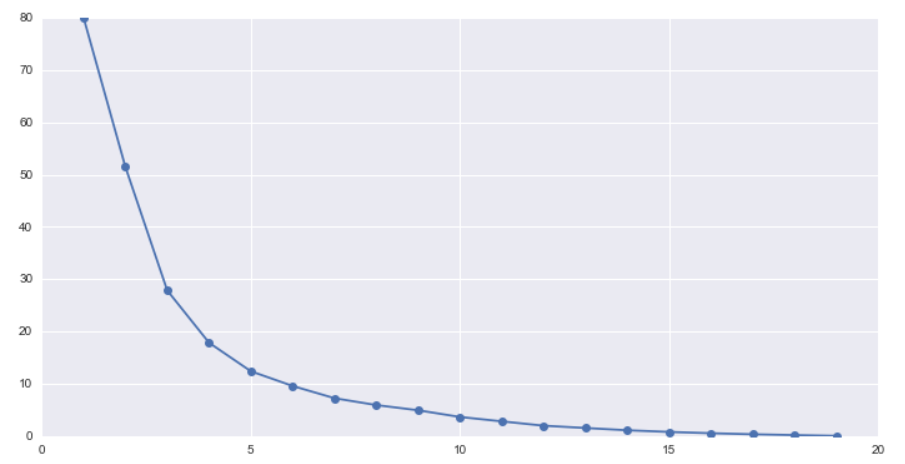
**Problem 3 Clusters of Text**

**Summary:**

* “nips-87-92” data set contains a *bag-of-words* data set for NIPS papers from 1987-1992. Columns in this bag-of-words model represent the (scaled) number of times a specified word appears in the different documents. The first column specifies a document id for each paper.
* Data has 700 rows and approx. 11000 columns hence we cannot plot the data.
* We have to run k means clustering for range of k and find the optimal k using elbow method.
* Report your clusters by listing the document ids for each cluster.

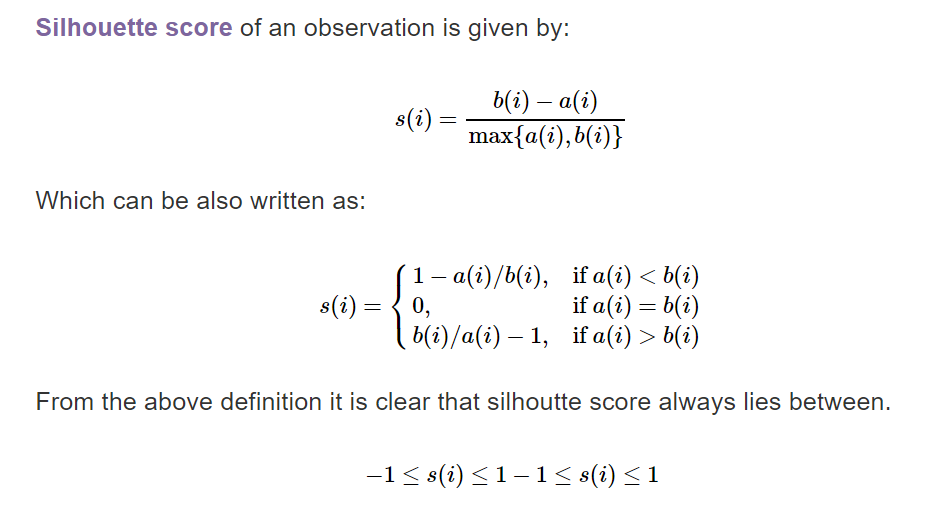
**Approach:**

* **K means Clustering** is implemented on row vectors. But number of clusters are unknown hence k means routine is run number of times to find optimal value of k.
* This value of k can be found out using elbow method.
* **The Elbow method** is a method of interpretation and validation of consistency within cluster analysis designed to help finding the appropriate number of clusters in a dataset.



**Figure 10. Elbow Method**

* The elbow diagram shows that the gain in explained variance reduces significantly from 3 to 4 to 5. So, optimal number of clusters could either 4 or 5. The actual number of clusters chosen can be finally based on business context and convenience of dealing with number of segments or clusters.
* **The silhouette value** is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation).
* The silhouette ranges from -1 to 1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters.
* The silhouette can be calculated with any distance metric, such as the Euclidean distance or the Manhattan distance.



* Score closer to 1 means assigned to the cluster correctly and score closer to -1 is assigned to a wrong cluster. A score close to 0 means the point lies between almost at the boundary of both the clusters.

**Code:**

"""

@author: Pranav Gundewar

Project 3

Q3- Clustes of Text

Dataset: nips-87-92

"""

# Importing Libraries

import numpy as np

from sklearn import metrics

from sklearn.cluster import KMeans

# Reading the data using

inputData = np.genfromtxt('nips-87-92.csv', delimiter=',')

data = inputData[1:,2:]

print('Data Size:',data.shape)

label = np.zeros(700, dtype=np.int)

label = inputData[1:,1]

k = []

distance = []

score = []

for i in range (2, 10):

kmeans = KMeans(n\_clusters = i, init='k-means++').fit(data)

clusterLables = kmeans.labels\_

k.append(i)

distance.append(kmeans.inertia\_)

a = metrics.silhouette\_score(data, clusterLables, metric='euclidean')

score.append(a)

index = np.argmax(score)+2

print ("For ",i," Clusters","\tSilhouette Coefficient: %0.3f"

% metrics.silhouette\_score(data, clusterLables, metric='euclidean'))

print('Optimal Value of k: ',index)

kNewMeans = KMeans(n\_clusters = index,init='k-means++').fit(data)

newClusterLables = kNewMeans.labels\_

class1 = []

class2 = []

for i in range (len(label)):

if newClusterLables[i] == 1:

class2.append(label[i])

else:

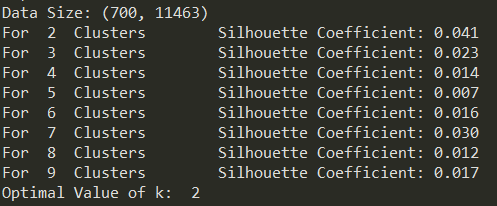
class1.append(label[i])

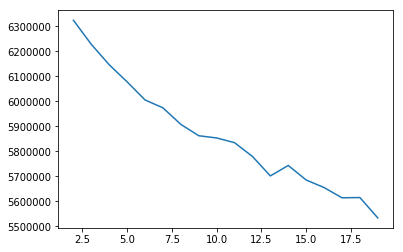
print('Document IDs for Cluster 1: \n',class1)

print('Document IDs for Cluster 2: \n',class2)

**Result and Analysis-**

* K means routine has been run for numerous values of k clusters.
* Optimal value of cluster k is found out using silhouette value. The highest value denotes optimal value.
* Using optimal value of k, k means routine is used to cluster the data in k clusters and labels are found out using attributes.
* Comparing labels with document id, document ids for each cluster has been printed on console.
* To improve the performance of clustering, we can mini batch k means instead of k means to obtain faster convergence.
* Due to random centres, you are not guaranteed to get knee point while using elbow method. Hence higher value of silhouette value is taken as optimal k.
* For the plot of optimal k, we are not getting knee point in graph which indicates data is highly overlapping and is not linearly separable. Hence, I have chosen k which has highest score.





**Fig. 11 – Plot of K value vs score**

From figure 11, we can conclude that data is highly overlapping. Hence it cannot be clustered.

