ELL784 : Assignment-3 Report

Raghav Mittal (2014MT10600)

November 12, 2016

1 Basic Details

Let us first look what unsupervised learning means.
Unsupervised Learning : Unsupervised learning is a very famous field of machine learning where we predict output for a given data for which no labels are given. This means that we have no correct outputs like supervised learning to check our predictions calculated. The most famous unsupervised learning method is cluster prediction which is used for data analysis to find pattern matching or probabilistic distances.

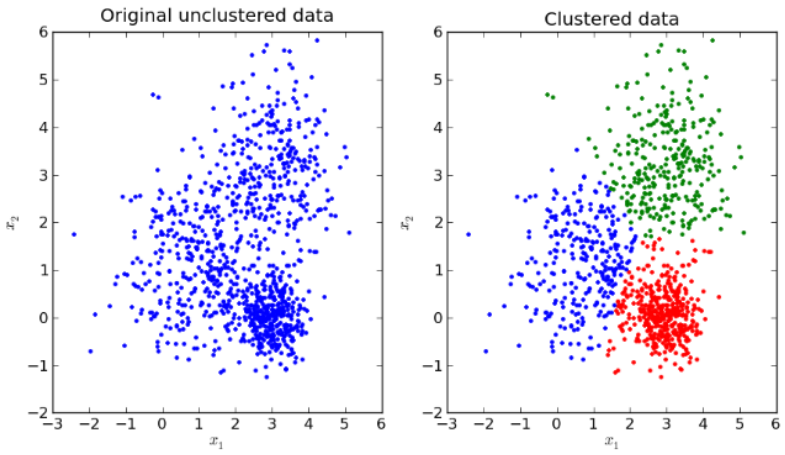

Figure 1: Sample Data to be Clustered

Let us see each algorithm in detail now and see how accuracy changes by change of various parameters and then I will explain how I proceeded my code for each method.

2 K-Means Clustering

The process followed here is very simple. First of all we have to anticipate the number of clusters into which the data is divided approximately. The name is given K-means clustering since the no. of clusters are k. After selecting the no. of clusters i.e. k we have to assume k means or centres, one for every cluster which is the mean over all the datapoints contained in the particular cluster. The choosing of these centres have to be done in a proper way since this choice have a significant effect on our prediction. Ideally these should be chosen as far as possible.

The next step is to find to which cluster every data point belong. For this we calculate the distance of every data point from every centre of k clusters and find the closest centre to the particulate data point. We assign this cluster to the point and repeat the procedure for the whole dataset. After assigning clusters to data points we calculate the new means/centres of each clusters with the help of points in them.

$$Mean_i = \frac{\sum_{k=1}^{c_i} x_i^k}{c_i}$$

Here $Mean_i$ denotes the mean of i^{th} cluster, i=(1,2,....c_i), c_i denotes the no. of points in i^{th} cluster, x_i^k denotes the k^{th} point in i^{th} cluster.

Figure 2: Sample Data to be Clustered

This process is repeated until every point gets a stable cluster assigned to it i.e. every data point should not change the cluster to which it belongs or when less than 2-3% of data points are unstable. This algorithm aims at minimising a cost function known as squared error function given by :

$$\mathrm{J} = \sum_{i=1}^{c} \sum_{j=1}^{c_i} \|x_i - v_j\|^2$$

In this assignment by observing the data given to us I have anticipated 6 clusters in which the dataset is divided approximately. The means initialised at the very first step of K-Means Clustering are chosen by observing data and estimating of clusters and their midpoints through matplotlib (python plotting library). Data points initially are given cluster numbers according to the minimum distance calculated from the means taken. After this the basic procedure of K-Means is followed as explained above.

Lets have a look on how accuracy varied on changing no. of clusters Let's now look at the second method which I implemented i.e. Gaussian Mixture Means method

Number of Clusters (Assumed)	Accuracy Obtained in %
5	72.20
6	78.85
7	78.05
8	72.79

Figure 3: Accuracy vs Cluster Number

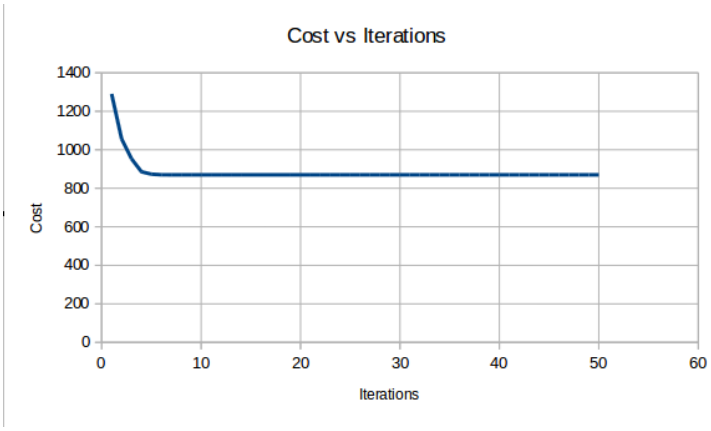

Figure 4: Plot of Cost Function

3 Gaussian Mixture Means

Gaussian Mixture Means is also a widely used method for prediction in unsupervised learning techniques. It works on a basic assumption that every data point has aroused from a mixture of gaussian or normal disributions. This method is an extension of K means algorithm with information about the covariance of the data and its weighted mean. Here also we assume our data to be separated in k Gaussian/Normal distributions and then try to predict the means and covariances of each cluster. Since we are talking about multi variate/dimensional data variance will be analogical to covariance matrix.

A gaussian mixture model is a weighted sum of k gaussian distributions

which is :

$$\mathrm{p}(x) = \sum_{i=1}^{k} \pi_i g(x|\mu_i, \sum_i)$$

Sum of all these π_i is 1.Here x is a D dimensional data point and π_i is the weight of i_{th} gaussian distribution. Gaussian Disribution g is calculated as Here also we assume first k means and their covariance matrices respectively.

$$g(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = \frac{1}{(2\pi)^{D/2}|\Sigma_i|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)' \, \Sigma_i^{-1} \, (\mathbf{x} - \boldsymbol{\mu}_i)\right\}.$$

After this we calculate probability of every datapoint to lie in each of the k distributions γ_j given by the formula :

$$\frac{\boldsymbol{\pi}_\mathbf{k}\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_\mathbf{k}, \boldsymbol{\Sigma}_\mathbf{k})}{\sum_{\mathbf{j=1}}^{\mathbf{K}} \boldsymbol{\pi}_\mathbf{j}\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_\mathbf{j}, \boldsymbol{\Sigma}_\mathbf{j})} \quad \textbf{where,} \quad \boldsymbol{\pi}_\mathbf{k} = \frac{\mathbf{N}_\mathbf{k}}{\mathbf{N}}$$

Here N_k denotes the no.of points in k^{th} distribution and N is the total no. of points in the dataset. We then calculate the maximum of all these probabilites and k for which this probability is highest is assigned to the datapoint as the distribution in which it should lie. This is also known as Expectation Maximisation Algorithm. After calculating the distribution for each point to lie in we update the means and covarince matrix of all distributions according to the new data points which are assigned to them given by the formulae:

$$\mu_j = \frac{\sum_{n=1}^{N} \gamma_j(x_n) x_n}{\sum_{n=1}^{N} \gamma_j(x_n)}$$

$$\boldsymbol{\Sigma}_\mathbf{j} = \frac{\sum_{\mathbf{n=1}}^{\mathbf{N}} \boldsymbol{\gamma}_\mathbf{j}(\mathbf{x}_\mathbf{n})(\mathbf{x}_\mathbf{n} - \boldsymbol{\mu}_\mathbf{j})(\mathbf{x}_\mathbf{n} - \boldsymbol{\mu}_\mathbf{j})^\mathbf{T}}{\sum_{\mathbf{n=1}}^{\mathbf{N}} \boldsymbol{\gamma}_\mathbf{j}(\mathbf{x}_\mathbf{n})}$$

$$\pi_j = \frac{\sum_{n=1}^{N} \gamma_j(x_n)}{N}$$

This process is repeated until every point or more than 97% gets assigned to a distribution which is stable for it i.e. it does not change its distribution further after any iteration.Lets now look at a sample example and how this algorithm works on the datsset.

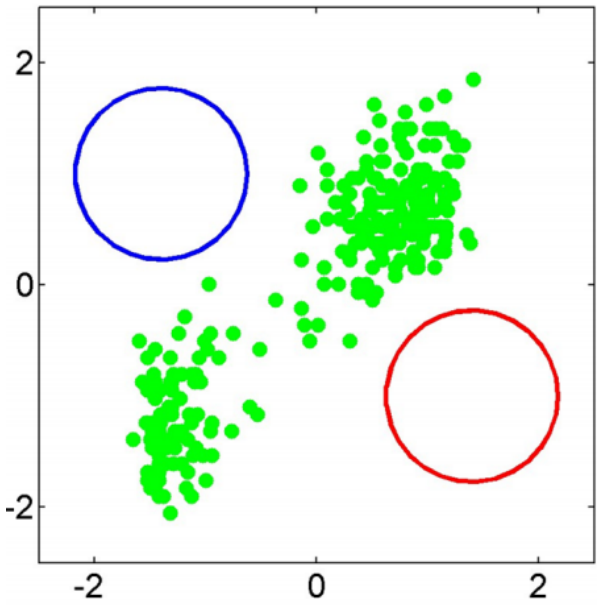

Figure 5: Sample Data Given

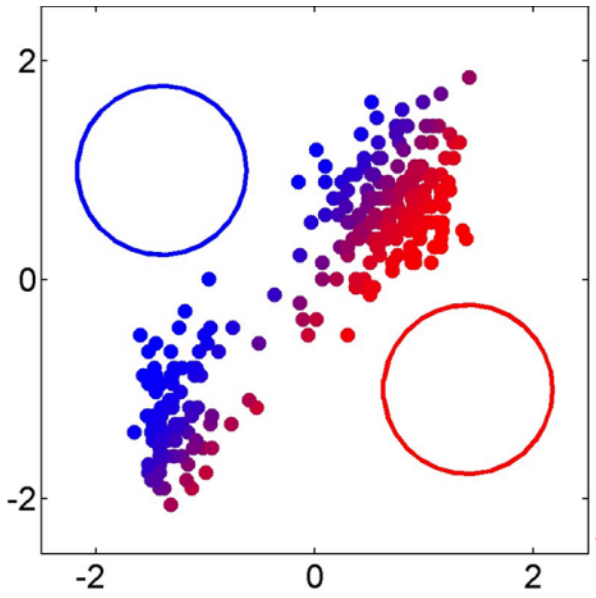

Figure 6: Initial Assignment of distributions

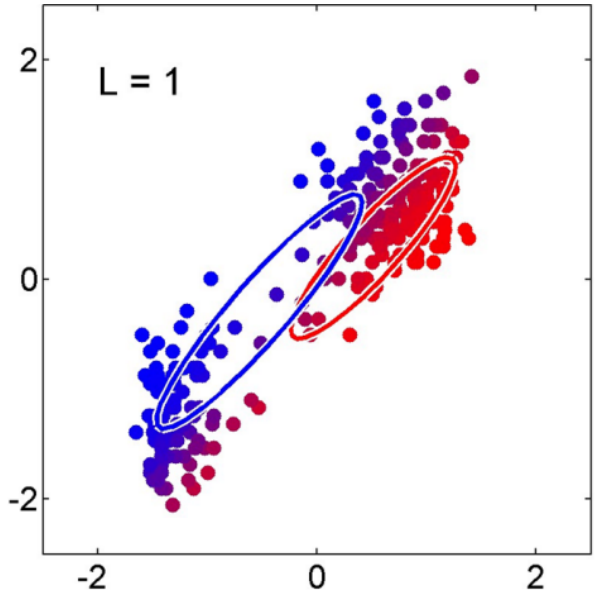

Figure 7: Data After L=1 iteration

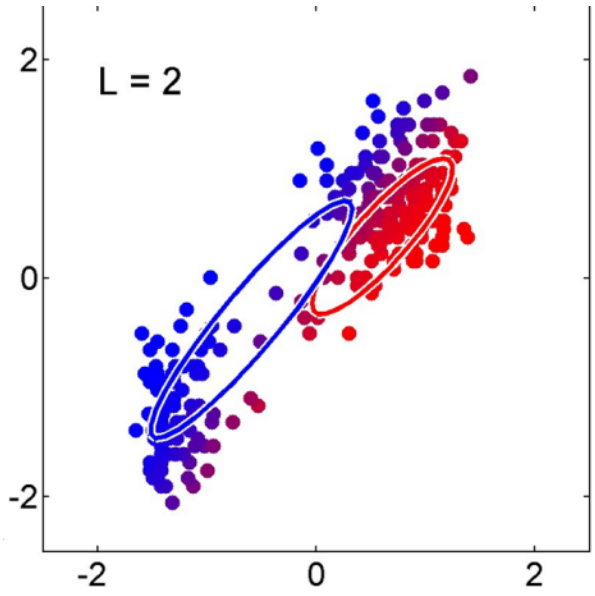

Figure 8: Data After L=2 iterations

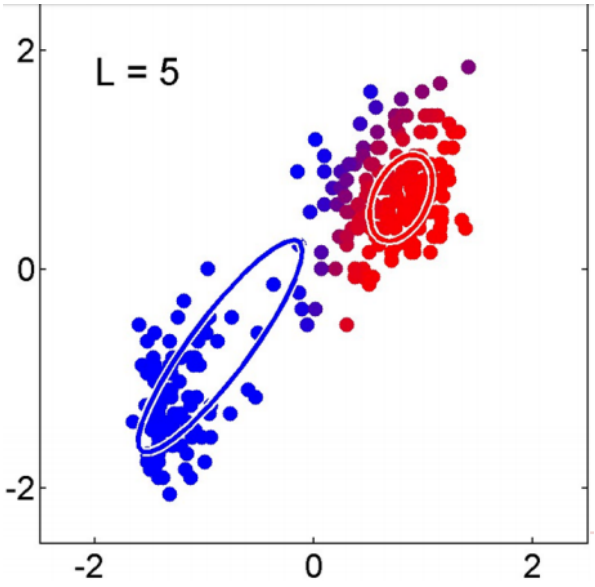

Figure 9: Data After L=5 iterations

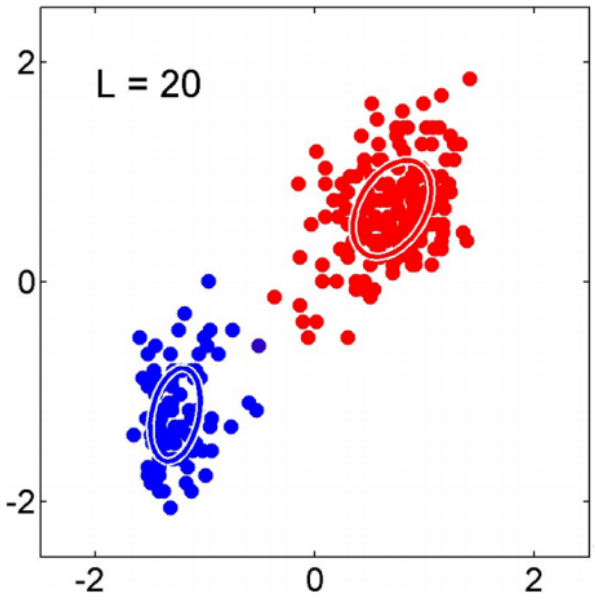

Figure 10: Data After L=20 iterations

In my code I have initialised the means of gaussian distributions that are the results of K means algorithm. This is done since the predictions very much depend on the parameters initialised. Similarly the mixing coefficients i.e. the no. of data points in each distribution are calculated and stored in an array and then the process is repeated until convergence according to the procedure explained above. Lets look at accuracies that change with no. of gaussian distributions assumed.

No. of Distributions (Assumed)	Accuracy Obtained in %
5	71.10
6	77.05
7	76.62
8	71.79

4 Difference Between the algorithms

Please note that in Expectation Mximisation Gaussian Mixture Models, we consider variance and covariance into account which are not considered in K means algorithm. But infact k means is a problem but an algorithm and we solve k means bby using Lloyd algorithm which is a modification of Expectaation Maximisation algorithm with a centroid type modeland hard assignments i.e. points belonging to one cluster does not have an effect on other clusters.

When we solve using k means that is variance minimisation we co-incidentally minimise squared distances since variance contributuion of within-cluster sum of squares (WCSS) of distances is equivalent to euclidean distances. Also we assign clusters to the data points in a hard method as per the minimum euclidean distance since the square root of this distance function is monotone.Here means are not the one which are responsible for optimising of Euclidean distances but the WCSS function.

Clusters are represented in K means using their means/centroids only while in Gaussian Mixture Means method every cluster is identified by not only just mean value but also with covariance matrices and its mixing coefficients also(fraction of points in a cluster). Clusters are best separated when they are in spherical form in K means but in general we get polygon shaped clusters. Whereas EM algorithm doesnot hard assigns the data points rather it soft assigns them and give them a probability which is the measure of expectation finding them in a particular cluster and does not depend on Euclidean distance L2 norm. This makes K means biased towards spherical clusters

Code for K-Means Clustering Algorithm

```
# import of required libraries
import numpy as np
import csv
import math

# Assumed no. of clusters to be 6
clusters = 6

# Basic function to convert string into a int array
# since the data points are given in string form this
# is used to convert them into an integer array(matrix)
def strToInt(row):
    # initialised a new array and then appended the two strings
#contained in float form
    arr=[]
    arr.append(float(row[0]))
    arr.append(float(row[1]))
    return (arr)

# Save the model in a pickel file
def save_model(file_name, model_list):
    import pickle
    with open(file_name, 'wb') as fid:
        pickle.dump(model_list, fid)

# Loading the model from the pickle file
def load_model(file_name):
    import pickle
    with open(file_name, 'rb') as fid:
        model = pickle.load(fid)
    return model

# A function which calculates distance between 2 points which are
#two dimensional
def calcDistance(p1, p2):
    return math.sqrt((p1[0]-p2[0])*(p1[0]-p2[0])+
    (p1[1]-p2[1])*(p1[1]-p2[1]))

# this function takes a datapoint and set of means as input and return the mi
```

```
# distance of this point from the set of means and also tells which
#particular mean is giving the
#  shortest distance
def calcMinDistance(data, clusterCentres):
    # variable for storing minimum distance is initialised to infinity
    minDis = math.inf
    distances = []
    # cluster centre is the set of means
    # distance from every point is calculated and at every step it is
        # checked
    # whether it is less than minimum distance calculated yet or not.
        #If yes minimum
    # distance is updated
    for i in range(0,len(clusterCentres)):
        x = calcDistance(data, clusterCentres[i])
        distances.append(x)
        if(x<=minDis):
            minDis = x
    return [minDis, distances.index(minDis)]

# this function calculates the mean of a data given by using all the
 #data points contained in it
def calcMean(data):
    # if data is empty mean is returned to be (0,0)
    if(len(data)==0):
        return np.array([0.,0.])
    # temporary variable for storing mean
    x = np.array([0.,0.])
    # every point is added to the temporary variable
    for i in range(0,len(data)):
        x = x + data[i]
    # finally it is divided by no. of points in the set
    x = x/len(data)
    return x

# This function calculates the minimum distance of every data point
 #from each of
# k centres and assigns the data point a cluster from whose mean it
 #is nearest
def kMeansCluster(dataset, clusterCentres):
```

```
    # Array for storing indexes of min distance cluster
    clusterIdentifier = []
    for i in range(0, len(dataset)):
        j = calcMinDistance(dataset[i], clusterCentres)[1]
        clusterIdentifier.append(j)
    return clusterIdentifier

# This function changes the k means by recalculating them in a way
 #that
# this finds out the points belonging to the ith cluster(i=0,1,2,...k),
# stores them in a set and finds the mean of ith cluster and update the previ
def calcMeanCluster(dataset, clusterIdentifier, clusters):

    clusterMean = []
    for i in range(0,clusters):
        # temp. variable for storing datapoints belonging to ith
        cluster
        clusterSet=[]
        for j in range(0, len(clusterIdentifier)):
            # check if jth data point belong to ith cluster or not
            if(clusterIdentifier[j]==i):
                clusterSet.append(dataset[j])
        # if it belongs then add the point to the dataset temporarily
        #for mean calculation
        clusterMean.append(calcMean(clusterSet))
    return clusterMean

# here is our main function where the main logic is working
if __name__=='__main__':
    # First of all we read the data and store it in a matrix form
    csv_trainlabel = "train_data.csv"
    data = []
    with open(csv_trainlabel) as ifile:
        read = csv.reader(ifile)
        for row in read:
            data.append(strToInt(row))
    data = np.array(data)
    # Means are initialised. here these choices are made after a lot of brute
        #force
    clusterCentres = [[-1.82517034, -0.40389968], [1.10371788,
```

```
 0.58891785], [0.86267684 ,-1.1656794], [-1.93101753, 0.77641046],
[1.06681984 , 0.46228307], [-0.66701728, 0.66295488]];

    # saved the parameters in a model
    m = [clusterCentres]
    save_model('kmeans.pkl',m)

    # This stores the cluster no. to which the data points belong
    clusterIdentifiers = kMeansCluster(data,clusterCentres)
    print(clusterIdentifiers)

    # iteration starts here
    i=0
    while(i<50):
        print(i)
        # first we calcuate new means with help of the cluster no.
        # assigned to the data points
        clusterCentres = calcMeanCluster(data, clusterIdentifiers,
clusters)
        # With these new means we again calculate the new cluster
        #no. for each datapoint which is
        # closest to it.
        clusterIdentifiers = kMeansCluster(data,clusterCentres)
        print(clusterIdentifiers)
        i=i+1

    # Finally The output calculated are stored into a csv file
    with open('Kmeans.csv', 'w', encoding='utf8', newline='') as
        svfile:
        spmwriter = csv.writer(svfile)
        for i in range(0, len(clusterIdentifiers)):
            spmwriter.writerow([clusterIdentifiers[i]])
    print(clusterIdentifiers)
```

Code for Gaussian Mixture Models Algorithm

```
# import of required libraries
import numpy as np
import csv
import math

# Assumed no. of clusters to be 6. x123 is the cluster no. assigned by K
#means algorithm
clusters = 6
x123 = [3,3,3,3,2,3,3,3,3,3,3,3,3,3,3,3,4,4,3,3,3,0,3,3,3,4,3,3,3,3,3,2,3,3,3
,0,3,4,3,3,3,3,3,3,0,0,3,3,3,3,3,0,3,3,4,3,3,3,0,3,0,4,0,3,2,3,3,3,2,3,3,2,3,
4,4,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,2,3,2,3,3,3,3,3,0,3,3,3,3,3,3,3,3,4,0
3,3,3,3,4,3,0,3,0,3,3,4,0,3,3,3,0,3,3,4,3,3,3,0,3,3,4,3,3,3,3,3,3,2,3,0,3,4,2
3,3,3,3,3,0,0,2,3,3,3,3,0,3,3,3,3,3,3,4,3,3,3,3,3,3,3,3,3,3,3,3,3,0,3,3,3,3,4
3,4,0,3,3,3,0,3,3,2,3,4,2,3,3,3,3,2,0,3,3,3,3,0,2,3,3,3,0,3,2,3,2,3,2,0,3,0,3
3,4,3,3,3,0,3,3,3,2,3,3,0,3,3,3,3,3,2,3,3,3,4,3,3,4,2,3,4,3,2,3,0,4,3,3,3,0,3
3,3,3,3,3,3,3,3,4,3,3,3,3,3,3,0,3,3,4,3,3,2,3,3,4,0,4,3,3,0,3,0,3,3,0,4,0,3,0
3,3,3,3,3,3,3,3,3,3,3,3,3,2,4,3,0,3,0,4,2,3,3,3,3,3,0,3,3,3,3,0,2,3,3,3,0,3,3,3,0,3
3,4,3,2,3,3,3,3,3,3,3,3,0,2,3,2,3,3,2,3,4,4,3,0,3,3,3,3,2,3,3,0,0,0,3,3,2,3,3,3
3,0,2,3,3,3,0,3,3,2,3,3,0,3,3,4,3,3,4,3,3,3,3,3,4,2,3,3,3,3,3,3,3,3,3,3,0,4,3
3,3,3,0,0,3,3,0,3,3,3,3,3,2,3,3,3,0,3,3,3,3,3,3,3,3,3,3,0,3,2,4,3,3,3,2,3,3,3,3
3,3,3,3,0,3,0,3,4,2,3,3,3,4,2,3,3,2,3,3,3,0,2,3,4,3,3,3,3,3,2,4,2,3,3,0,3,3,3
3,3,4,0,0,2,3,3,3,3,3,0,2,2,2,3,3,0,4,3,0,3,3,3,0,3,0,3,3,3,3,2,3,0,3,3,3,3,3
3,3,3,3,2,3,3,2,3,0,3,3,3,3,3,3,3,4,3,3,3,4,0,3,3,3,3,3,3,3,3,2,3,0,3,3,3,3,2
2,0,4,0,0,3,3,3,4,0,0,3,3,3,3,3,3,0,2,4,3,3,3,3,3,3,3,2,3,3,3,3,3,3,2,3,3,4,3,0,3
3,3,3,4,3,3,3,0,3,2,3,3,0,4,3,3,3,3,0,4,2,3,3,3,3,3,3,3,3,3,3,0,3,2,3,3,3,3,3
3,0,3,3,4,3,3,2,3,3,0,3,3,3,3,3,3,3,3,3,3,3,0,3,3,3,3,3,3,4,3,3,4,3,3,3,3,3,3,0
3,0,0,3,3,3,4,2,4,3,3,2,3,3,3,4,2,0,3,0,3,3,4,3,0,3,3,3,3,3,3,3,3,3,0,4,3,3,3,0
2,3,3,3,3,3,3,3,3,3,2,3,2,3,0,3,3,3,4,3,3,3,3,2,3,4,3,3,3,3,3,3,0,3,4,3,3,3,2,0,3
2,3,2,3,3,0,0,3,3,3,3,3,3]

# This saves model in a pickel file
def save_model(file_name, model_list):
    import pickle
    with open(file_name, 'wb') as fid:
        pickle.dump(model_list, fid)

# Loading the model from the pickle file
def load_model(file_name):
    import pickle
```

```
    with open(file_name, 'rb') as fid:
        model = pickle.load(fid)
    return model

# Basic function to convert string into a int array
# since the data points are given in string form this
# is used to convert them into an integer array(matrix)
def strToInt(row):
    arr=[]
    arr.append(float(row[0]))
    arr.append(float(row[1]))
    arr = np.array(arr)
    return (arr)

# This function calculates probability of a datapoint to lie in a cluster
# The probability is calculated using normal distribution function.
# Since this is
# a multivariate function covariance matrix is used
def calcProb(data, means, covariance):
    # converted variables into numpy matrixes for multiplication
    data = np.array(data)
    means = np.array(means)
    covariance = np.array(covariance)
    p = data-means
    p1 = np.matrix(p)
    # the exponent of e is calculated and stored in a variable y
    y1 = np.dot(np.linalg.inv(covariance),p1.transpose())
    y = np.dot(p,y1)
    # constant factor is calculated
    norm = 2*(math.pi)*(math.sqrt(abs(np.linalg.det(covariance))))
    prob = (1/norm)*math.exp((-0.5)*(y))
    return prob

# This function calculates the maximum probability of a datapoint to lie
#in k clusters
# by calculating probability for every cluster and then comparing to find
# the maximum of them
def maxProb(data, means, covariances, mixingCoeffs):
    # array for storing probabilities
    probs = []
```

```
    # temp variable for maximum probability storing
    maxP = -math.inf

    # probabilities are calculated and compared here
    for i in range(0,clusters):
        x =  (mixingCoeffs[i])*calcProb(data, means[i], covariances[i])/790
        probs.append(x)
        if(x>maxP):
            maxP = x
    return [maxP,probs.index(maxP)]

# This calculates the no. of times 'n' comes in an array 'arr'
def calcMix(n,arr):
    count=0
    for i in range(len(arr)):
        if(arr[i]==n):
            count=count+1
    return count

# this function calculates the mean of a data given by using all the
#data points contained in it
def calcMean(data):
    # if data is empty mean is returned to be (0,0)
    if(len(data)==0):
        return np.array([0.,0.])
    # temporary variable for storing mean
    x = np.array([0.,0.])
    # every point is added to the temporary variable
    for i in range(0,len(data)):
        x = x + data[i]
    # finally it is divided by no. of points in the set
    x = x/len(data)
    return x

# This function changes the k means by recalculating them in a way that
# this finds out the points belonging to the ith cluster(i=0,1,2,...k),
# stores them in a set and finds the mean of ith cluster and update the
#previous mean
def calcMeanCluster(dataset, clusterIdentifier):
```

```
    clusterMean = []
    for i in range(0,clusters):
        # temp. variable for storing datapoints belonging to ith cluster
        clusterSet=[]
        for j in range(0, len(clusterIdentifier)):
            # check if jth data point belong to ith cluster or not
            if(clusterIdentifier[j]==i):
                clusterSet.append(dataset[j])
        # if it belongs then add the point to the dataset temporarily for
        # mean calculation
        clusterMean.append(calcMean(clusterSet))

    return clusterMean

# This function calculates the maximum probability of every data point from
# each of k centres and assigns the data point a cluster from whose
# probability is highest
def GaussMixtureMeans(datset, means ,covarianes, mixingCoeffs):
    # Array for storing indexes of max probability cluster
    clusterIdentifier = []
    for i in range(0,len(datset)):
        j = maxProb(datset[i], means, covarianes, mixingCoeffs)[1]
        clusterIdentifier.append(j)
    return clusterIdentifier

# here is our main function where the main logic is working
if __name__=='__main__':

    # First of all we read the data and store it in a matrix form
    csv_trainlabel = "train_data.csv"
    data = []
    with open(csv_trainlabel) as ifile:
        read = csv.reader(ifile)
        for row in read:
            data.append(strToInt(row))

    # Means are initialised. here these choices are made after a lot of
        #brute force and
    #  finally these the ones given by k means clustering
```

```
    clusterMeans = np.array([[-0.92517034, -0.90389968],[1.00371788,
        -0.58891785],[1.356267684, 1.2556794],[2.6501753 ,2.077641046]
        ,[1.46681984,  3.66228307],[-0.66701728, 0.66295488]])

    # for initialising covariance matrix it is done by calculating
        #covariances using the assignment of clusters by k means algorithm
    v = []
    for j in range(0, clusters):
        # no. of points in jth cluster are calculated
        n = calcMix(j, x123)
        if (n == 0):
            v.append(np.array([[1., 0.], [0., 1.]]))
        else:
            # for every point it is checked whether it belongs to jth
        #cluster or not
            o = np.array([[0., 0.], [0., 0.]])
            number = 0
            for k in range(0, len(x123)):
                # if yes the covariance is calculated and added
                if (x123[k] == j):
                    number = number + 1
                    p = (data[k] - clusterMeans[j])
                    p = np.matrix(p)
                    o = o + np.dot(p.transpose(), p)
            o = o / number
            v.append(o)

    clusterCovariances =v
    mixingCoefficients = []

    # Mixing coefficient are initialsed as all are equal
    for i in range(0,clusters):
        mixingCoefficients.append(1/clusters)

    # Clusters are assigned using the means and covariance matrix calculated
    clusterIdentifiers = GaussMixtureMeans(data, clusterMeans,
        clusterCovariances, mixingCoefficients)

# Iterations are started
    i=0
```

```
# model is saved
m = [clusterMeans, clusterCovariances, mixingCoefficients]
save_model('model.pkl', m)

while(i<25):
    print(i)
    #
    #  new means are calculated with new identifiers of data
    clusterMeans = calcMeanCluster(data, clusterIdentifiers)

    # new covariance matrix are calculates using the same procedure
    #as above
    for j in range(0,clusters):
        n  = calcMix(j,clusterIdentifiers)
        if(n==0):
            clusterCovariances[j]=(np.array([[10., 0.], [0., 1.]]))
        else:
            x = np.array([[0., 0.], [0., 0.]])
            number = 0
            for k in range(0,len(clusterIdentifiers)):
                if(clusterIdentifiers[k]==j):
                    number = number+1
                    p = (data[k]-clusterMeans[j])
                    p = np.matrix(p)
                    x = x + np.dot(p.transpose(),p)
            x = x/number
            clusterCovariances[j]=x

    # new mixing coefficients are calculated
    for j in range(0,clusters):
        mixingCoefficients[j] = calcMix(j,clusterIdentifiers)

    # Finally the clusterIdentifiers are calculated using newly
    #found means and covariances
    clusterIdentifiers = GaussMixtureMeans(data, clusterMeans,
    #clusterCovariances, mixingCoefficients)
    i = i+1

# identifiers are stored in a csv file
with open('gauss.csv', 'w', encoding='utf8', newline='') as svfile:
```

```
spmwriter = csv.writer(svfile)
for i in range(0, len(clusterIdentifiers)):
    spmwriter.writerow([clusterIdentifiers[i]])
```

References

[1] Research paper. http://www.cse.iitm.ac.in/ vplab/courses/D-VP/PDF/gmm.pdf.

[2] Wikipedia. https://en.wikipedia.org/wiki/K-means_clustering.

[3] Youtube tutorial. https://www.youtube.com/watch?v=qMTuMa86NzU&t=514s.