ECEN 689 : Assignment 2

Rangeesh Venkatesan

UIN:527004482

The codes have been written from scratch (without using any predefined toolboxes/packages for classifiers) in Python. On request, the same can be provided to the professor for evaluation. Currently, snippets of the same are attached as and when deemed necessary

Question 1

Estimating the class conditional densities and finding the accuracy of the resulting Bayes Classifier

5 Random Samples Code

```
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
from scipy.stats import norm
from sys import maxsize
import math
```

```
a=np.loadtxt(open("P1a_train_data_2D.csv", "rb"), delimiter=",")
X=a[:,0:2]
```

```
c1=list(range(0,100))
c2=list(range(100,200))
X1_5=X[random.sample(c1,5)]
X2_5=X[random.sample(c2,5)]
X1_5_mean=np.mean(X1_5,axis=0)
X2_5_mean=np.mean(X2_5,axis=0)
X1_5_var=np.matmul(np.transpose(X1_5-np.ones((5,1))*X1_5_mean),(X1_5-
np.ones((5,1))*X1_5_mean))/5
X2_5_var=np.matmul(np.transpose(X2_5-np.ones((5,1))*X1_5_mean),(X2_5-
np.ones((5,1))*X2_5_mean))/5
X_5=np.concatenate((X1_5,X2_5))
X_5_mean=np.mean(X_5,axis=0)
X_5_var=np.matmul(np.transpose(X_5-np.ones((10,1))*X_5_mean),(X_5-
np.ones((10,1))*X_5_mean))/10
atest=np.loadtxt(open("P1a_test_data_2D.csv", "rb"), delimiter=",")
```

The above code generates the 5 samples for each class and calculates the mean and variances.

Below, the **Bayes classifier** is defined and the accuracy is calculated at the end.

```
b_scre=0
atest=np.loadtxt(open("P1a_test_data_2D.csv", "rb"), delimiter=",")
#X_mean1 = np.mean(a[0:100,0:2],axis=0)
#X_mean2 = np.mean(a[100:200,0:2],axis=0)
#X_var1 = np.cov(a[0:100,0],a[0:100,1]).tolist()
#X_var2 = np.cov(a[100:200,0],a[100:200,1]).tolist()
for j in range(200):

    X_test=atest[j,0:2] ##
    Y_test=atest[j,2] ##
    f1 = prob(X_test, X1_5_mean, X1_5_var,1)
    f_1 = prob(X_test, X2_5_mean, X2_5_var,1)
    if f1>f_1:
        b_clss=1
    else:
        b_clss=-1
    if Y_test==b_clss:
        b_scre=b_scre+1
b_scre/len(X_test)
```

Using numpy.argsort(), we can define the **Nearest Neighbour Classifier** and the same is coded below

```
scre=0
for j in range(200):

    amd=np.array([])
    X_test=atest[j,0:2] ##
    Y_test=atest[j,2] ##
    for i in range(10):
        amdp=np.matmul(np.matmul(X_test -
X_5[i],np.linalg.inv(X_5_var)),np.transpose(X_test - X_5[i]))
        amd=np.append(amd,amdp)
```

```
    ar=np.argsort(amd)
    flag=0
    for k in range(7):
        if ar[k]>4:
            flag=flag-1
        if ar[k]<5:
            flag=flag+1
    if flag>=0:
        clss=1
    else:
        clss=-1
    if Y_test==clss:
        scre=scre+1


print(scre/len(X_test))
```

The accuracies for the Bayes Classifier and the NN- Classifier are shown below (for **a**):

Estimator sample size	Bayes Classifier Accuracy	kNN Classifier Accuracy (k=5)	kNN Classifier Accuracy (k=7)
5	0.725	0.66	0.71
10	0.755	0.705	0.735
25	0.77	0.695	0.70
75	0.77	0.735	0.735

The accuracies for the Bayes Classifier and the NN- Classifier are shown below (for **b**):

Estimator sample size	Bayes Classifier Accuracy	kNN Classifier Accuracy (k=5)	kNN Classifier Accuracy (k=7)
5	0.685	0.80	0.835
10	0.95	0.955	0.94
25	0.975	0.97	0.965
75	0.98	0.97	0.975

Observations

1. As we increase the number of samples that we train with, we find that the accuracy of both the classifiers improves. Although the improvement seems to reduce as the number of samples increases.
2. Bayes classifier, in general, seems to be always having a higher accuracy value that the NN- Classifier (for both k = 5 and 7)
3. k-NN Classifiers: As we increase the value of k, we find that the accuracy in general improves.

Conclusions

1. More the number of samples, better is the classifier accuracy, irrespective of what classifier we use
2. More the neighbours we consider - more is the accuracy of the NN classifier - although this significantly increases computing time
3. More the sample size, better is the accuracy - Although this too seems to improve the computing time significantly
4. Therefore, it's a trade off between computation time and the level of accuracy that we want our model to achieve

---

Major functions and important codes used in the 2nd subdivision is listed below

Functions used in E - M Algorithm

```
def prob(X_test, mean, var, lam):
    p =lam*
pow(pow(2*math.pi,2)*np.linalg.det(var),-0.5)*pow(math.e,-0.5*np.matmul(np.matmul(X_test -
mean,np.linalg.inv(var)),np.transpose(X_test - mean)))
    #print(p)
    return p
```

Expectation Function

```
def expectation(dataFrame, parameters):
  for i in range(dataFrame.shape[0]):
    x = dataFrame['x'][i]
    y = dataFrame['y'][i]
    np.array([x,y])
    np.array([parameters['mu1'][0],parameters['mu1'][1]])
    np.array(parameters['lambda'][0])
    np.array([[parameters['sig1'][0][0],parameters['sig1'][0][1]], [parameters['sig1'][1]
[0],parameters['sig1'][1][1]]])

    p_cluster1 = prob(np.array([x,y]), np.array([parameters['mu1'][0],parameters['mu1']
[1]]) , np.array([[parameters['sig1'][0][0],parameters['sig1'][0][1]], [parameters['sig1']
[1][0],parameters['sig1'][1][1]]]), np.array(parameters['lambda'][0]) )
    p_cluster2 = prob(np.array([x,y]), [parameters['mu2'][0],parameters['mu2'][1]]
,np.array([[parameters['sig2'][0][0],parameters['sig2'][0][1]], [parameters['sig2'][1]
[0],parameters['sig2'][1][1]]]), parameters['lambda'][1] )
    if p_cluster1 > p_cluster2:
      dataFrame['label'][i] = 1
    else:
      dataFrame['label'][i] = 2
  return dataFrame
```

Maximization Function

```
def maximization(dataFrame, parameters):
  points_assigned_to_cluster1 = dataFrame[dataFrame['label'] == 1]
```

```
  #print('points_assigned_to_cluster1', points_assigned_to_cluster1)
  points_assigned_to_cluster2 = dataFrame[dataFrame['label'] == 2]
  #print('points_assigned_to_cluster2', points_assigned_to_cluster2)
  percent_assigned_to_cluster1 = len(points_assigned_to_cluster1) / float(len(dataFrame))
  percent_assigned_to_cluster2 = 1 - percent_assigned_to_cluster1
  parameters['lambda'] = [percent_assigned_to_cluster1, percent_assigned_to_cluster2 ]
  parameters['mu1'] = [points_assigned_to_cluster1['x'].mean(),
points_assigned_to_cluster1['y'].mean()]
  #print("parameters['mu1']",parameters['mu1'])
  parameters['mu2'] = [points_assigned_to_cluster2['x'].mean(),
points_assigned_to_cluster2['y'].mean()]
  parameters['sig1'] =
np.cov(points_assigned_to_cluster1['x'],points_assigned_to_cluster1['y']).tolist()
  #print("points_assigned_to_cluster1['x']\n")
  #print("parameters['sig1']\n",parameters['sig1'])
  parameters['sig2'] =
np.cov(points_assigned_to_cluster2['x'],points_assigned_to_cluster2['y']).tolist()
  return parameters,points_assigned_to_cluster1['x'],points_assigned_to_cluster1['y']
```

Distance Function - Used to calculate the distance between the points

```
def distance(old_params, new_params):
  dist = 0
  for param in ['mu1', 'mu2']:
    for i in range(len(old_params)):
      dist += (old_params[param][i] - new_params[param][i]) ** 2
  return dist ** 0.5
```

Looping until the parameters converge

```
shift = maxsize
epsilon = 0.01
iters = 0
df_copy = df.copy()

df_copy['label'] = map(lambda x: x+1, np.random.choice(2, len(df)))
params = pd.DataFrame(guess)

while shift > epsilon:
  iters += 1
  updated_labels = expectation(df_copy.copy(), params)
  updated_parameters,xx,yy = maximization(updated_labels, params.copy())
  shift = distance(params, updated_parameters)
  print("iteration {}, shift {}".format(iters, shift))
  df_copy = updated_labels
  params = updated_parameters
  print(params)
  params
```

DATASET B	Given Parameters	Bayes learnt using EM	Bayes learnt using Class label
Mean (Class 1)	[0,0]	[-0.0866,-0.0853]	[-0.0023,-0.0115]
Mean (Class 1)	[3,3]	[2.9765,3.3037]	[3.0150,3.3424]
Covariance (Class 1)	[1, 0.5; 0.5 ,1]	[0.9297, 0.4399; 0.4399 ,0.8566]	[1.0723, 0.6153; 0.6153, 1.0601]
Covariance (Class 2)	[1, 0; 0 ,2]	[0.9874, -0.1219; -0.1219 ,2.8099]	[0.9812, -0.1608; -0.1608 ,2.8731]
Accuracy		97.5	97.5

Observation

1. Both the Bayes Classifiers compared here result in parameter values that are really close to the generated parameter values (nearly with 0.1 of the ideal values)
2. There seem to be more variation in the results for the Covariance matrix than the Mean values
3. There seems to be a more accurate result from the E-M algorithm than the Class Label Bayes, but the difference is too insignificant to be considered.

Conclusions

1. The E-M algorithm gives a fairly accurate value of the parameters and it's accuracy can be improved with a smaller distance value - although the computational time is bound to increase
2. The accuracy values of both the Bayes Classifiers seem to be the same and are pretty high as well. Therefore, we can conclude that the E-M algorithm is indeed an excellent way to estimate the two component mixture density parameters.

In part C, the Maximum Likelihood Estimate of 'lambda' was found out to be 1/mean.

Using the same, and assuming the features to be independent of each other, we can design the Bayes classifier as shown below:

```
b_scre=0
atest=np.loadtxt(open("P1c_test_data_2D.csv", "rb"), delimiter=",")
X_mean1 = np.mean(c[0:100,0:2],axis=0)
X_mean2 = np.mean(c[100:200,0:2],axis=0)
X_var1 = np.cov(c[0:100,0],c[0:100,1]).tolist()
lamc1=1/X_mean2[0]
lamc2=1/X_mean2[1]
for j in range(200):

    X_test=atest[j,0:2] ##
    Y_test=atest[j,2] ##
    f1 = prob(X_test, X_mean1, X_var1,1)
    f_1 = lamc1*lamc2*pow(math.e,-lamc1*X_test[0]-lamc2*X_test[1])
    if f1>f_1:
        b_clss=1
```

```
    else:
        b_clss=-1
    if Y_test==b_clss:
        b_scre=b_scre+1
b_scre/len(X_test)
```

The accuracy values for the above Bayes classifier and the ones with different sample sizes are shown below:

Type of Classifier	Accuracy
5 Sample Bayes	0.93
10 Sample Bayes	0.98
25 Sample Bayes	0.985
75 Sample Bayes	0.985
Normal + Exponent Bayes (Full Sample)	0.84
Bayes (Full Sample)	0.98

Observation

1. As the same size increases, the general trend is that the value of accuracy improves with the sample size, though not much at higher sample sizes.
2. The full Sample Normal and Exponential Bayes Classifier gives a lower accuracy than the full class label Bayes Classifier.

Conclusion

1. The Normal and Exponential Bayes Classifier isn't the best classifier around to accurately predict the classes.
2. A quarter or half Sample Size Bayes classifier is sufficient to give a really high value for the accuracy

---

Question 2

Below is the brief code for generating the Bayes and NN Classifier for 20 Dimensions

Generating Random Samples and finding mean and covariance matrix

```
a=np.loadtxt(open("P2a_train_data_20D.csv", "rb"), delimiter=",")
Y=a[:,20]
X=a[:,0:20]
X1_10=X[random.sample(c1,10)]
X2_10=X[random.sample(c2,10)]
X1_10_mean=np.mean(X1_10,axis=0)
X2_10_mean=np.mean(X2_10,axis=0)
X1_10_var=np.matmul(np.transpose(X1_10-np.ones((10,1))*X1_10_mean),(X1_10-
np.ones((10,1))*X1_10_mean))/10
X2_10_var=np.matmul(np.transpose(X2_10-np.ones((10,1))*X1_10_mean),(X2_10-
np.ones((10,1))*X2_10_mean))/10
X_10=np.concatenate((X1_10,X2_10))
X_10_mean=np.mean(X_10,axis=0)
X_10_var=np.matmul(np.transpose(X_10-np.ones((20,1))*X_10_mean),(X_10-
np.ones((20,1))*X_10_mean))/20
```

NN Classifier

```
atest=np.loadtxt(open("P2a_test_data_20D.csv", "rb"), delimiter=",")
scre=0
for j in range(1000):

    amd=np.array([])
    X_test=atest[j,0:20] ##
    Y_test=atest[j,20] ##
    for i in range(20):
        amdp=np.matmul(np.matmul(X_test -
X_10[i],np.linalg.inv(X_10_var)),np.transpose(X_test - X_10[i]))
        amd=np.append(amd,amdp)
    ar=np.argsort(amd)
    flag=0
    for k in range(7):
        if ar[k]>9:
            flag=flag-1
        if ar[k]<10:
            flag=flag+1
    if flag>=0:
        clss=1
    else:
        clss=-1
    if Y_test==clss:
        scre=scre+1


print(scre)
```

Bayes Classifier - the distance used here is the Mahalanobis Distance

```
b_scre=0
for j in range(1000):
```

```
    X_test=atest[j,0:20] ##
    Y_test=atest[j,20] ##
    f1 =
pow(pow(2*math.pi,20)*np.linalg.det(X1_10_var),-0.5)*pow(math.e,-0.5*np.matmul(np.matmul(X_
test - X1_10_mean,np.linalg.inv(X1_10_var)),np.transpose(X_test - X1_10_mean)))
    f_1 =
pow(pow(2*math.pi,20)*np.linalg.det(X2_10_var),-0.5)*pow(math.e,-0.5*np.matmul(np.matmul(X_
test - X2_10_mean,np.linalg.inv(X2_10_var)),np.transpose(X_test - X2_10_mean)))
    if f1>f_1:
        b_clss=1
    else:
        b_clss=-1
    if Y_test==b_clss:
        b_scre=b_scre+1
b_scre
```

Here are the results for Dataset 1:

Estimator sample size	Bayes Classifier Accuracy	kNN Classifier Accuracy (k=5)	kNN Classifier Accuracy (k=7)
10	0.543	0.56	0.574
50	0.848	0.783	0.812
100	0.974	0.836	0.855
300	0.982	0.875	0.903

Here are the results for Dataset 2:

Estimator sample size	Bayes Classifier Accuracy	kNN Classifier Accuracy (k=5)	kNN Classifier Accuracy (k=7)
10	0.500	0.498	0.461
50	0.806	0.751	0.767
100	0.823	0.787	0.779
300	0.873	0.785	0.802

Here are the results for Dataset 3:

Estimator sample size	Bayes Classifier Accuracy	kNN Classifier Accuracy (k=5)	kNN Classifier Accuracy (k=7)
10	0.500	0.519	0.531
50	0.987	0.890	0.888
100	0.991	0.918	0.931
300	0.997	0.961	0.969

Observation

1. The values of accuracy for all the classifiers increases as we increase the sample size
    - There's a steep increase in the beginning when then seems to flatten out as we reach higher values of the sample size
2. In general, the Bayes classifier has better accuracy than the kNN classifier, although at low sample sizes, the opposite is observed.
3. Changing the value of k from 5 to 7, shows us a general trend of increase in the value of accuracy, although it isn't the case at low sample sizes

Conclusion

1. From the above analysis, we can see that the Bayes classifier almost always provides us with a better value of accuracy than the kNN classifier and is expected to do so.
2. kNN classifier's accuracy is expected to increase with more k since, more the number of neighbours, more is the accuracy of correctly labelling a particular datapoint
3. More the sample size, better the accuracy.

Question 3

The Expectation Maximization Algorithm is similar to the one that has been used in Question 1. The parameters for the Mean and Standard Deviation of the 2 classes have been obtained using this algorithm and has been displayed in the tables below:

The Code of the **Classifiers** is given below:

```
params = { 'mu1': [-0.291311],
           'sig1': [1.782934] ,
           'mu2': [4.332953],
           'sig2': [1.77231],
           'lambda1': [0.44],
           'lambda2': [0.56]
         }
```

```
params1 = { 'mu1': [7.703065],
          'sig1': [1.662945] ,
          'mu2': [12.089492],
          'sig2': [1.293634],
          'lambda1': [0.56],
          'lambda2': [0.44]
        }
```

Single Gaussian Bayes Classifier

```
a=np.loadtxt(open("P3a_train_data.csv", "rb"))
xs=a[0:200,0]
labels=a[0:200,1]
data = {'x': xs, 'label': labels}
df = pd.DataFrame(data=data)
```

```
X1=a[0:100,0]
X2=a[100:200,0]
#Y1=a[0:100,0]
#X2=a[100:200,0]

X1_m=np.mean(X1)
X2_m=np.mean(X2)
X1_s=np.std(X1)
X2_s=np.std(X2)

print(X1_m,X1_s,X2_m,X2_s)

clss=[]
test=np.loadtxt(open("P3a_test_data.csv", "rb"))

xt=test[:,0]
yt=test[:,1]

flag=0
for i in range(len(xt)):
    f1=norm.pdf(xt[i],X1_m,X1_s)
    f2=norm.pdf(xt[i],X2_m,X2_s)
    #print(f1,f2)

    if f1>f2:
        clss.append(1)
    else:
        clss.append(-1)

    if clss[i]==yt[i]:
        flag=flag+1

print(flag/len(xt))
```

Mixture Gaussian Bayes Classifier

```
flag=0
clss1=[]
for i in range(len(xt)):
    f_1=params['lambda1'][0]*norm.pdf(xt[i],params['mu1'][0],params['sig1']
[0])+params['lambda2'][0]*norm.pdf(xt[i],params['mu2'][0],params['sig2'][0])
    f_2=params1['lambda1'][0]*norm.pdf(xt[i],params1['mu1'][0],params1['sig1']
[0])+params1['lambda2'][0]*norm.pdf(xt[i],params1['mu2'][0],params1['sig2'][0])
    #print(f_1,f_2)

    if f_1>f_2:
        clss1.append(1)
    else:
        clss1.append(-1)

    if clss1[i]==yt[i]:
        flag=flag+1

print(flag/len(xt))
```

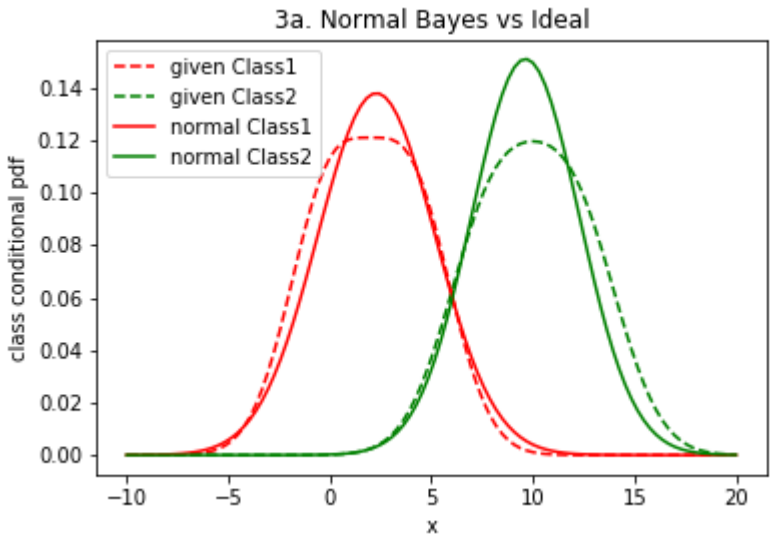
NN Classifier

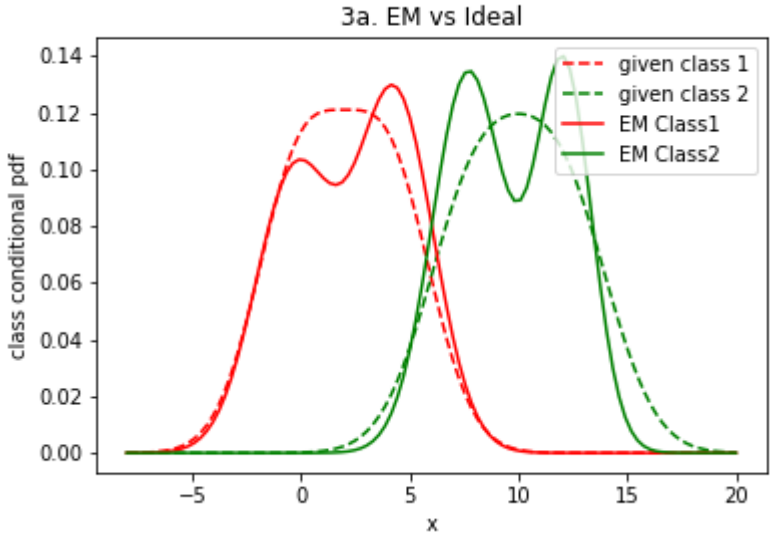
```
a=np.loadtxt(open("P3a_train_data.csv", "rb"))
Y=a[0:200,1]
X=a[0:200,0]
X1=a[0:100,0]
Y1=a[0:100,1]
X2=a[100:200,0]
Y2=a[100:200,1]

test=np.loadtxt(open("P3a_test_data.csv", "rb"))
Xt=test[:,0]
Yt=test[:,1]

ct=0
clss2=[]
for i in range(len(Xt)):
    d=abs(X-Xt[i])
    ar=np.argsort(d)
    flag=0
    for j in range(7):
        flag=flag+Y[ar[j]]

    if flag>0:
        clss2.append(1)
    else:
        clss2.append(-1)

    if clss2[i]==Yt[i]:
        ct=ct+1
ct/len(Xt)
```

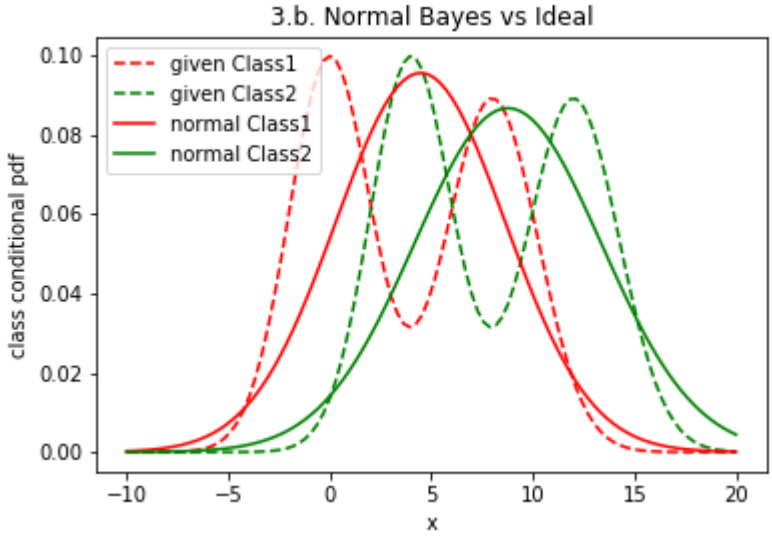
Sub-part	Mix Gaussian (EM Algorithm) Bayes Classifier Accuracy	Single Gaussian Bayes Classifier Accuracy	kNN Classifier Accuracy (k=5)
3(a)	0.925	0.925	0.905
3(b)	0.73	0.57	0.68

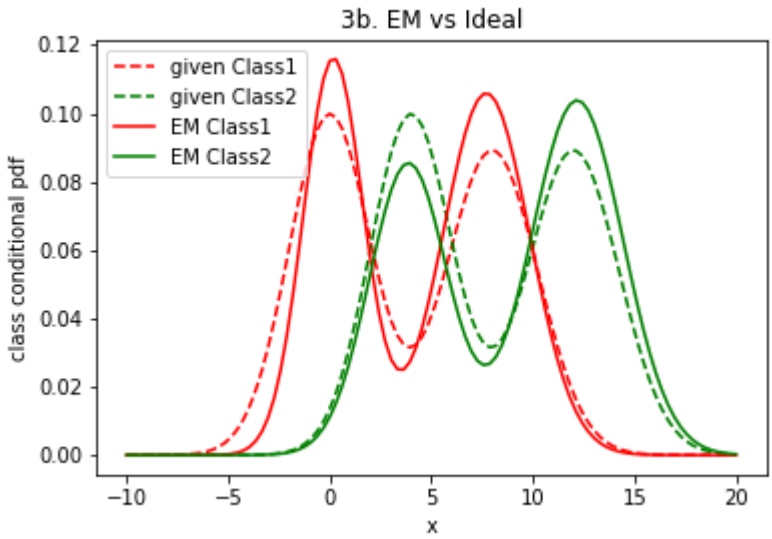
3(a)	Given Paramenters	EM	Bayes
Mean(Mixture Class 1)	[0,4]	-0.2913,4.3329	2.2983
St.Dev(Mixture Class 1)	2,2	1.7829,1.7723	2.8920
Lambda(Class 1)	0.5,0.5	0.44,0.56	-
Mean(Mixture Class 2)	8,12	7.7031,12.0895	9.6331
St.Dev(Mixture Class 2	2.236,2.236	1.6629,1.2936	2.6422
Lambda(Class 2)	0.5,0.5	0.56,0.44	-

3(a)	Given Paramenters	EM	Bayes
Mean(Mixture Class 1)	[0,4]	-0.2913,4.3329	2.2983
St.Dev(Mixture Class 1)	2,2	1.7829,1.7723	2.8920
Lambda(Class 1)	0.5,0.5	0.44,0.56	-
Mean(Mixture Class 2)	8,12	7.7031,12.0895	9.6331
St.Dev(Mixture Class 2	2.236,2.236	1.6629,1.2936	2.6422
Lambda(Class 2)	0.5,0.5	0.56,0.44	-


3a. Normal Bayes vs Ideal
given Class1
given Class2
normal Class1
normal Class2
class conditional pdf
x


3a. EM vs Ideal
given class 1
given class 2
EM Class1
EM Class2
class conditional pdf
x


3.b. Normal Bayes vs Ideal
given Class1
given Class2
normal Class1
normal Class2
class conditional pdf
x



Observation

1. The accuracy of the Single Gaussian Bayes Classifier seems to be nearly the same as the mixture Gaussian Bayes Classifier. The Mixture Gaussian Bayes Classifier has two peaks, while the single Gaussian Bayes classifier has just one.
2. Both the Bayes Classifiers have a better accuracy than the kNN classifier
3. Looking at the peaks above, we can see that the peaks are pretty far apart in the mixture Gaussian and hence the single gaussian doesn't capture the data well. This explains the poor accuracy in case (ii)
4. The peaks are overlapping here and hence the accuracy for the second case is lower than the first one

Conclusion

1. If the mixture Gaussian peaks are isolated, then the single Gaussian bayes classifier will have a low accuracy
2. Bayes classifier can perform better than the NN - classifier

---

Question 4

There are two features here

- bag-of-words
- TF-IDF

An assumption made here is that the features are independent and the Naive Bayes Classifier has been used here

There are two steps in **Sentiment Analysis**:

- Extracting features from the given text data(reviews)
- Applying the Classifier to the dataset Obtained

Here's a snippet of the code used to extract the features:

```
import collections, re
import numpy as np
import pandas
a=pandas.read_csv('sentiment_analysis.csv')
bagsofwords = [ collections.Counter(re.findall(r'\w+', txt))
            for txt in a['text'][:]]
sumbags = sum(bagsofwords, collections.Counter())
```

Scikit-learn packages were used for the logistic Regression and Naive Bayes Classifier

Classifier	Accuracy with TF-IDF method	Accuracy with bag-of-words method
Naive Bayes Classifier	0.719	0.702
Logistic Regression Classifier	0.810	0.852

Observation

- With both the methods, we can observe that the Logistic Regression Classifier performs better than the Naive Bayes Classifier.