IE406 Machine Learning

Lab Assignment - 4

Group 28

201801015: Shantanu Tyagi
201801076: Shivani Nandani
201801407: Pratvi Shah
201801408: Arkaprabha Baerjee

Question 1

Plot Mean Image of all the 10 digits.

Answer

In this section we consider the MNIST dataset from openml and split it into training and testing datasets. Then we find the mean image of each of the class from the training set.

Code

```
1
2  import numpy as np
3  import pandas as pd
4  from matplotlib import pyplot as plt
5  from sklearn.model_selection import train_test_split
6  from sklearn.datasets import fetch_openml
7
8  mnist = fetch_openml('mnist_784')
9  x = mnist.data
10 y = mnist.target
11 X_train , X_test, Y_train, Y_test = train_test_split(mnist.data,mnist.target,test_size = 0.1)
12 X_train , X_test = (X_train.to_numpy())/255 , (X_test.to_numpy())/255
13 Y_train  = Y_train.cat.codes
14 Y_train = Y_train.to_numpy()
15 Y_test = Y_test.cat.codes
16 Y_test = Y_test.to_numpy()
17 meanImgArray = []
18
19 for i in range(10):
20
21     tempArray = np.vstack(np.mean(X_train[np.where(Y_train==i)],axis=0))
22     meanImgArray.append(tempArray)
23     tempArray = []
24
25 fig,axes=plt.subplots(1,10,figsize=[50,5])
26 for i in range(10):
27     meanImgArray[i] = meanImgArray[i].reshape(28,28)
28     axes[i].imshow(meanImgArray[i],interpolation=None,cmap='gray')
29
30 plt.show()
```

Listing 1: Question 1

Result

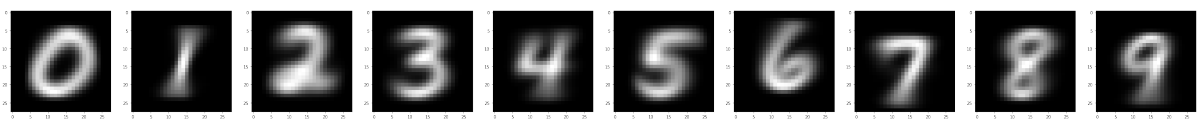


Figure 1: Mean Images

Question 2

Perform Linear Discriminant Analysis (LDA) on the MNIST dataset* for binary as well as for multiclass classification. Plot confusion matric and find out the combinations where the classifier is confused in predicting the right label.

Answer

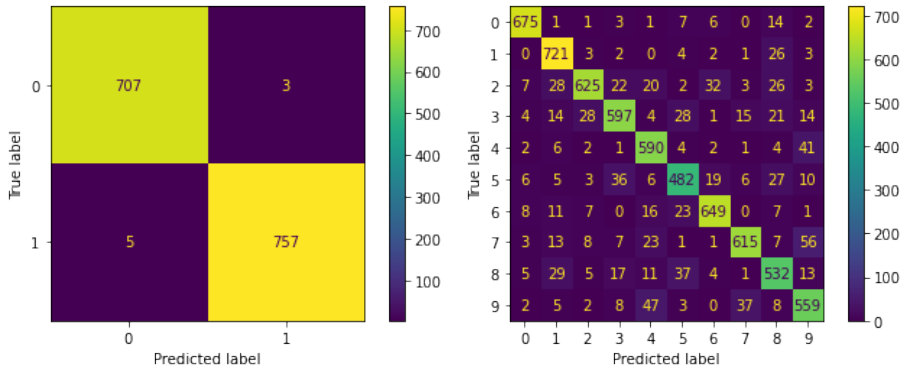
In this part we performed LDA on the MNIST Dataset using the inbuilt LinearDiscriminantAnalysis function of sklearn library. For binary classification we consider numbers 0 and 1. Also an svd solver has been employed to resolve the matrices.

Code

```
1
2  import numpy as np
3  import pandas as pd
4  from matplotlib import pyplot as plt
5  from sklearn.model_selection import train_test_split
6  from sklearn.datasets import fetch_openml
7
8  mnist = fetch_openml('mnist_784')
9  x = mnist.data
10 y = mnist.target
11 X_train , X_test, Y_train, Y_test = train_test_split(mnist.data,mnist.target,test_size = 0.1)
12 X_train , X_test = (X_train.to_numpy())/255 , (X_test.to_numpy())/255
13 Y_train  = Y_train.cat.codes
14 Y_train = Y_train.to_numpy()
15 Y_test = Y_test.cat.codes
16 Y_test = Y_test.to_numpy()
17
18 # Binary - b/w 0 and 1
19
20 train_data = X_train[np.where((Y_train==0) | (Y_train==1) )]
21 train_target =Y_train[np.where((Y_train==0) | (Y_train==1))]
22 test_data = X_test[np.where((Y_test==0) | ( Y_test==1))]
23 test_target =Y_test[np.where((Y_test==0) | ( Y_test==1))]
24 clf = LinearDiscriminantAnalysis()
25 clf.fit(train_data, train_target)
26 pred_data = clf.predict(test_data)
27 cm = confusion_matrix(test_target, pred_data)
28 disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=clf.classes_)
29 disp.plot()
30 plt.show()
31 print(classification_report(test_target,pred_data))
32
33 # Multiclass LDA
34
35 clf = LinearDiscriminantAnalysis()
36 clf.fit(X_train, Y_train)
37 pred_data = clf.predict(X_test)
38 cm = confusion_matrix(Y_test, pred_data)
39 disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=clf.classes_)
40 disp.plot()
41 plt.show()
42 print(classification_report(Y_test,pred_data))
```

Listing 2: Question 2

Result



(a) (b)

Figure 2: Confusion Matrix for LDA
a) Binary classification b)Multiclass classfication

	precision	recall	f1-score	support
0	0.99	1.00	0.99	710
1	1.00	0.99	0.99	762
accuracy			0.99	1472
macro avg	0.99	0.99	0.99	1472
weighted avg	0.99	0.99	0.99	1472

Figure 3: Accuracy metrics for binary classification

	precision	recall	f1-score	support
0	0.95	0.95	0.95	710
1	0.87	0.95	0.90	762
2	0.91	0.81	0.86	768
3	0.86	0.82	0.84	726
4	0.82	0.90	0.86	653
5	0.82	0.80	0.81	600
6	0.91	0.90	0.90	722
7	0.91	0.84	0.87	734
8	0.79	0.81	0.80	654
9	0.80	0.83	0.81	671
accuracy			0.86	7000
macro avg	0.86	0.86	0.86	7000
weighted avg	0.87	0.86	0.86	7000

Figure 4: Accuracy metrics for multiclass classification

Observation/ Justification

- In binary classification we obtain quite good results (99 %)
- In multi-class classification:
  1. Number 5 is predicted wrong the most number of times and is often confused with number 3.
  2. Number 1 is predicted right most of the time.
  3. Number 9 is often confused with number 4 and 7.
  4. Number 8 is often confused with number 5 and 2.
  5. Number 7 is often confused with number 9.
  6. Number 2 is often confused with number 6.

Question 3

Perform Quadratic Discriminant Analysis (QDA) on the MNIST dataset for multiclass classification. Plot confusion matrix and find out the combinations where the classifier is confused in predicting the right label.

Answer

In this part we performed QDA on the MNIST Dataset using the inbuilt QuadraticDiscriminantAnalysis function of sklearn library and our implementation too.

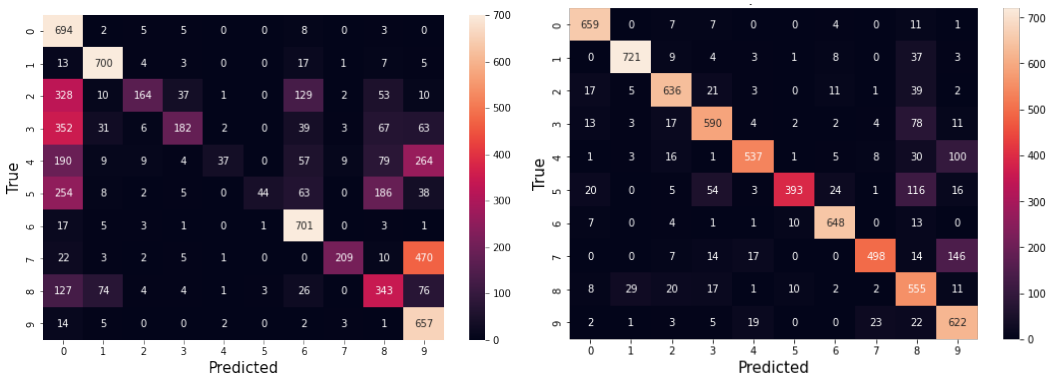
Code

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
# %matplotlib inline
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.model_selection  import  train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score
from sklearn import datasets
from sklearn.datasets import fetch_openml

# Load dataset
mnist = fetch_openml('mnist_784')
x = mnist.data
y = mnist.target

#Split training and testing data
x_train , x_test, Y_train, Y_test = train_test_split(mnist.data,mnist.target,test_size = 0.1)
y_train = np.array([int(Y_train[i]) for i in range(len(Y_train))])
y_test = np.array([int(Y_test[i]) for i in range(len(Y_test))])

# Implementing QDA
clf = QuadraticDiscriminantAnalysis()
clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
print('Accuracy: ',accuracy_score(y_test,y_pred))

#Plot confusion matrix
plt.figure(figsize=[9,6])
sns.heatmap(confusion_matrix( y_test,y_pred),annot=True, fmt='.4g')
plt.ylabel('True',fontsize=15)
plt.xlabel('Predicted',fontsize=15)
plt.title('Inbuilt Function',fontsize=15)

# Adding noise to remove the collinearity
for i in range(len(x_train)):
  x_train[i] = x_train[i] + np.random.rand(1,len(x_train[i]))*(1**-10)

# Implementing QDA
clf = QuadraticDiscriminantAnalysis()
clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
print('Accuracy: ',accuracy_score(y_test,y_pred))

# Our implementation
def qda(x,mu,pi,logdet,sigma_inverse):
  numerator = -np.inf
  y = 0
  for i in range(10):
    temp_numerator = -0.5*(np.dot(np.dot((x-mu[i]).T,sigma_inverse[i]),x-mu[i])) + np.log(pi[i
    ]) - 0.5*logdet[i]
    if temp_numerator>numerator:
      numerator = temp_numerator
      y=i
  return y

# Calculate necessary values
pi=[]
mu=[]
det = []
sigma_inverse = [[] for i in range(10)]
yy = [np.where(y_train==i) for i in range(10)]
for i in range(10):
    x = x_train[yy[i]]
    pi.append(np.size(yy[i])/np.size(y_train))
    mu.append(np.mean(x,axis=0))
    sigma_i = np.cov(np.array(x), rowvar = False)
    det.append(np.linalg.slogdet(sigma_i)[1]) #slogdet returns (det,log(det))
    sigma_inverse[i] = np.linalg.pinv(sigma_i)

# Prediction
y_pred_qda = []
for x in x_test:
```

```
  y_pred_qda.append(qda(x,mu,pi,det,sigma_inverse))
print('Accuracy of our function: ',accuracy_score(y_test,y_pred_qda))

# Plot confusion matrix of our function
plt.figure(figsize=[9,6])
sns.heatmap(confusion_matrix( y_test,y_pred_qda),annot=True, fmt='0.4g')
plt.ylabel('True',fontsize=15)
plt.xlabel('Predicted',fontsize=15)
plt.title('Our implementation',fontsize=15)
```

Listing 3: Quadratic Discriminant Analysis

Result



(a) (b)

Figure 5: Confusion Matrix for QDA
a) Inbuilt sklearn model b)Our Model

Observation/ Justification

From the above confusion matrices we can observe that the inbuilt classifier is confused in the following:

- 2, 3, 4, 5 and 8 are more frequently predicted to be 0
- 7 is wrongly predicted as 9 majority of the times
- 4 is also predicted as 9 more often than its correct value
- 2 is also predicted as 6 as frequently as it is predicted as 2

Digits 0, 1, 6 and 9 have some errors in prediction but they are not as significant as the previously listed ones. This is mainly because the variables possessed some collinearity but after we added some gaussian noise to the image data we could observe a significant increase in accuracy from 53% initially t 84%. This was verified using both inbuilt and our own implementation using *function qda()*. From, Fig 5(b) confusion matrix, we can also see the improved performance.

Question 4

Perform Naïve-Bayes on the MNIST dataset for multiclass classification. Plot confusion matrix and find out the combinations where the classifier is confused in predicting the right label.

Answer

In this part we used Naive-Bayes classification on the MNIST Dataset using the inbuilt MultinomialNB function of sklearn library.

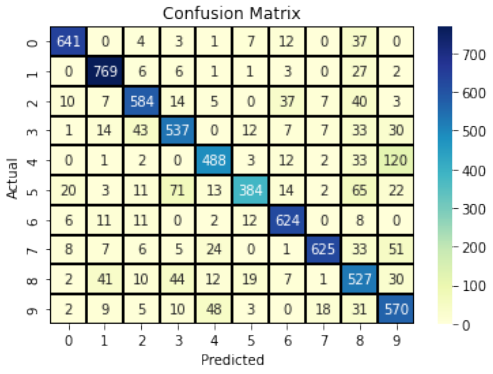
Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

```python
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import precision_score, recall_score, confusion_matrix,
    classification_report, accuracy_score, f1_score
import seaborn as sns

# load dataset
X, y = fetch_openml('mnist_784', return_X_y=True)
X = X / 255

# get classes
classes = np.sort(np.array(y.unique()))

# split into training and testing
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.1, random_state=0)
X_train.shape, X_test.shape

# naive-bayes classifier
clf = MultinomialNB()

# predict
y_pred = clf.fit(X_train, y_train).predict(X_test)

# printing classification report
print(classification_report(y_pred, y_test, target_names=classes))

# printing the confusion matrix
cm = confusion_matrix(y_test, y_pred)

ax = plt.axes()
sns.heatmap(pd.DataFrame(cm), annot=True, fmt='d', cmap='YlGnBu', ax=ax,
            xticklabels=classes, yticklabels=classes, linewidths=1.5, linecolor='black')
ax.set_title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

Listing 4: Naive-Bayes

Result



	precision	recall	f1-score	support
0	0.91	0.93	0.92	690
1	0.94	0.89	0.92	862
2	0.83	0.86	0.84	682
3	0.79	0.78	0.78	690
4	0.74	0.82	0.78	594
5	0.63	0.87	0.73	441
6	0.93	0.87	0.90	717
7	0.82	0.94	0.88	662
8	0.76	0.63	0.69	834
9	0.82	0.69	0.75	828
accuracy			0.82	7000
macro avg	0.82	0.83	0.82	7000
weighted avg	0.83	0.82	0.82	7000

Figure 6: Confusion Matrix and Classification Report for Naive-Bayes

Observation/ Justification

We get an accuracy of $\approx 83\%$.
From the above-given confusion matrix and classification report (6) we can observe that:

- 0 is wrongly predicted as 8 often
- 4 and 7 are wrongly predicted as 9 majority of the times
- 5 is wrongly predicted as 3 and 8 often
- 9 is wrongly predicted as 4 often
- 8 is often confused with 3

- 0 , 1 and 6 are least confused
- 0, 1, 2, 3, 4, 5, 6, 7, 9 are all wrongly predicted as 8

Question 5

Classify 50 samples each of three different normal distributions into two normally distributed classes using Naive-Bayes classifier having three apriory probabilities.

Answer

We have 50 samples from three different normal distribution and we need to classify them into two given normally distributed classes. We use Naive-Bayes classifier with varying apriory probability values to classify each of the 50 samples into the given two classes which can be observed from the colored scatter plots corresponding to the respective color class. Histograms are plotted to show the distribution of the 150 samples and the final classification is shown by a scatter plot and the merged probability distribution of both the classes which was used to classify the samples in the scatter plot.

Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Mean and variance of the two classes
class1_mu = 8
class1_sigma = np.sqrt(20)
class2_mu = 16
class2_sigma = np.sqrt(25)

# 50 random samples from N[5,20]
samples1 = np.random.normal(5,np.sqrt(20),50)
# 50 random samples from N[11,10]
samples2 = np.random.normal(11,np.sqrt(10),50)
# 50 random samples from N[20,8]
samples3 = np.random.normal(20,np.sqrt(8),50)

x = np.arange(-15,35,0.1)

plt.figure(figsize=(10, 6))
# histogram for N[5,20]
plt.hist(samples1, bins = 25, density=True, label='N[5,20] samples', alpha=0.5)
# histogram for N[11,10]
plt.hist(samples2, bins = 25, density=True, label='N[11,10] samples', alpha=0.5)
# histogram for N[20,8]
plt.hist(samples3, bins = 25, density=True, label='N[20,8] samples', alpha=0.5)

# PDF of class 1
pdf_class1 = (1/(class1_sigma*(np.sqrt(2*np.pi))))*np.exp(-(x-class1_mu)**2/(2*(class1_sigma
    **2)))
plt.plot(x, pdf_class1, label = 'Class 1: N[8,20]', linewidth = 2)
# PDF of class 2
pdf_class2 = (1/(class2_sigma*(np.sqrt(2*np.pi))))*np.exp(-(x-class2_mu)**2/(2*(class2_sigma
    **2)))
plt.plot(x, pdf_class2, label = 'Class 2: N[16,25]', linewidth = 2)

plt.grid()
plt.legend()
plt.show()

# Actual mean and SD for the samples
mu1 = np.mean(samples1)
sigma1 = np.std(samples1)
mu2 = np.mean(samples2)
sigma2 = np.std(samples2)
mu3 = np.mean(samples3)
sigma3 = np.std(samples3)

#classification function
def distribute(P1,P2,mu1,sigma1,mu2,sigma2,samples):
    n = len(samples)
```

```
    classes = [0]*n
    for i in range(n):
        sample = samples[i]

        p1 = (1/(sigma1*(np.sqrt(2*np.pi))))*np.exp(-(sample-mu1)**2 / (2*(sigma1**2)))
        p2 = (1/(sigma2*(np.sqrt(2*np.pi))))*np.exp(-(sample-mu2)**2 / (2*(sigma2**2)))

        classes[i] = 'b' if p1*P1 > p2*P2 else 'g'
    return classes

# sample dictionary
samples = {
    0:samples1,
    1:samples2,
    2:samples3
}
# appriory probability dictionary
P = {
    0:[0.5,0.5],
    1:[0.3,0.7],
    2:[0.7,0.3]
}
label = {
    0:"N[5,20]",
    1:"N[11,10]",
    2:"N[20,8]"
}

for j in range(3):
    P1 = P[j][0]
    P2 = P[j][1]

    for i in range(3):
    # classify
        classes = distribute(P1,P2,mu1,sigma1,mu2,sigma2,samples[i])

    # plot
        plt.plot(x,pdf_class1,'b-',label = 'class 1')
        plt.plot(x,pdf_class2,'g-',label = 'class 2')

    # merge
        pdf = P1*pdf_class1  + P2*pdf_class2
        plt.plot(x,pdf,'r-',label = 'pdf')

    # plot classification
        plt.scatter(samples[i],np.zeros_like(samples[i]),c = classes)
        plt.title("P = " + str(P[j]))
        plt.grid()
        plt.legend()
        plt.show()
```

Listing 5: Naive-Bayes classification into 2 normal classes

Result

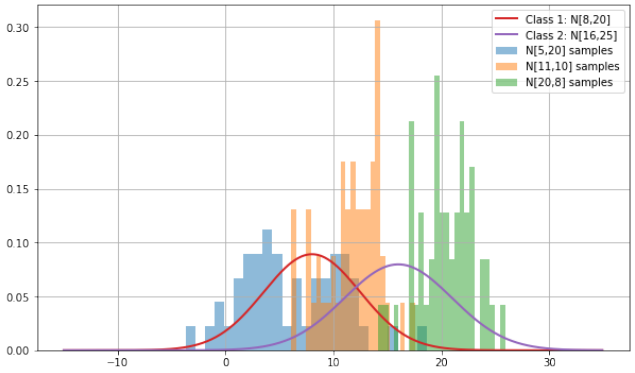


Figure 7: 50 random samples from 3 normal distributions and the 2 classes to be classified into

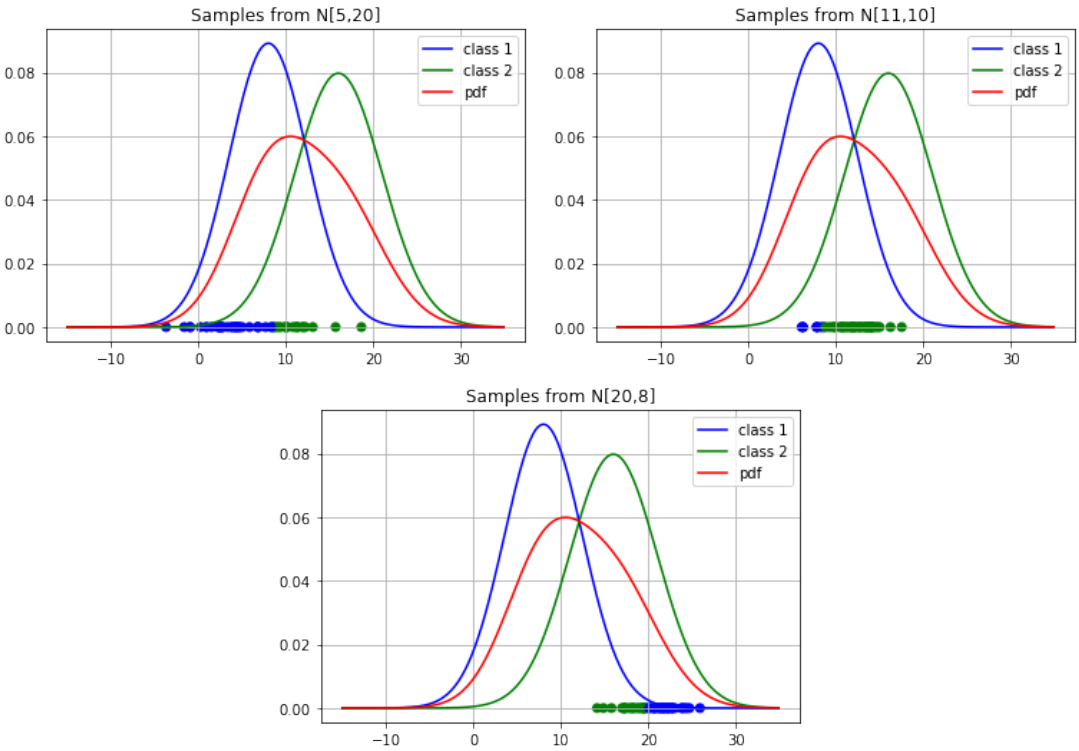


Figure 8: Appriory probability = (0.5,0.5)

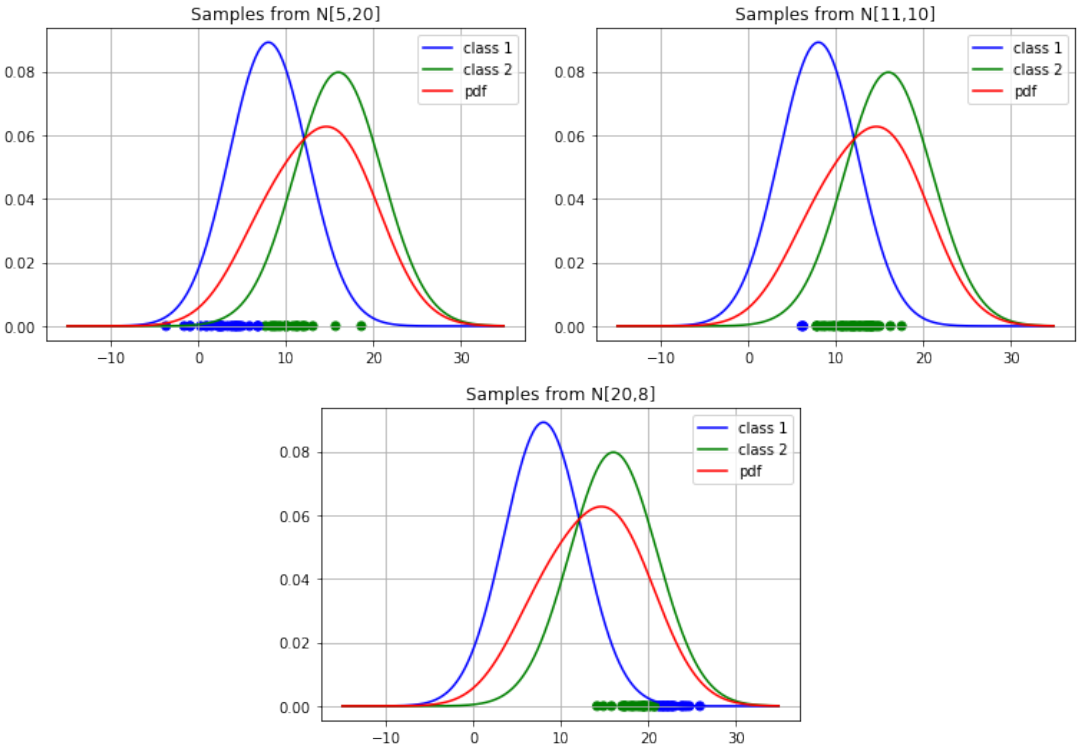


Figure 9: Appriory probability = (0.3,0.7)

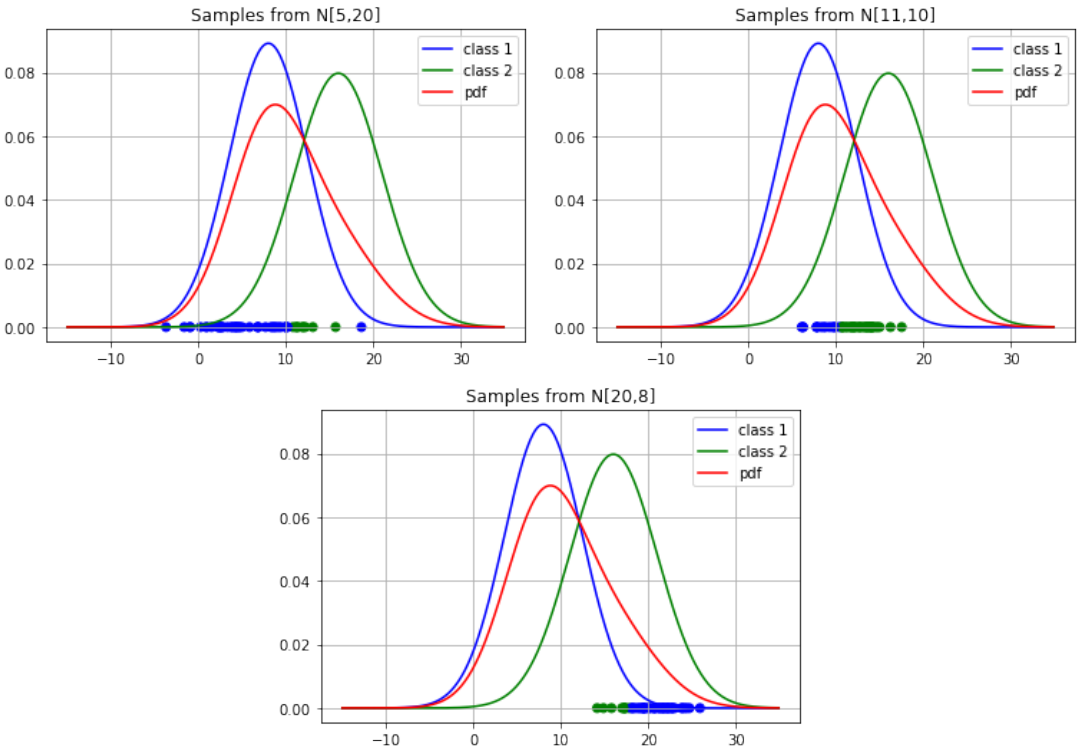


Figure 10: Appriory probability = (0.7,0.3)

Observation/ Justification

From the above-given confusion matrix and classification report (10) we can observe that:

- For a given normally distributed sample, as the apriory probability of a class increases the number of samples classified into that class also increases.
- The mean plays a significant role in deciding which class a particular sample will get classified into.