IE406 Machine Learning

Lab Assignment - 4

Group 39

201901201: Sharvil Sheth

201901216: Prabhav Shah

201901301: Mohil Desai

201901462: Dhaval Vaidya

201901464: Vishvesh Patel

Question 1

The MNIST database of handwritten digits, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on pre-processing and formatting. (hint : use scikit-learn library's "fetchmldata" to load dataset) Plot Mean Image of all the 10 digits.

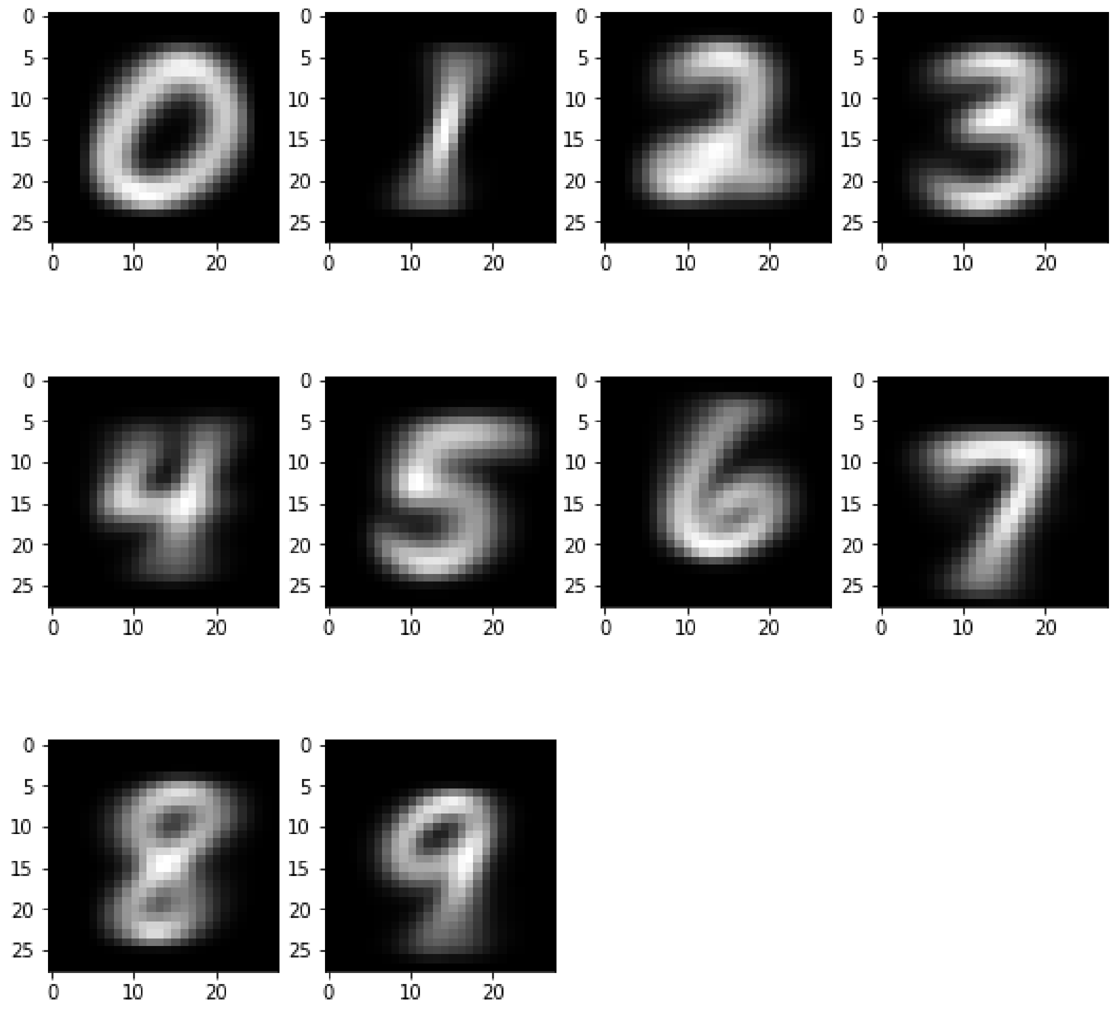
Answer

Code

[1]:
```
from sklearn.datasets import fetch_openml
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
mnist = fetch_openml('mnist_784')
X = mnist.data
Y = mnist.target
```

[3]:
```
fig, axs = plt.subplots(3, 4, figsize=(10,10))
j=0
for i, ax in enumerate(axs.flatten()):
    x = X[Y  ==  str(j)]
    j=j+1
    mean_images = np.array(np.mean(x, axis = 0))
    mean_images = mean_images.reshape(28,28)
    if i < 10:
        plt.gray()
        ax.imshow(mean_images)
    else:
        ax.remove()
plt.show()
```



[4]:
```
import numpy as np
import pprint
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
import seaborn as sns
from sklearn.metrics import confusion_matrix
```

[5]:
```
def covv(x1,x2):
 m1 = np.sum(x1)/len(x1)
 m2 = np.sum(x2)/len(x2)
 temp = (x1-m1)*(x2-m2)
 return np.sum(temp)/len(temp)

# Check if matrix is singular or not
# If matrix is singular then add some noise to it
def sing(c):
 if(np.linalg.det(c) == 0):
  noise = np.random.normal(0,0.000000000001,len(c)**2)
  noise = noise.reshape(len(c),len(c))
  c = c + noise
 return c

# Calculating delta for LDA
def LDA(x,m,c,p):
 t = x - m
 c = np.linalg.inv(c)
 return ( (-1/2)*(np.dot( np.dot(t.T,c) , t)) + np.log(p) )

# Calculating delta for QDA
def QDA(x,m,c,p):
 t = x - m
 d = np.log(abs(np.linalg.det(c)))
 c = np.linalg.inv(c)
 return (-1/2)*( d + (np.dot( np.dot(t.T,c) , t)) ) + np.log(p)
```

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

Code

Binary Classification

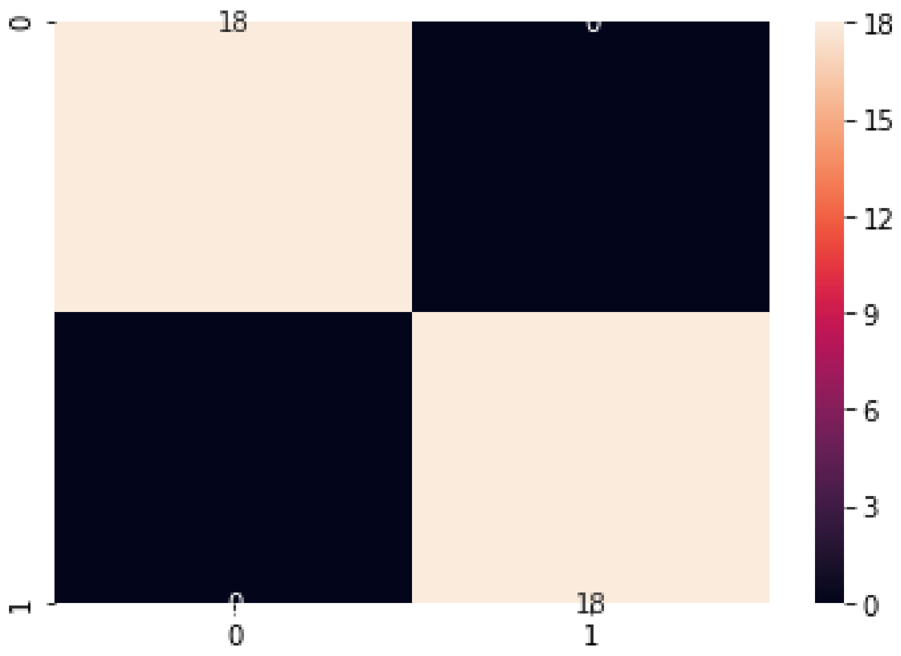
[6]:
```
################## Training ######################
digits = load_digits()
X = digits.data
y = digits.target

# Select images of 0 and 1 only
X1 = X[( y  ==  0)]
y1 = y[( y  ==  0)]
X2 = X[( y  ==  1)]
y2 = y[( y  ==  1)]
```

```
# splitting data into test and train
X1, X_test1, y1, y_test1 = train_test_split(X1, y1, test_size = 0.1,␣
 ↪random_state = 42)

X2, X_test2, y2, y_test2 = train_test_split(X2, y2, test_size = 0.1,␣
 ↪random_state = 42)

X1 = np.transpose(X1)
X2 = np.transpose(X2)

# making covariance matrix for both the digits
covM1  = []
for i in X1:
  t = []
  for j in X1:
    t.append(covv(i,j))
  covM1.append(t)

covM2  = []
for i in X2:
  t = []
  for j in X2:
    t.append(covv(i,j))
  covM2.append(t)

# To remove singularity
covM1 = sing(covM1)
covM2 = sing(covM2)

# Average
covM = (covM1 + covM2)/2

# Inverse of the covariance matrix
covInv = np.linalg.inv(covM)

# calculating mean vector
meanVector1T = []
meanVector2T = []

s = len(X1[0]) #samples 1797
n = len(X1) #number of random variables 64

for i in range(n):
 meanVector1T.append(np.sum(X1[i])/s)

s = len(X2[0]) #samples 1797
n = len(X2)
```

```python
for i in range(n):
  meanVector2T.append(np.sum(X2[i])/s)

meanVector1 = np.transpose([meanVector1T])
meanVector2 = np.transpose([meanVector2T])
theta = np.dot(covInv ,(meanVector1-meanVector2))

# Prior probability of both will be 1/2 since there are equal number of samples
p0 = 1/2
apriory1 = 1/2
theta0 = np.log(apriory1/p0) - 1/2*( np.dot( np.dot(meanVector1T , covInv),␣
 ↪meanVector1 )
 - np.dot( np.dot(meanVector2T , covInv), meanVector2 ))

############### Testing ##################

# Predict value using decision boundary
yPredicted = []
yActual = []

# predicting the value
for i in range(len(X_test1)):
  z1 = np.dot(X_test1[i],theta) + theta0
  if(z1 > 0):
    pred = 0
  else:
    pred = 1
  yPredicted.append(pred)
  yActual.append(0)
  z2 = np.dot(X_test2[i],theta) + theta0
  if(z2 > 0):
    pred = 0
  else:
    pred = 1
  yPredicted.append(pred)
  yActual.append(1)

# Confusion matrix
cm = confusion_matrix(yActual, yPredicted)
sns.heatmap(cm, annot = True)
print('----------Binary Classification Report -------------\n')
labels = ['Class 0', 'Class 1']
print(classification_report(yActual, yPredicted, target_names = labels))
```

```
----------Binary Classification Report -------------

              precision    recall  f1-score   support
```

```
     Class 0       1.00      1.00      1.00        18
     Class 1       1.00      1.00      1.00        18

    accuracy                           1.00        36
   macro avg       1.00      1.00      1.00        36
weighted avg       1.00      1.00      1.00        36
```



For binary class, only 0 and 1 are used for classification from the mnist dataset.As, shapes of 0 and 1 are very different , so the accuracy will be high to classify correct class.Here,covariance used for both classes is the average of the covariancce matrix of both 0 and 1.

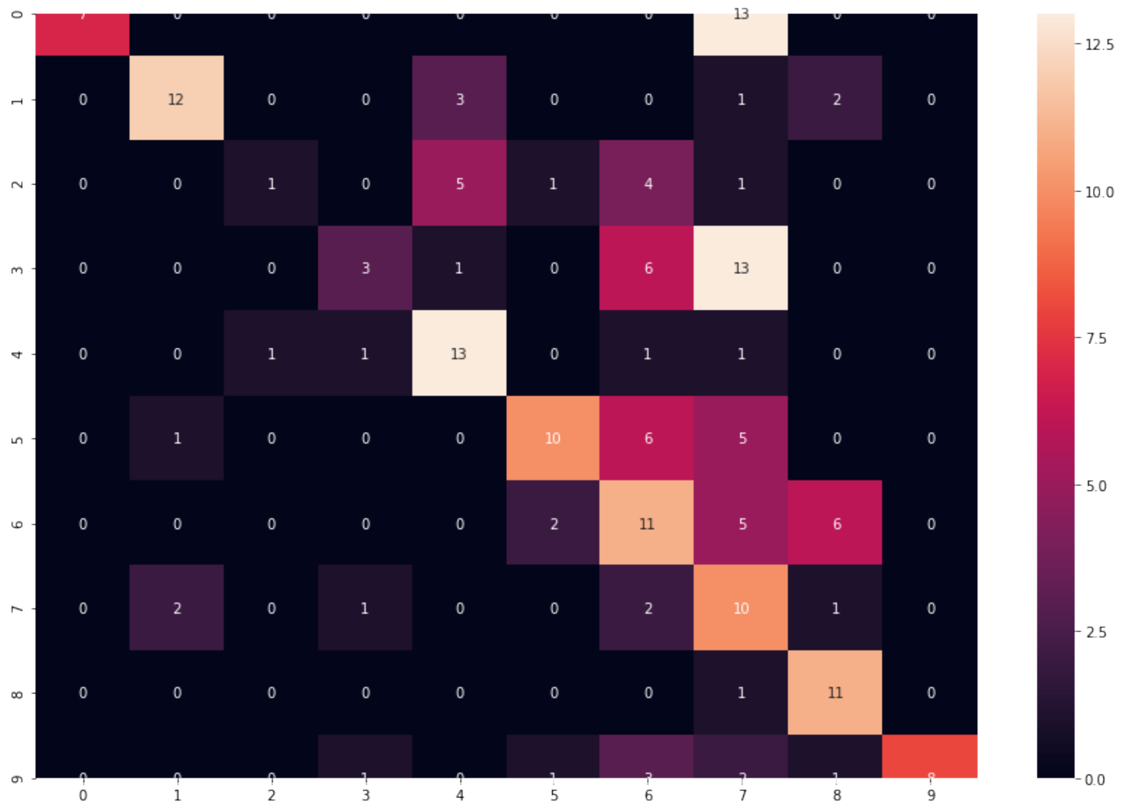
Multiclass classification

[7]:
```
################ Training ########################

# loading digits

digits = load_digits()
Xtemp = digits.data
ytemp = digits.target

# Number of classes
k = 10
```

```
# Splitting Train:test data into 9:1 ratio
Xtemp, X_test, ytemp, y_test = train_test_split(Xtemp, ytemp, test_size = 0.1)

# 3D arrays for storing data
X = []
total_samples = len(Xtemp)

for i in range(k):
 X.append( Xtemp[(ytemp == i)])

# Prior probability(1,10)
prior = []
for i in range(len(X)):
  prior.append(len(X[i])/total_samples)

meanVector = []

# mean vector: meanVector (10,64)
for i in range(len(X)):
  meanVectorT = []
  for j in range(len(X[0][0])):
    meanVectorT.append(np.sum(X[i][:,j])/len(X[0]))
  meanVector.append(meanVectorT)

# covariance matrix (10,64,64)
covM = []
for i in range(len(X)):
  covM1 = []
  for j in range(len(X[0][0])):
    t = []
    for l in range(len(X[0][0])):
        t.append(covv(X[i][:,j] , X[i][:,l] ))
    covM1.append(t)
  covM1 = sing(covM1)
  covM.append(covM1)
n  =  len(X[0][0])

# taking average of all the covariance matrices
covariance = [[0 for i in range(n)] for j in range(n)]
for i in covM:
  covariance +=  i
covariance /=  k

############### Testing ########################

yPredicted = []
yActual = []
```

```python
for i in range(len(y_test)):
  test = np.transpose([X_test[i]])
  delta = []
  for j in range(k):
    delta.append(LDA(test,np.transpose([meanVector[j]]),covM[j],prior[j]))
  yPredicted.append(delta.index(max(delta)))
  yActual.append(y_test[i])

# Confusion matrix
fig = plt.figure(figsize = (15,10))
cm = confusion_matrix(yActual, yPredicted)
sns.heatmap(cm, annot = True)
print('---------- Multiclass Classification Report -------------\n')
labels = ['Class 0', 'Class 1', 'Class 2', 'Class 3', 'Class 4', 'Class 5',␣
 ↪'Class 6',
'Class 7', 'Class 8', 'Class 9']
print(classification_report(yActual, yPredicted, target_names = labels))
```

```
---------- Multiclass Classification Report -------------

              precision    recall  f1-score   support

     Class 0       1.00      0.35      0.52        20
     Class 1       0.80      0.67      0.73        18
     Class 2       0.50      0.08      0.14        12
     Class 3       0.50      0.13      0.21        23
     Class 4       0.59      0.76      0.67        17
     Class 5       0.71      0.45      0.56        22
     Class 6       0.33      0.46      0.39        24
     Class 7       0.19      0.62      0.29        16
     Class 8       0.52      0.92      0.67        12
     Class 9       1.00      0.50      0.67        16

    accuracy                           0.48       180
   macro avg       0.62      0.49      0.48       180
weighted avg       0.62      0.48      0.48       180
```



Accuracy for LDA binary classification is higher than LDA multiclass classification.

Question 3

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

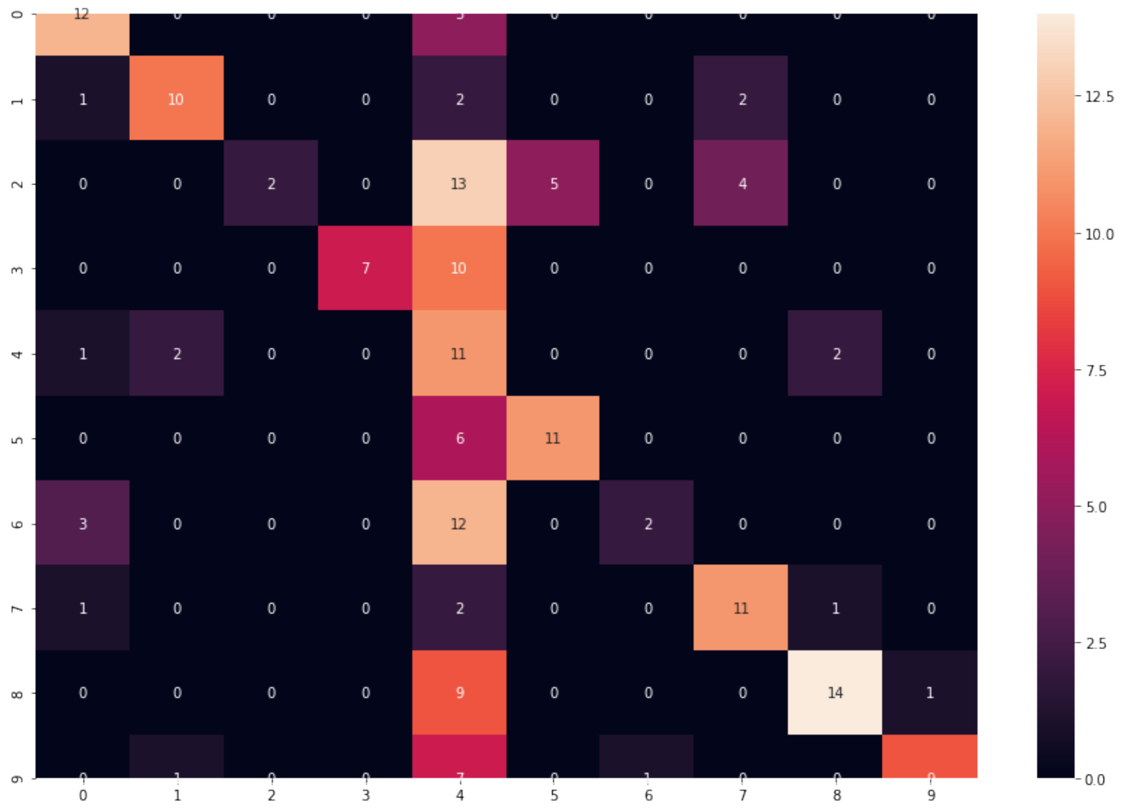
Answer

Code

[8]:
```
############# Training #########################

# loading the data
digits = load_digits()
Xtemp = digits.data
ytemp = digits.target
total_samples = len(Xtemp)

# total number of classes
k = 10
```

```
# splitting data into test and train
Xtemp, X_test, ytemp, y_test = train_test_split(Xtemp, ytemp, test_size = 0.1)

# 3D arrays for storing data
X = []
total_samples = len(Xtemp)

for i in range(k):
 X.append( Xtemp[( ytemp == i )])

# prior probabilitis
prior = []
for i in range(len(X)):
 prior.append(len(X[i])/total_samples)
meanVector = []

# mean vectors meanVector (10,64)
for i in range(len(X)):
  meanVectorT = []
  for j in range(len(X[0][0])):
    meanVectorT.append(np.sum(X[i][:,j])/len(X[0]))
  meanVector.append(meanVectorT)

# 3D covariance matrix
covM = []

# calculating covariance matrix for every class
for i in range(len(X)):
  covM1 = []
  for j in range(len(X[0][0])):
    t = []
    for l in range(len(X[0][0])):
      t.append(covv(X[i][:,j] , X[i][:,l] ))
    covM1.append(t)
  covM1 = sing(covM1)
  covM.append(covM1)

#number of random variables
n  =  len(X[0][0])

############## Testing ######################

yPredicted = []
yActual = []

for i in range(len(y_test)):
  test = np.transpose([X_test[i]])
  delta = []
  for j in range(k):
    delta.append(QDA(test,np.transpose([meanVector[j]]),covM[j],prior[j]))
  yPredicted.append(delta.index(max(delta)))
  yActual.append(y_test[i])

# Confusion Matrix
fig = plt.figure(figsize = (15,10))
cm = confusion_matrix(yActual, yPredicted)
sns.heatmap(cm, annot = True)
print('---------- Classification Report -------------\n')
labels = ['Class 0', 'Class 1', 'Class 2', 'Class 3', 'Class 4', 'Class 5',␣
 ↪'Class 6',
'Class 7', 'Class 8', 'Class 9']
print(classification_report(yActual, yPredicted, target_names = labels))
```

```
---------- Classification Report -------------

              precision    recall  f1-score   support

     Class 0       0.67      0.71      0.69        17
     Class 1       0.77      0.67      0.71        15
     Class 2       1.00      0.08      0.15        24
     Class 3       1.00      0.41      0.58        17
     Class 4       0.14      0.69      0.24        16
     Class 5       0.69      0.65      0.67        17
     Class 6       0.67      0.12      0.20        17
     Class 7       0.65      0.73      0.69        15
     Class 8       0.82      0.58      0.68        24
     Class 9       0.90      0.50      0.64        18

    accuracy                           0.49       180
   macro avg       0.73      0.51      0.53       180
weighted avg       0.75      0.49      0.52       180
```



QDA takes into account covariance matrix of the respective class. For both LDA and QDA the results obtained will vary with every simulation

Question 4

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

Answer

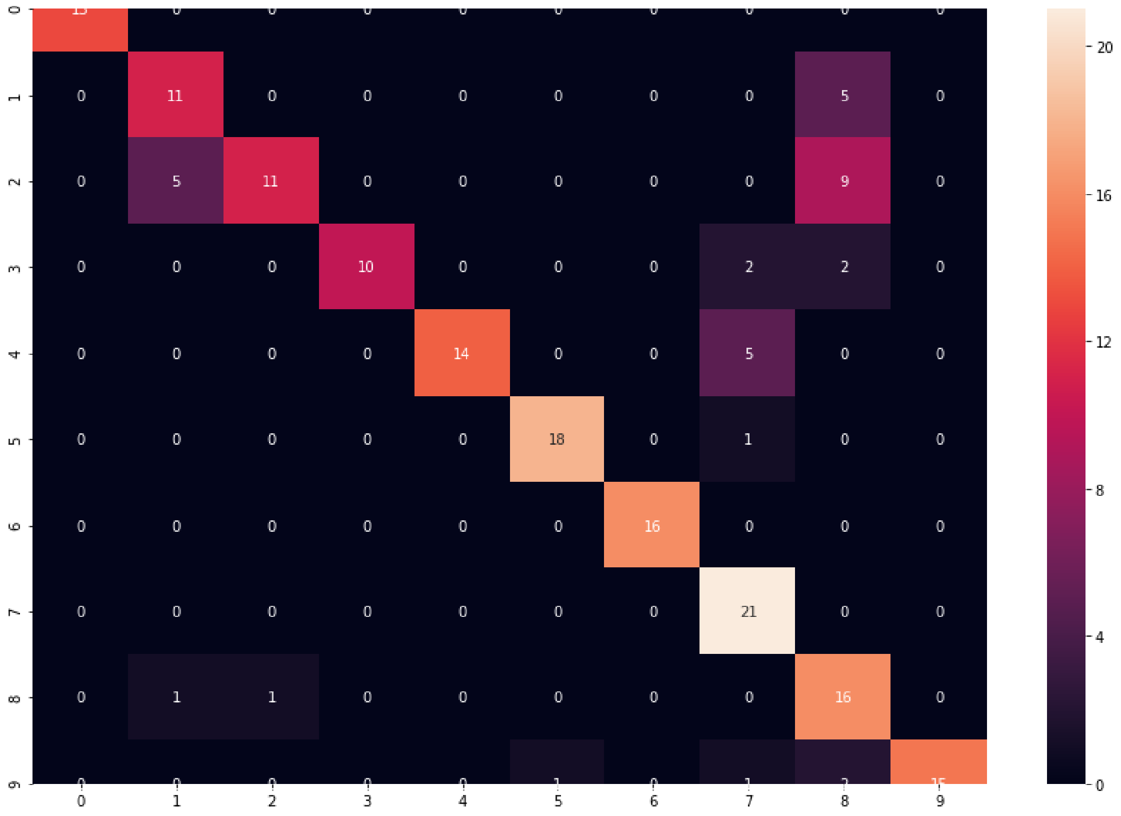
Code

[9]:
```
from sklearn.naive_bayes import GaussianNB
```

[10]:
```
############# Training #########################

# loading the data
digits = load_digits()
Xtemp = digits.data
ytemp = digits.target
total_samples = len(Xtemp)
```

```
# total number of classes
k = 10

# splitting data into test and train
Xtemp, X_test, ytemp, y_test = train_test_split(Xtemp, ytemp, test_size = 0.1)

# Using Naive-Bayes Classifier from sklearn
model = GaussianNB()
model.fit(Xtemp,ytemp)
yPredicted = model.predict(X_test)

# Confusion Matrix
fig = plt.figure(figsize = (15,10))
cm = confusion_matrix(y_test, yPredicted)
sns.heatmap(cm, annot = True)
print('---------- Classification Report -------------\n')
labels = ['Class 0', 'Class 1', 'Class 2', 'Class 3', 'Class 4', 'Class 5',␣
 ↪'Class 6',
'Class 7', 'Class 8', 'Class 9']
print(classification_report(y_test, yPredicted, target_names = labels))
```

```
---------- Classification Report -------------

              precision    recall  f1-score   support

     Class 0       1.00      1.00      1.00        13
     Class 1       0.65      0.69      0.67        16
     Class 2       0.92      0.44      0.59        25
     Class 3       1.00      0.71      0.83        14
     Class 4       1.00      0.74      0.85        19
     Class 5       0.95      0.95      0.95        19
     Class 6       1.00      1.00      1.00        16
     Class 7       0.70      1.00      0.82        21
     Class 8       0.47      0.89      0.62        18
     Class 9       1.00      0.79      0.88        19

    accuracy                           0.81       180
   macro avg       0.87      0.82      0.82       180
weighted avg       0.86      0.81      0.81       180
```



Question 5

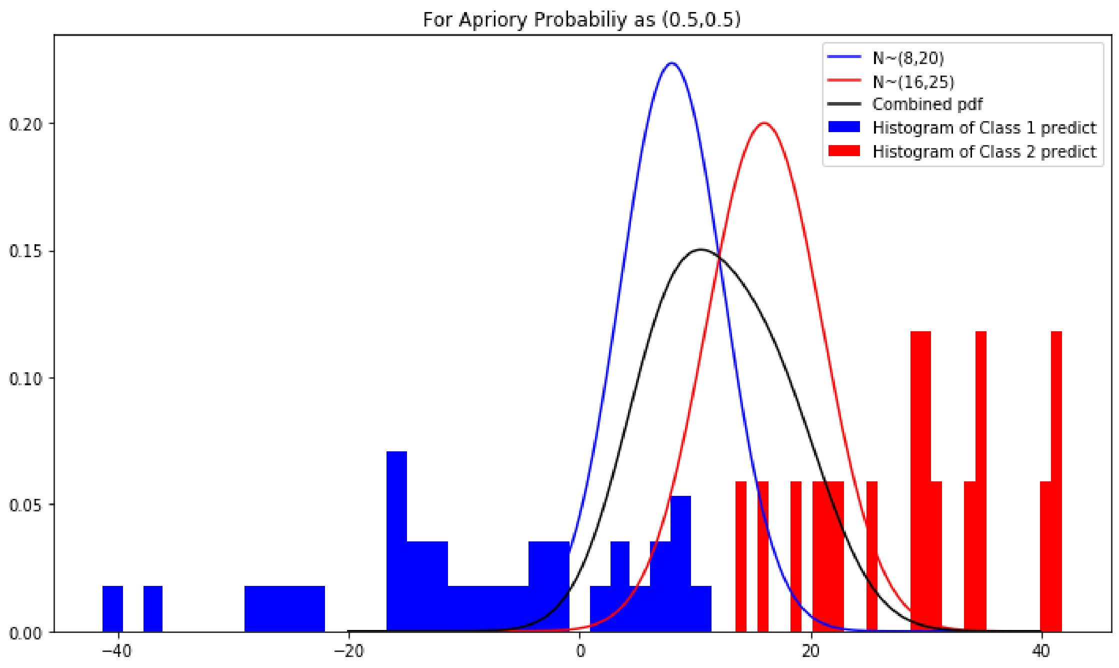
classify using Naïve-Bayes classifier having apriory probabilities as (0.5,0.5), (0.3,0.7) and (0.7,0.3) and visualize data and class by plotting histogram.

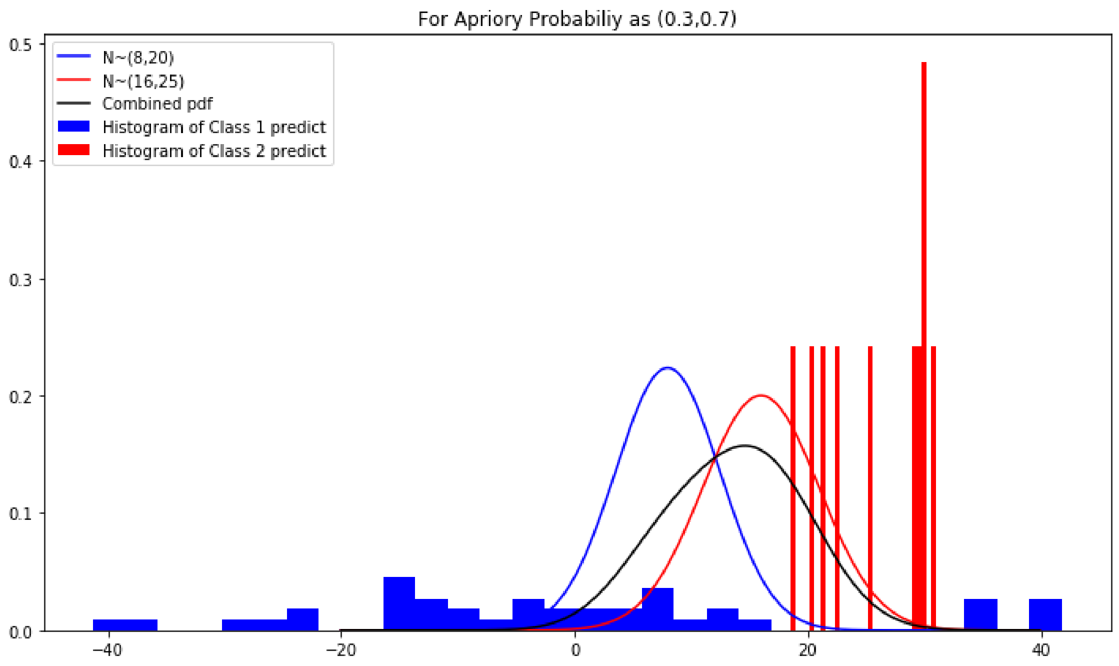
Answer

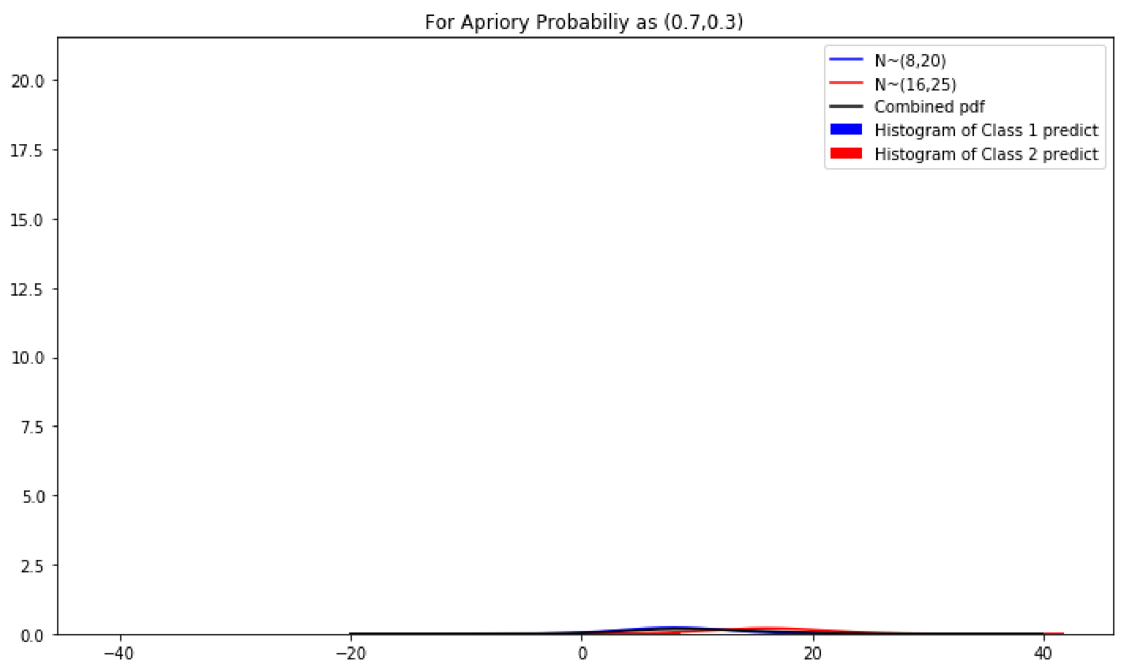
Code

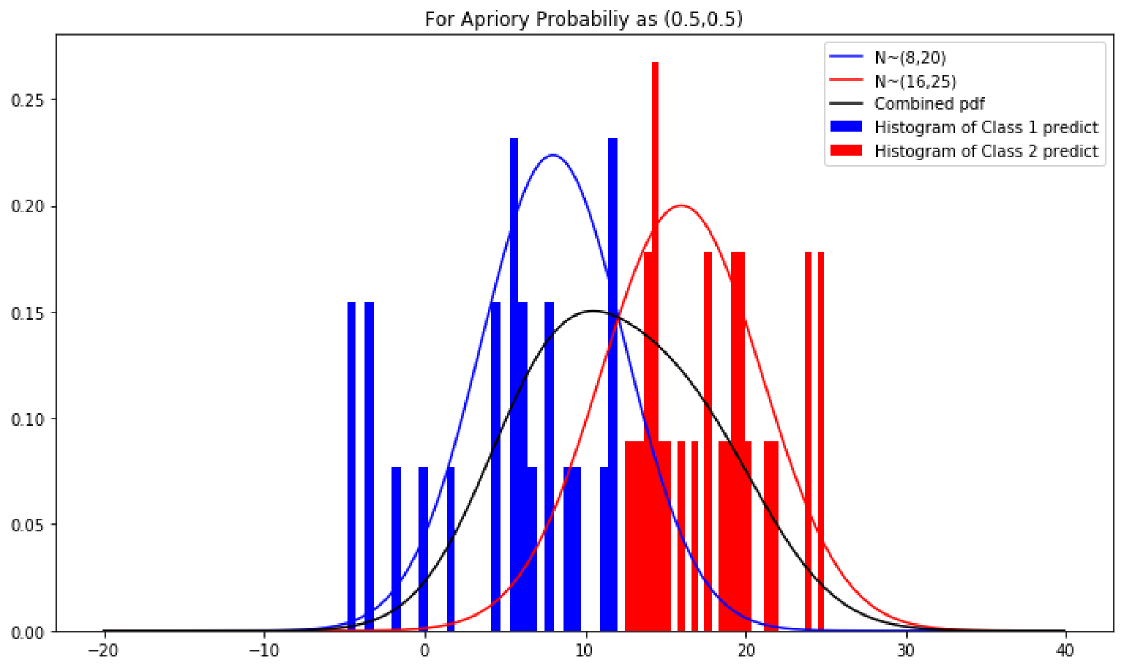
[11]:
```
import math
def plotHistograms(X, Y, apriory1, apriory2):
    x = np.arange(-20, 40, 0.1)

    pdf0 = ((1/math.sqrt(20))*np.exp((-1/2)*((x - 8)/math.sqrt(20))**2))
    pdf1 = ((1/math.sqrt(25))*np.exp((-1/2)*((x - 16)/math.sqrt(25))**2))

    plt.figure(figsize = (10, 6))

    plt.title(f"For Apriory Probabiliy as ({apriory1},{apriory2})")

    # Plotting histogram and pdf for class = 1
    plt.hist(X[Y==0], bins=30, density=True, color='blue', label='Histogram of␣
↪Class 1 predict')
    plt.plot(x, pdf0, 'b', label='N~(8,20)')

    # Plotting histogram and pdf for class = 2
    plt.hist(X[Y==1], bins=30, density=True, color='red', label='Histogram of␣
↪Class 2 predict')
    plt.plot(x, pdf1, 'r', label='N~(16,25)')

    pdf2 = apriory1*pdf0 + apriory2*pdf1
    plt.plot(x, pdf2, 'k', label='Combined pdf')

    plt.tight_layout()
    plt.legend()
    plt.show()

def predict(X_i, apriory1, apriory2):
    value1 = ((1/math.sqrt(20))*np.exp((-1/2)*((X_i - 8)/math.
↪sqrt(20))**2))**apriory1
    value2 = ((1/math.sqrt(25))*np.exp((-1/2)*((X_i - 16)/math.
↪sqrt(25))**2))**apriory2
    if value1 > value2:
        return 0
    else:
        return 1

def naive_bayes(X, apriory1, apriory2):
    Y = []
    for i in range(50):
        Y.append(predict(X[i], apriory1, apriory2))

    Y = np.array(Y)

    #Calling plot histogram function
    plotHistograms(X, Y, apriory1, apriory2)

def dataGen(mu, sigma):
    #Generating data for given mu and sigma
    X = np.random.normal(mu, sigma, 50)
    naive_bayes(X, 0.5, 0.5)
    naive_bayes(X, 0.3, 0.7)
    naive_bayes(X, 0.7, 0.3)

# dataset N[5,20]
dataGen(5, 20)
# dataset N[11,10]
dataGen(11, 10)
```

```
# dataset N[20,8]
dataGen(20, 8)
```

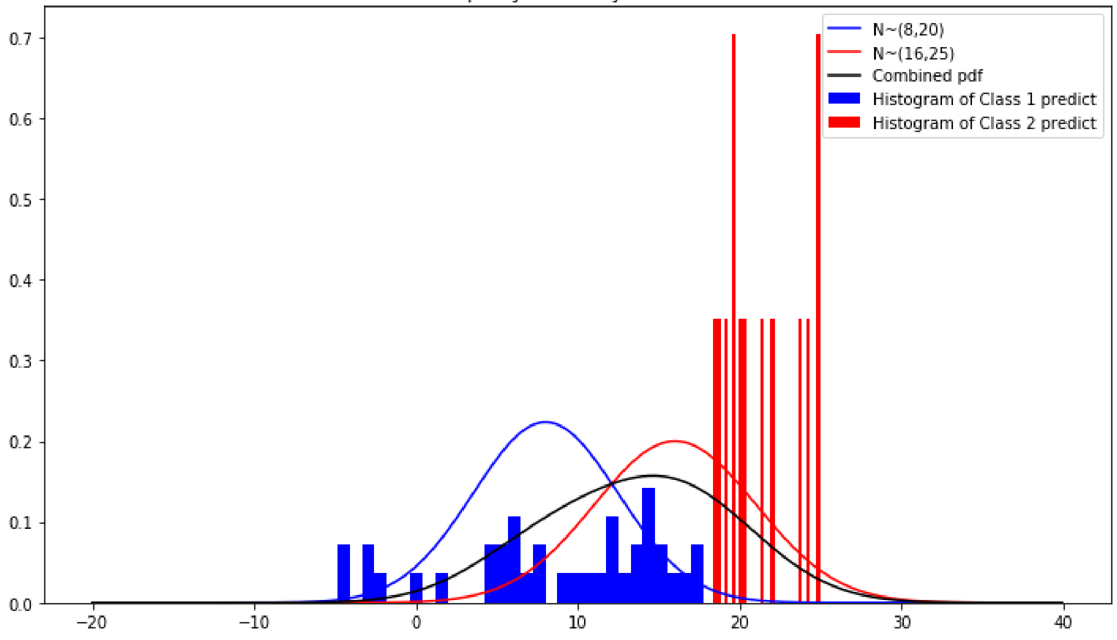




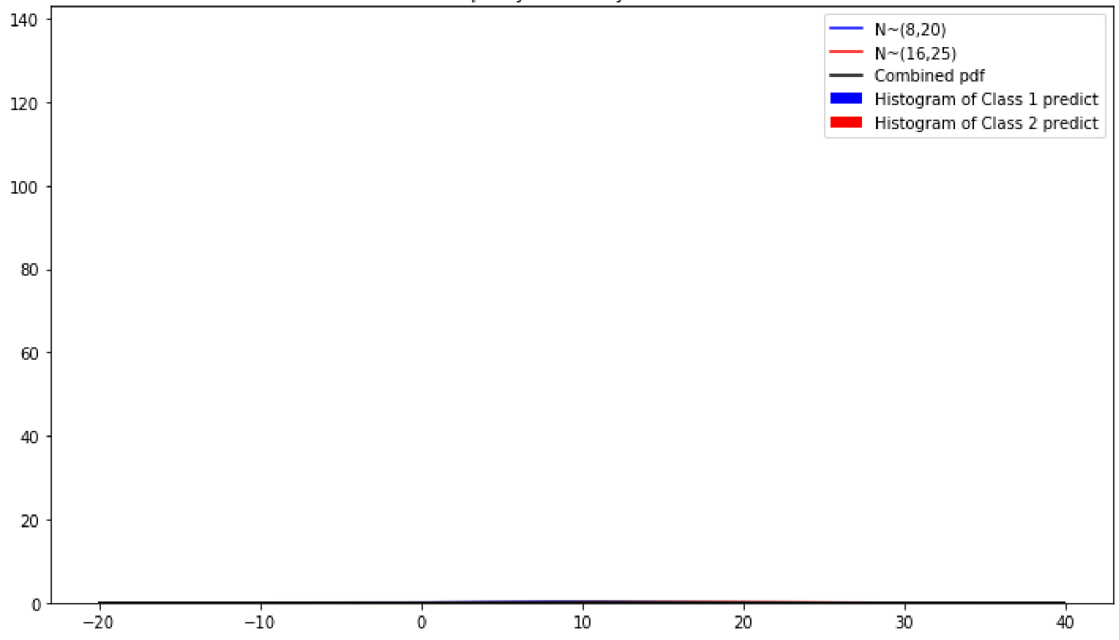

For Apriory Probabiliy as (0.7,0.3)
N~(8,20)
N~(16,25)
Combined pdf
Histogram of Class 1 predict
Histogram of Class 2 predict
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−40
−20
0
20
40

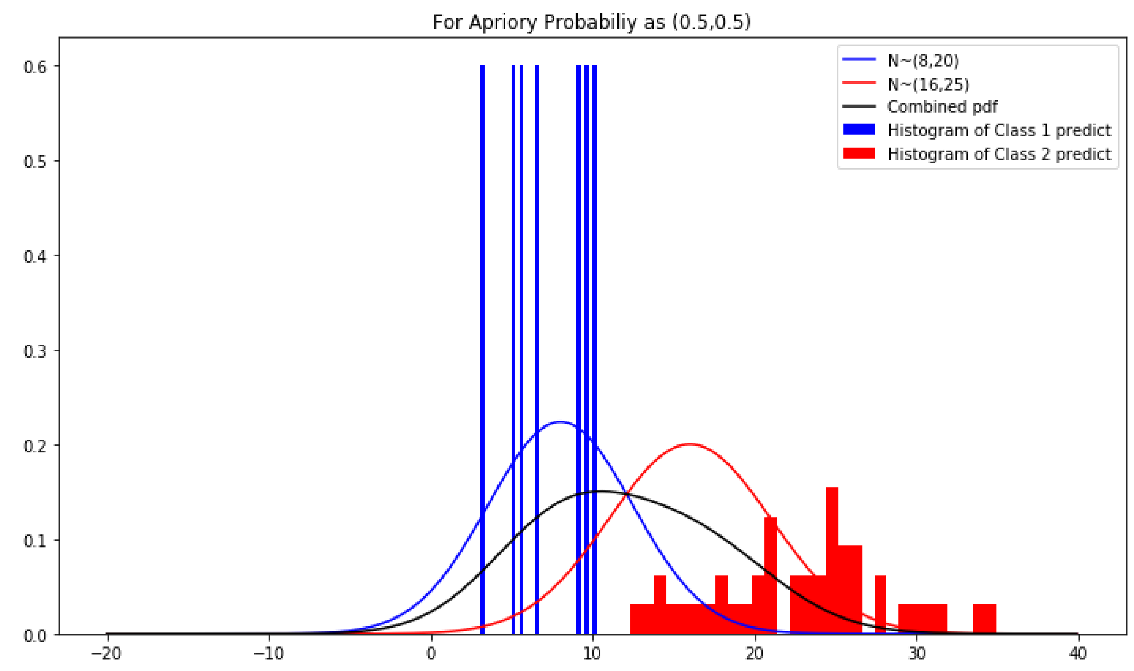

For Apriory Probabiliy as (0.5,0.5)
N~(8,20)
N~(16,25)
Combined pdf
Histogram of Class 1 predict
Histogram of Class 2 predict
0.25
0.20
0.15
0.10
0.05
0.00
−20
−10
0
10
20
30
40

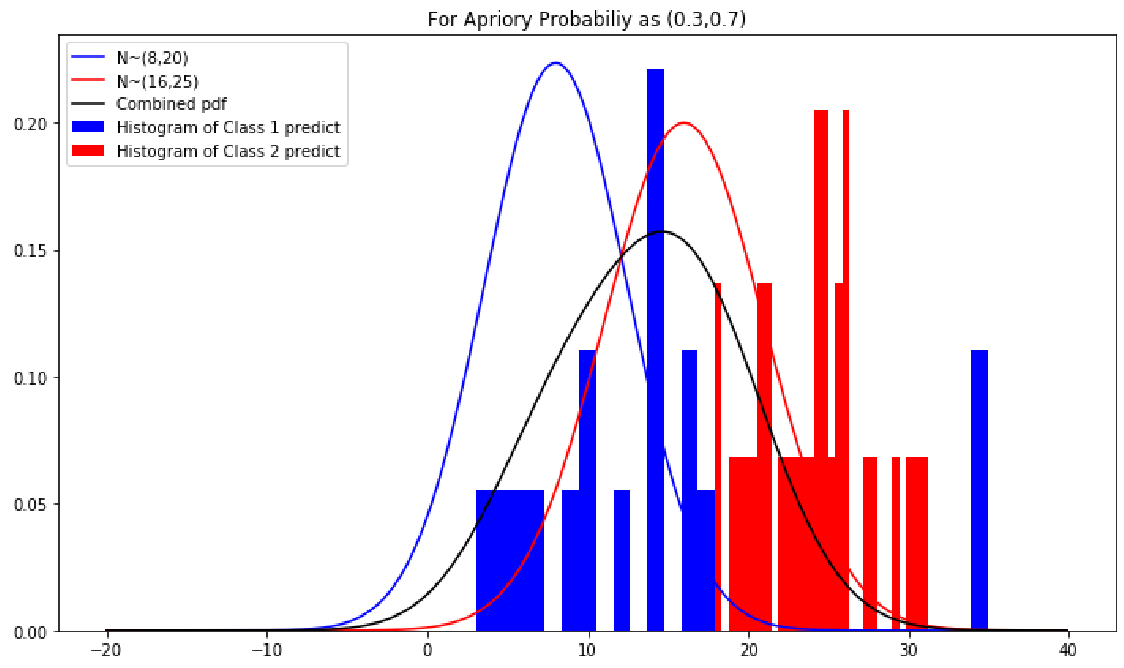
For Apriory Probabiliy as (0.3,0.7)

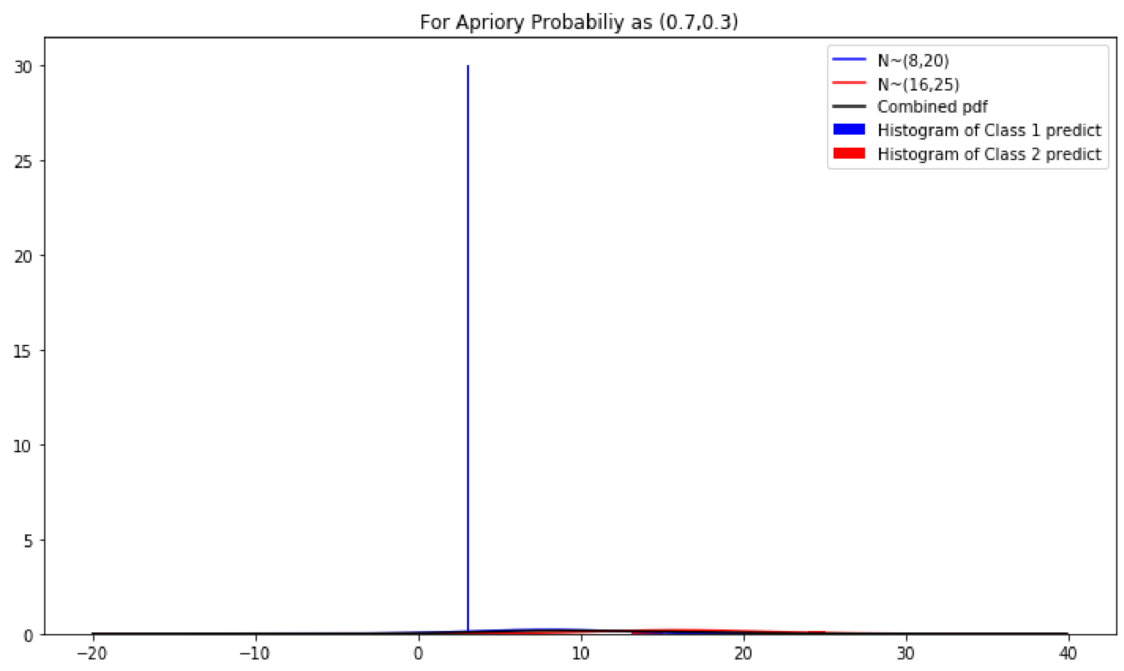


For Apriory Probabiliy as (0.7,0.3)




For Apriory Probabiliy as (0.5,0.5)
N~(8,20)
N~(16,25)
Combined pdf
Histogram of Class 1 predict
Histogram of Class 2 predict


For Apriory Probabiliy as (0.3,0.7)
N~(8,20)
N~(16,25)
Combined pdf
Histogram of Class 1 predict
Histogram of Class 2 predict


For Apriory Probabiliy as (0.7,0.3)
N~(8,20)
N~(16,25)
Combined pdf
Histogram of Class 1 predict
Histogram of Class 2 predict
30
25
20
15
10
5
0
−20
−10
0
10
20
30
40

```
[ ]:
```