Two Category Classification Using Bayesian Decision Rule

Kevin De Angeli
kevindeangeli@utk.edu
COSC 522 - Machine Learning
University of Tennessee, Knoxville

September 19, 2019

Abstract

This paper explores the application of Bayesian decision rules for classification tasks on a synthetic data set. I compare the performance of four models based on different assumptions about the data. Performance was measured based on accuracy and I present graphs to provide a broader insight about the classifier. I developed a prior probability analysis for each of the classifiers, and I identified the parameters that lead to the greatest accuracy. Finally, I compared class-wise accuracy given different priors. My result shows that even though certain assumptions about the data can simplify the model, the greatest accuracy is achieved when assumptions are minimized or nonexistent.

1 Introduction

Bayes' theorem is one of the cores of decision theory. Bayesian models are based on the assumption that the problem in hand can be described in probabilistic terms, and that the probability parameters can be extracted from the data [1]. Different variations of Bayesian models are used in diverse problem domains including language identification [2] and outlier detection [3]. The objective of this project is to design a decision rule on a synthetic data set with two categories, assuming that the data comes from a Gaussian distribution. All four classifiers presented in this paper showed a positive performance, and they can be used as the basis for future classification problems. The Python scripts created for this project are provided in the Appendix.

2 Methods

2.1 Data Set

The synthetic data set used in this project comes from [1] and it consist of two files. One of the files is the training data set (Figure 1a) which contains 250 rows and 3 columns; two of the columns are variables (x_1, x_2), and the third column (y) contains the associated label or class. The label column takes values of 1 or 0. The other file is the test data (Figure 1b), and it contains 1000 rows and 3 columns with the same pattern that the training data set.

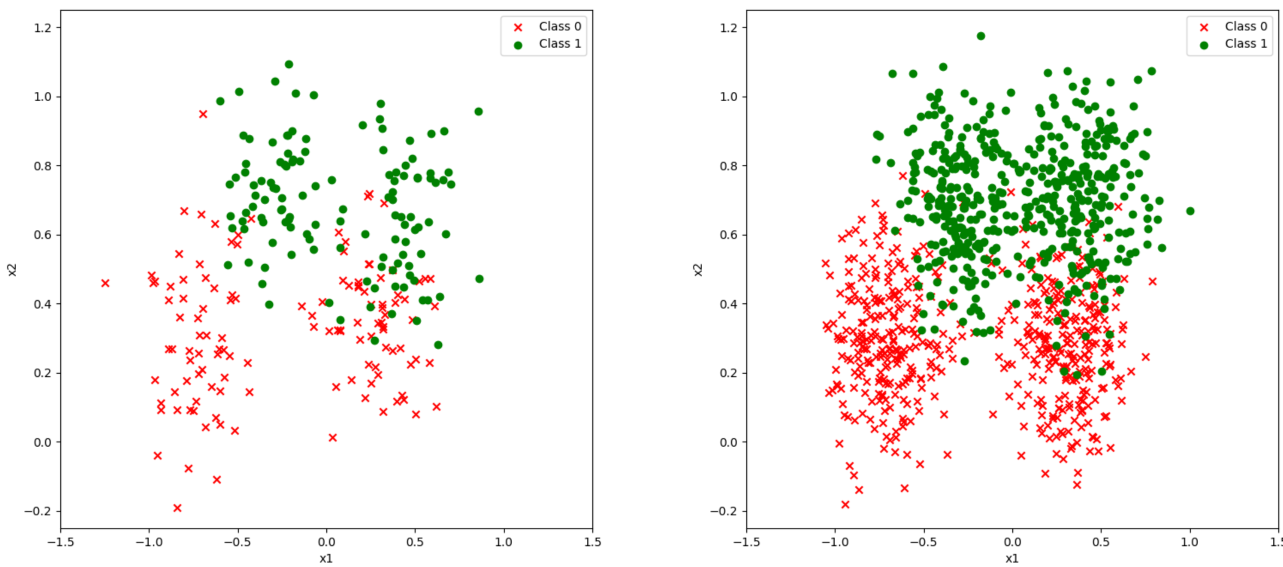
2.2 One-modal Gaussian: Case I

For the first classifier, I make the assumption that the standard deviation of the classes is the same and there is no correlation between the classes. This assumption implies that the features are statistically independent[1] :

$$\Sigma_i = \sigma^2 \mathbf{I}$$

In this project, I choose σ as the average of the standard deviations of the two columns of the synthetic data set (x_1, x_2).

Geometrically, Case I assumption corresponds to the case in which "the samples fall in equal-size hyperspherical clusters" [1] It is also important to notice that Case I classify data points based on the Euclidean distance between each point and the center of the clusters.



(a) Training data set

(b) Testing data set

Figure 1: The synthetic data set

Under this assumption, the discriminant function takes the form:

$$g_i(x) = -\frac{1}{2\sigma^2}[\mathbf{x}^t\mathbf{x} - 2\mu_i^t\mathbf{x} + \mu_i^t\mu_i] + \ln P(\omega_i) \tag{1}$$

Figure 2 shows the plot of the discriminant function g_0 with σ obtained from the synthetic data set. Even though equation (1) seems to take the form of a quadratic equation, the term $\mathbf{x}^t\mathbf{x}$ is the same for all classes, making this equation a linear discriminant function [1]. Given that the synthetic data set contains two classes, the decision boundary is simply calculated by solving:

$$g_1(x) = g_2(x)$$

This idea is applied to find the decision boundaries of Case I, II, and III. However, Richard O. Duda [1] points out an alternative way to calculate the decision boundary in case I:
Give the vector

$$w = \mu_i - \mu_j$$

and the point

$$\mathbf{x}_0 = \frac{1}{2}(\mu_i + \mu_j) - \frac{\sigma^2}{||\mu_i - \mu_j||^2} \ln \frac{P(\omega_i)}{P(\omega_j)}(\mu_i - \mu_j) \tag{2}$$

The decision boundary will also be given by the line which passes through $\mathbf{x}_0$ orthogonal to the vector w. Additionally, note that in Equation 2, the right part containing ln becomes 0 when the prior probabilities are equal. Therefore, these identities provide a simple and quick way to compute the decision boundary. The Python program provided in the appendix contains both, the discriminant functions from equation (1) and the decision boundary obtained from equation (2).

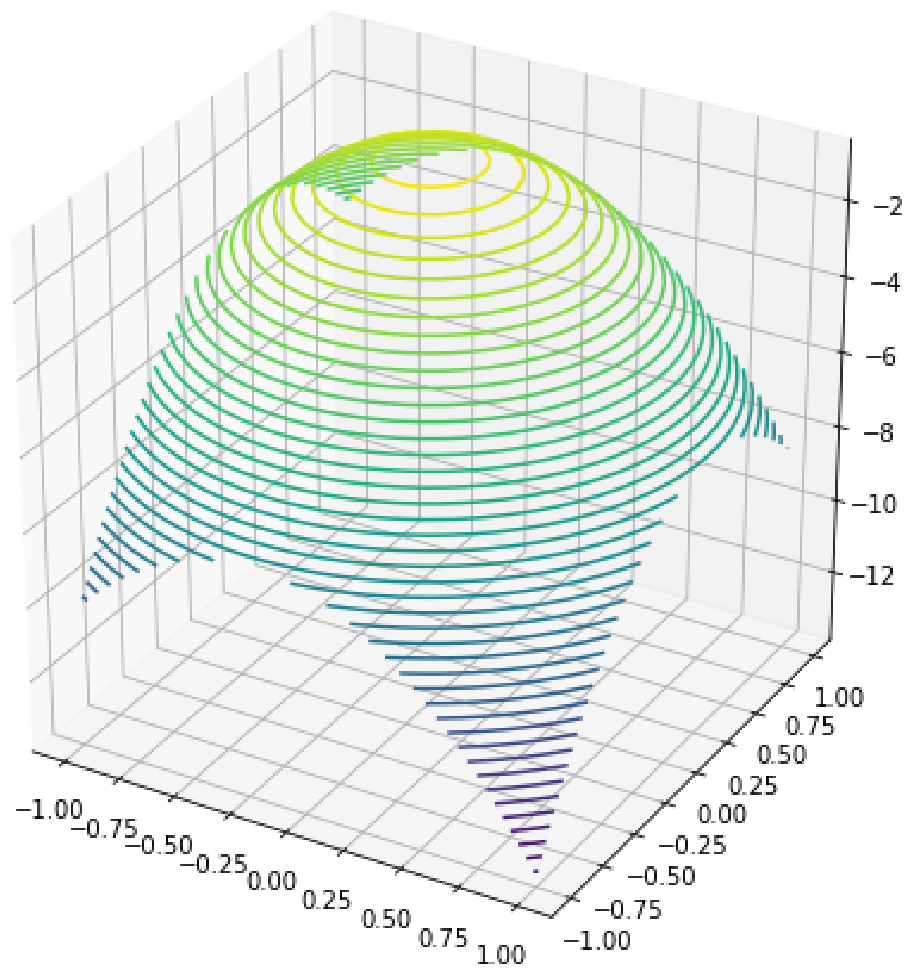


Figure 2: g_0 with μ and σ obtained from the synthetic data set.

2.3 One-modal Gaussian: Case II

The second case of the Gaussian classifiers assumes that the covariance matrices of both classes are the same, that is

$$\Sigma_i = \Sigma$$

Geometrically, this assumption "corresponds to the situation in which the sample fall in hyperllipsoidal clusters of equal size and shape" [1]. In contrast to Case I, Case II classifies data points based on

Mahalanobis distance which considers not only the distance between the data point and the center of the cluster, but also takes into consideration the covariance matrix of the classes. Note that just like in Case I, the decision boundary of Case II is also defined as a linear function.
A natural question to ask is how to choose the common covariance matrix. Assuming no correlation between classes, there are two methods that seem to justify the two entries of the matrix intuitively:

1. Use the standard deviation of the first and second columns of the data set as the two entries.

2. Use the average of the two standard deviations of the two columns when $y = 0$ and the average of the two standard deviations when $y = 1$

For this project, I have choose Σ following method 2.

Under the assumptions of Case II, the discriminant functions can be simplified as:

$$g_i(x) = (\Sigma^{-1}\mu_i)^t\mathbf{x} - \frac{1}{2}\mu_i^t\Sigma^{-1}\mu_i + \ln P(\omega_i) \tag{3}$$

2.4 One-modal Gaussian: Case III

The third Baysian classifier makes not assumption about the covariance matrix. Each discriminant function has their own covariance matrix calculated based on the statistics of the data set. Here, the discriminant function can not be simplified and take the following form:

$$g_i(x) = [\mathbf{x}^t(-\frac{1}{2}\Sigma_i^{-1})\mathbf{x}] + [(\Sigma_i^{-1}\mu_i)^t\mathbf{x}] - \frac{1}{2}[\mu_i^t\Sigma_i^{-1}\mu_i + \ln|\Sigma_i|] + \ln P(\omega_i) \tag{4}$$

2.5 Two-modal Gaussian

For the two-modal Gaussian model, I consider the case in which the data comes from a normal distribution but we two peaks instead of one. The μ and Σ were approximated based on the plot of the data. In order to approximate μ, I attempted to divide the data into four clusters (two for label 0 and two for label 1) and estimated the center of each of the clusters. Similarly, I approximated Σ by estimating how spread the data points of each clusters are. The discriminant for this model is given by:

$$g_i(x) = p_{x_1} + p_{x_2} \tag{5}$$

where:

$$\begin{aligned} p_{x_1} &= \tfrac{A_1}{(2\pi)^{d/2}|\Sigma_1|^{1/2}}\exp(\tfrac{(\mathbf{x}-\boldsymbol{\mu_1})^t\Sigma_1^{-1}(\mathbf{x}-\boldsymbol{\mu_1})}{-2}) \\ p_{x_2} &= \tfrac{A_2}{(2\pi)^{d/2}|\Sigma_2|^{1/2}}\exp(\tfrac{(\mathbf{x}-\boldsymbol{\mu_2})^t\Sigma_2^{-1}(\mathbf{x}-\boldsymbol{\mu_2})}{-2}) \end{aligned}$$

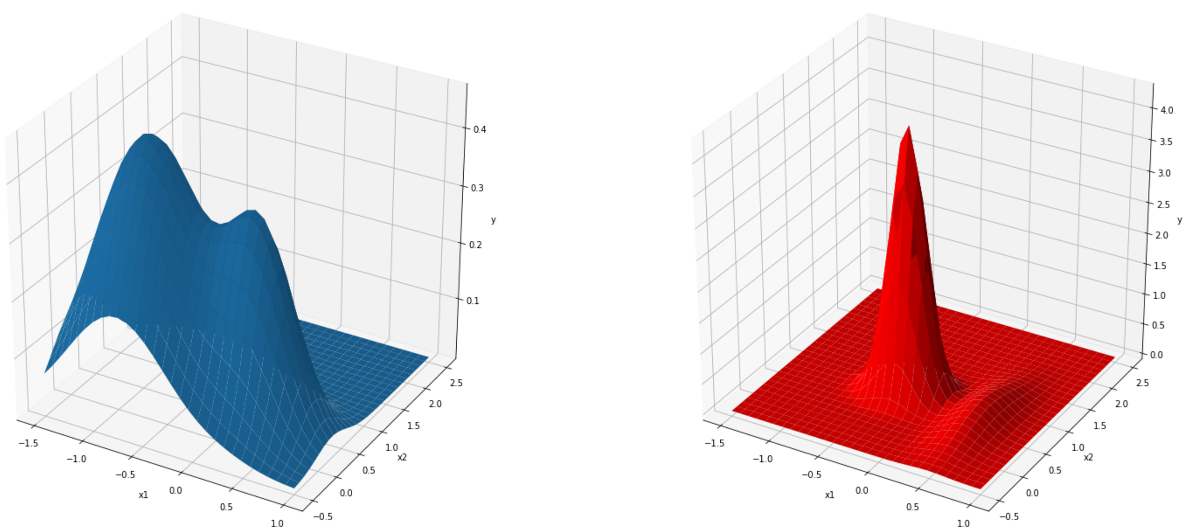
and

$$A_2 = 1 - A_1$$

Note that d stands for the dimensions of the data set, which in our problem domain $d = 2$.
The specific values chosen for this model are listed in Table 1. Figure 3 display the shape of the resulting curves.

Two-modal Gaussian		
Parameters	g_0	g_1
A_1	0.8	0.8
μ_1	$\begin{bmatrix}-0.75 & 0.2\end{bmatrix}$	$\begin{bmatrix}-0.31 & 0.75\end{bmatrix}$
μ_2	$\begin{bmatrix}0.3 & 0.3\end{bmatrix}$	$\begin{bmatrix}0.48 & 0.65\end{bmatrix}$
Σ_1	$\begin{bmatrix}0.25 & 0 \\ 0 & 0.3\end{bmatrix}$	$\begin{bmatrix}0.25 & 0 \\ 0 & 0.3\end{bmatrix}$
Σ_2	$\begin{bmatrix}0.1 & 0 \\ 0 & 0.1\end{bmatrix}$	$\begin{bmatrix}0.1 & 0 \\ 0 & 0.1\end{bmatrix}$

Table 1



(a) (b)

Figure 3: Two-modal Gaussian curves using parameters from Table 1. Plot (a) represents g_1 and plot (b) represents g_2.

2.6 Accuracy Analysis

I calculated the total accuracy by simply computing the ratio:

$$\frac{\text{Number of labels guessed correctly}}{\text{Number of total rows}}$$

For each model, I computed and compare the accuracy when equal probability is assumed. Additionally, For the one-modal cases, I calculated the accuracy over a wide range of different prior probabilities.

In order to further evaluate the classifier's accuracy, I have also compared class-wise accuracy. Assuming that class 1 is "positive" and 0 is "negative", I have calculated the True Positive Rate (TPR):

$$\frac{TP}{TP+FN}$$

and the True Negative Rate (TNR):

$$\frac{TN}{TN+FP}$$

3 Results

For the one-modal classifiers, I first calculated the decision boundary by solving $g_1(x) = g_2(x)$. Then, I developed an analysis of the accuracy with respect to a given prior probability. For the one-modal case, I assumed equal prior probability and calculated the overall accuracy. Section 3.4 presents an extensive comparison of the four classifiers.

3.1 One-modal Gaussian - Case I

3.1.1 Decision Rule

After setting up the discriminant equations equal to each other I obtained a the linear equation:

$$x_2 = -0.8326x_1 + 0.4438 \tag{6}$$

Figure 4a and 4b displays the decision boundary obtained from Equation 6.

Figure 4 shows that there exist a reasonable margin of error, specially when it comes to classifying class 0 data points as class 1. This demonstrates that making certain general assumptions about the data set can lead to simple models, but there exists a trade off between the simplicity of the model and the accuracy of the classifier.

3.1.2 Prior Probability Analysis

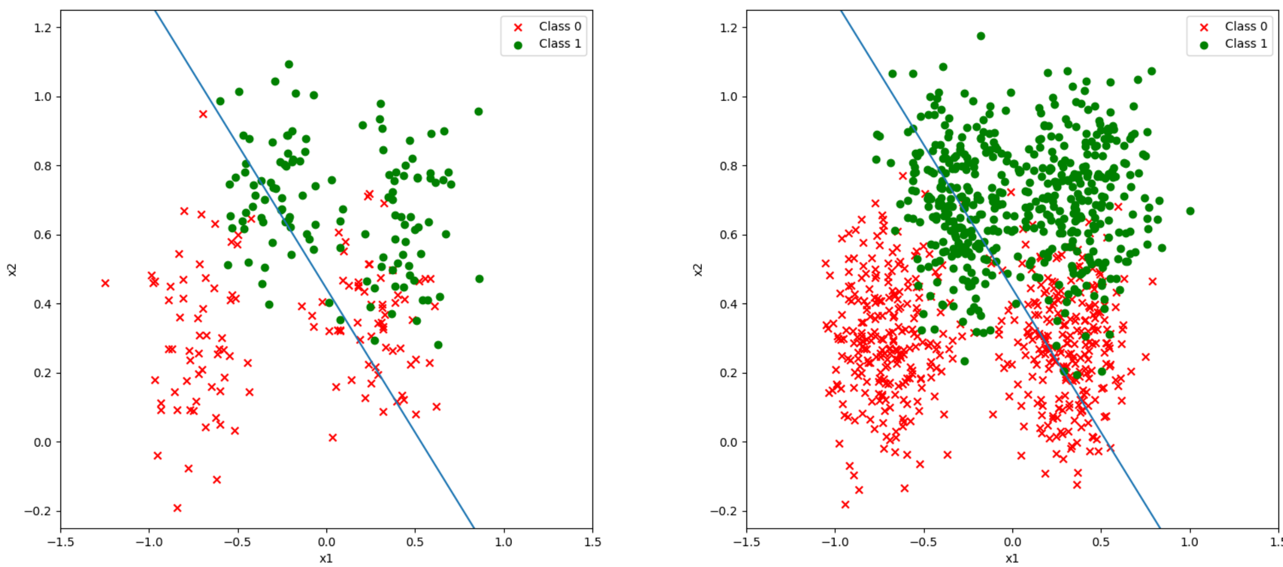
Figure 5 illustrates how the accuracy of the model is affected by different values of prior probabilities. One interesting aspect of this graph is that it contains multiple local maximums. It is simple to identify the maximum when working with two dimensions, but I predict certain complexities identifying the maximum of a data set with similar characteristics but with more dimensions. In Figure 5 the global maximum is obtained when $w = 0.329$ and that provides an accuracy of 73.5%.

3.2 One-modal Gaussian - Case II

3.2.1 Decision Rule

In the second case, solving $g_1(x) = g_2(x)$ also resulted in a linear equation:

$$x_2 = -0.504759841995623x_1 + 0.467636629718861 \tag{7}$$



(a) (b)

Figure 4: Decision rule from Case I applied to the training data set(a) and the test data set (b).

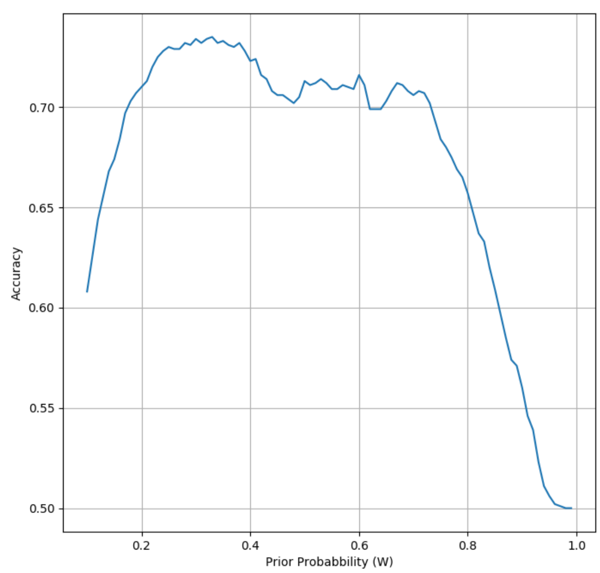
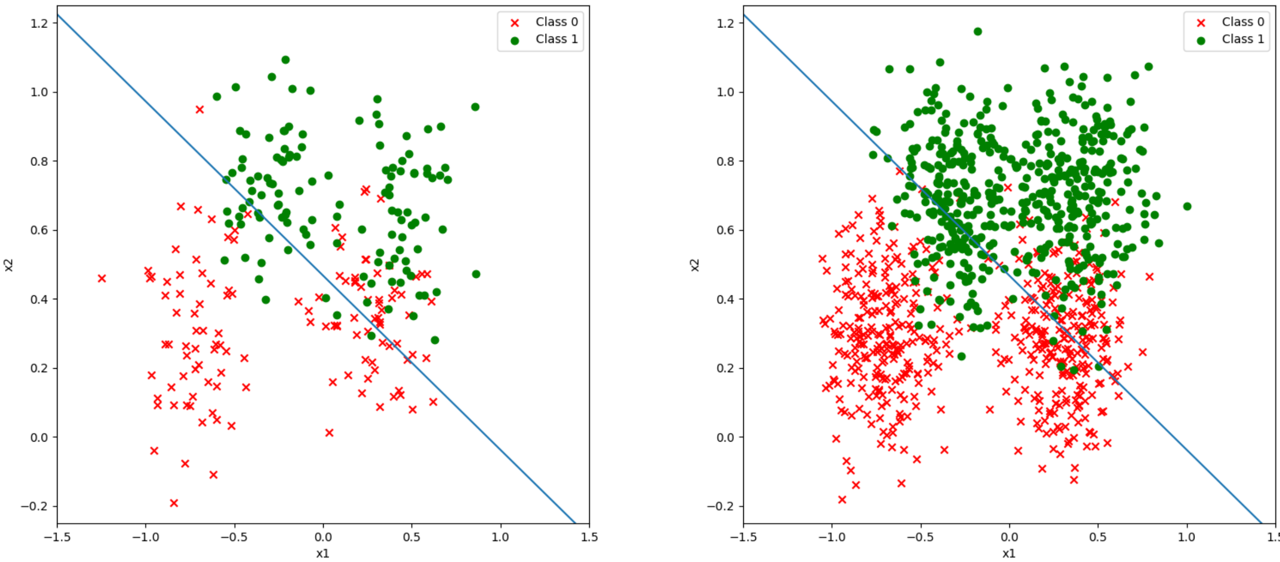


Figure 5: Prior probability vs. accuracy for Case I

It is clear from Figure 6 that this model provides a decision boundary that is a better fit for the data than Case I. This will also be shown in terms of accuracy in the next section.



(a) (b)

Figure 6: Decision rule from Case II applied to the training data set(a) and the test data set(b).

3.2.2 Prior Probability Analysis

Figure 7 displays the relationship between the prior probabilities and the accuracy of Case II. Unlike Figure 5, the shape of this graph is exactly what I would expect intuitively. The maximum accuracy is obtained when $w = 0.445$ which corresponds to an accuracy of 81.1%.

3.3 One-modal Gaussian - Case III

3.3.1 Decision Rule

The third case of the one-modal Gaussian models offers more flexibility than the previous two models because it provides a quadratic decision boundary instead of a linear function. Seems in the real world, clusters of data can rarely be separated by a linear function, this decision rule seems to perform better in most if not all scenarios. The decision boundary is generated by Equation 8 and its plot is displayed in Figure 8.

$$x_2 = -0.641052306096729x1 - 2.808174349754 17e - 16 * sqrt(-1.52678246136875e + 28x1^2 \\ - 1.96622605613547e + 31x1 + 3.28073472343665e + 31) + 2.0905232940685 \quad (8)$$

3.3.2 Prior Probability Analysis

The prior probability vs accuracy graph for Case III (Figure 9) follows a similar pattern that Figure 7 (Case II). Surprisingly, Figure 9 shows a high accuracy even when the w is extremely low. The lack

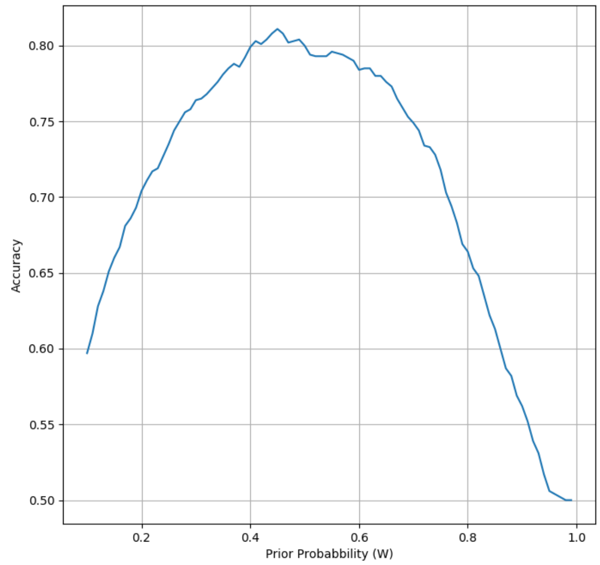
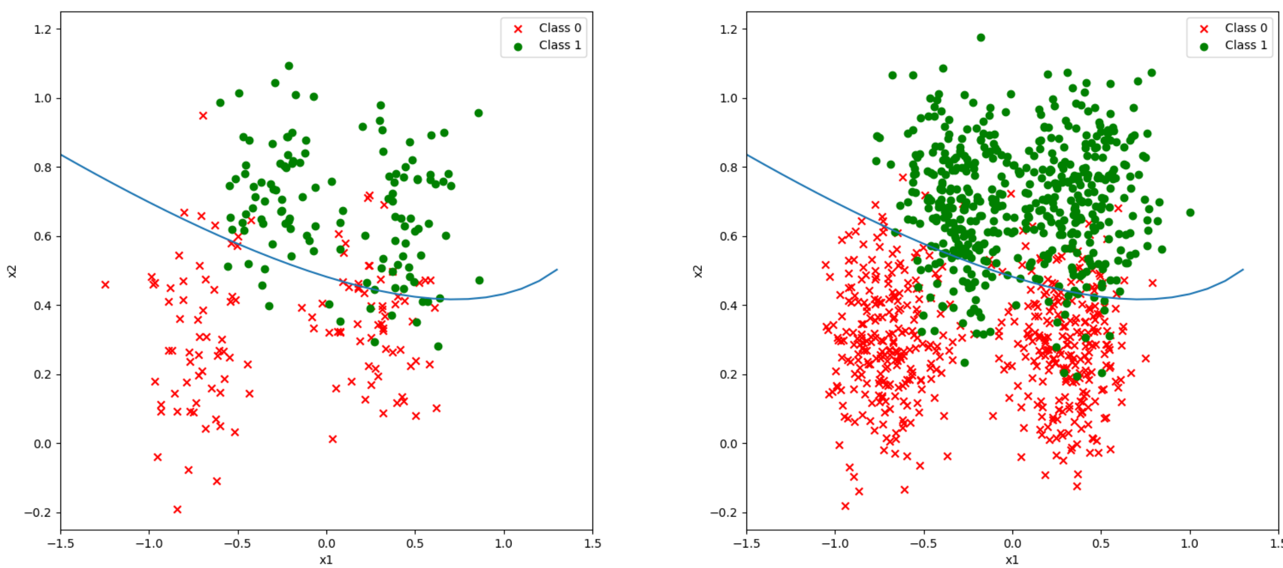


Figure 7: Prior probability vs. accuracy for Case II



(a)

(b)

Figure 8: Decision rule from Case III applied to the training data set(a) and the test data set(b).

of symmetry of this graph seemed counter intuitive. In Case III, the maximum accuracy achieved in the test data set is 90% and is archived when $w = 0.456$

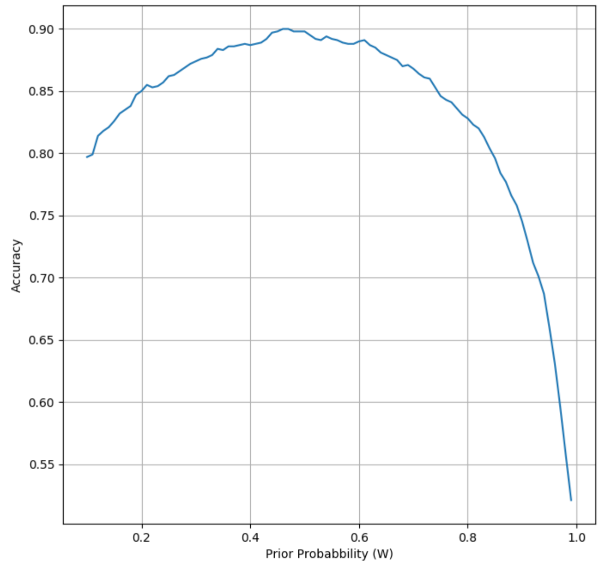


Figure 9: Prior probability vs. accuracy for Case III

3.4 Accuracy comparisons

Table 2 compares the accuracy of the four models developed in this project. Under equal prior probability, Case III performed better than the other models. Choosing the right prior probability improves the performance of all three models. However, the difference is small, with the greatest gap observed in Case I where the accuracy goes from 71.3% to 73.5%. Figure 10 offers an additional contrast by incorporating all three one-modal Gaussian decision boundaries.

Figure 11 provides a detailed contrast between the class-wise accuracy for the three one-modal classifiers. The common pattern between these plots is that when the prior probability is low, the classifier is bias towards Class 1, and with high prior probabilities, the model tends to favor Class 0. Case III seem offers a distinct scenario when the prior probability is low, since Class 0 is not heavily penalized.

Finally, Figure 12 shows the data points that were wrongly classified by each of the four models. Figure 12a and 12b show a similar pattern, which is expected since they both have linear decision boundaries.

Accuracy Comparison			
Classification Model	Equal Prior Probability	Maximum Accuracy	Prior Probability when Accuracy is Maximum
Case I	71.3%	73.5%	$w = 0.33$
Case II	80.0%	81.1%	$w = 0.45$
Case III	89.8%	90.0%	$w = 0.46$
Two-Modal	87.5%	—	—

Table 2

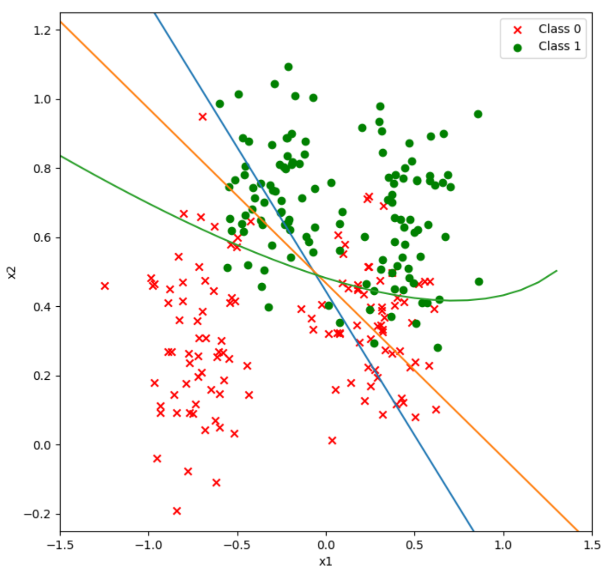
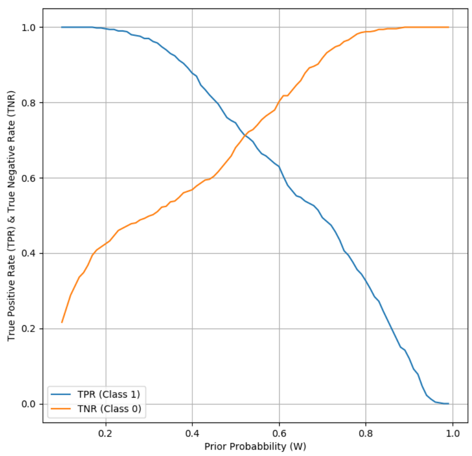
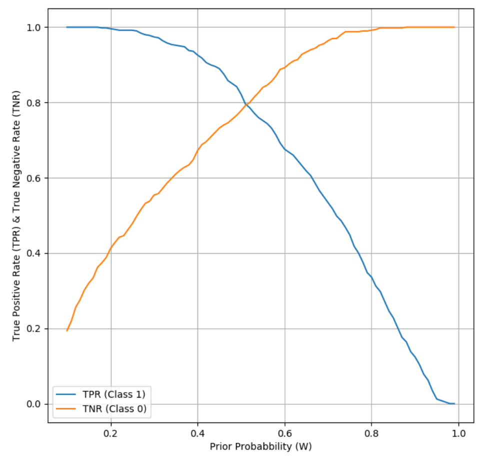


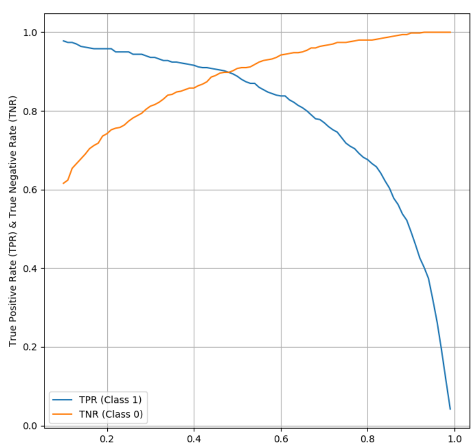
Figure 10: The three One-modal decision boundaries. The blue line and the yellow line represent Case I and II, respectively. Case III is represented by the green curve.



(a) Case I

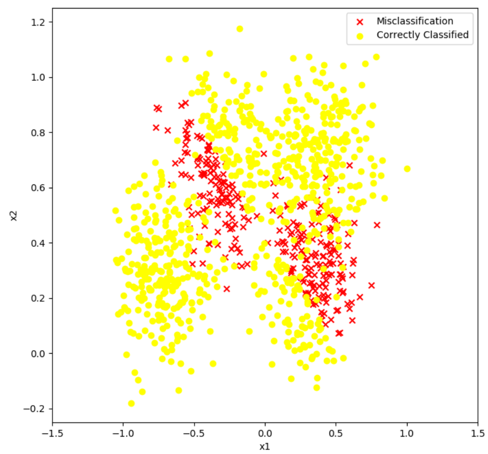


(b) Case II

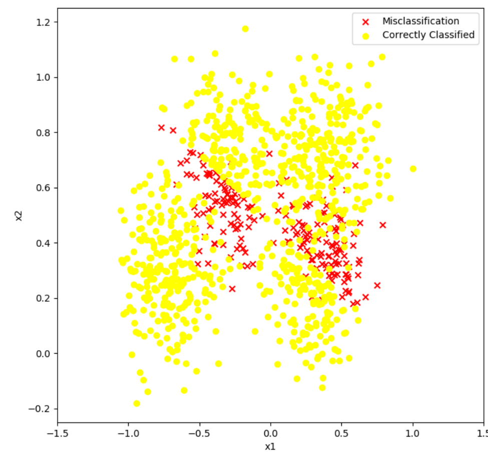


(c) Case III

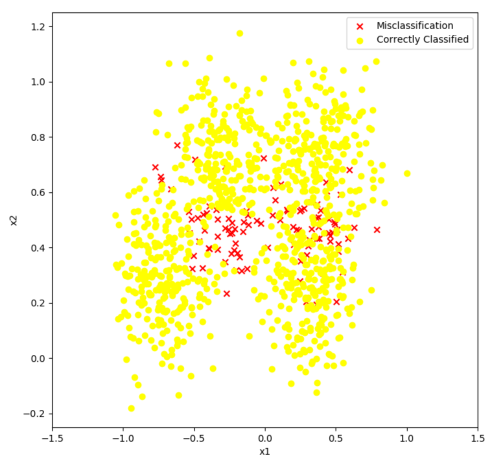
Figure 11: Prior Probabilities vs. True Positive Rates (TPR) & True Negative Rates (TNR) for the one-modal classifiers.



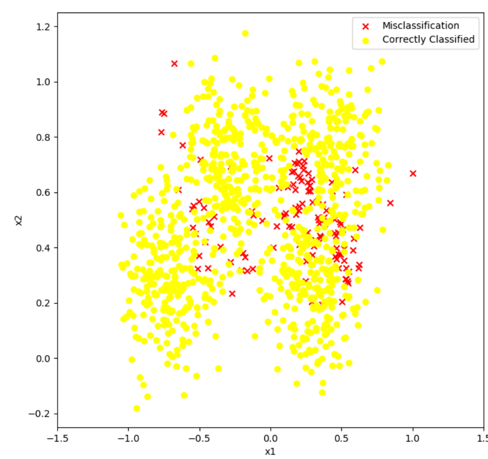
(a)



(b)



(c)



(d)

Figure 12: Misclassified points for Case I (a), Case II (b), Case III(c), and the Two-modal Case (d)

4 Discussion

This paper presents four Bayesian classification models based on different assumptions about the nature of the synthetic data set. For each classifier, the discriminant functions were derived based on statistics of the training data set. I used the discriminant functions to calculate decision boundaries and provide plots that reflect the uniqueness of each model. Overall, all four Bayesian cases exhibited a positive performance with a range of accuracies up to 90%. An extensive analysis of performance under different prior probabilities is presented, and the maximum possible accuracy for each case are 73.5% , 81.1%, and 90.0% for case I, II, and III respectively. Accuracy were also calculated for individual classes and plotted against a range of prior probabilities. Finally, I developed a brief error analysis to show the location of the misclassified points in contrast with the data points that were correctly classified. Ultimately, one can argue that Case III is the best classifier in terms of accuracy. However, accuracy does not take into consideration the complexity of the model and the computational power required to work with massive amounts of complex data. There may be situations in which a simpler model with a reasonable performance is required, and I think Case II may fit better for that type of scenarios. Future work should include a better way to approximate the parameters of the two-modal classifier, and a full analysis of performance that includes prior probabilities and weights (A_1 and A_2).

References

[1] Richard O. Duda, Peter E. Hart, David G. Stork. *Pattern Classification*. Second Edition. pdf.

[2] Pedro A. Torres-Carrasquillo I., Douglas A. Reynoldsl and J.R. Deller, Jr. *Language Identification using Gaussian Mixture Model Tokenization*

[3] David M.J. Tax and Robert P.W. Duin *Outlier Detection using Classifier Instability.*

5 Appendix

5.1 Python Script

```python
#************************************************************
# Read the data, calculate basic statistics and plot the data sets
#************************************************************


import numpy as np
import pandas as pd
import math  # Used for Pi and log()
import sympy as sym
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from numpy import ones, vstack

path = '/Users/kevindeangeli/Desktop/Fall2019/COSC520/' \
       'Project1/Project_files/synth.tr.txt'
training_data = pd.read_csv(path, delim_whitespace=1, header=None)
training_data.columns = ['x1', 'x2', 'y']
path2 = '/Users/kevindeangeli/Desktop/Fall2019/COSC520/' \
        'Project1/Project_files/synth.te.txt'
test_data = pd.read_csv(path2, delim_whitespace=1, header=None)
test_data.columns = ['x1', 'x2', 'y']
y1Values = training_data[training_data['y'] == 1]
y0Values = training_data[training_data['y'] == 0]

y0x1Mean = y0Values.loc[:, "x1"].mean()
y0x2Mean = y0Values.loc[:, "x2"].mean()
y1x1Mean = y1Values.loc[:, "x1"].mean()
y1x2Mean = y1Values.loc[:, "x2"].mean()

y0x0 = y0Values['x1'].tolist()
yox1 = y0Values['x2'].tolist()
y0Cov = np.cov(np.array([y0x0, yox1]))
y1x0 = y1Values['x1'].tolist()
y1x1 = y1Values['x2'].tolist()
y1Cov = np.cov(np.array([y1x0, y1x1]))

y0X1Std = y0Cov[0, 0]
y0X2Std = y0Cov[1, 1]
y1X1Std = y1Cov[0, 0]
y1X2Std = y1Cov[1, 1]


def plotData():
```

```python
    plt.figure(num=None, figsize=(8, 8), dpi=100, facecolor='w', edgecolor='k')
    p1 = plt.scatter(y0Values[['x1']], y0Values[['x2']], color='red', marker='x')
    p2 = plt.scatter(y1Values[['x1']], y1Values[['x2']], color='green', marker='o')
    plt.xlim(-1.5, 1.5)
    plt.ylim(-0.25, 1.25)
    # plt.title("Data set from the Pattern Classification by Richard O. Duda")
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.legend((p1, p2), ('Class 0', 'Class 1'))


def plotTestData():
    plt.figure(num=None, figsize=(8, 8), dpi=100, facecolor='w', edgecolor='k')
    y1Values = test_data[test_data['y'] == 1]
    y0Values = test_data[test_data['y'] == 0]
    p1 = plt.scatter(y0Values[['x1']], y0Values[['x2']], color='red', marker='x')
    p2 = plt.scatter(y1Values[['x1']], y1Values[['x2']], color='green', marker='o')
    plt.xlim(-1.5, 1.5)
    plt.ylim(-0.25, 1.25)
    # plt.title("Data set from the Pattern Classification by Richard O. Duda")
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.legend((p1, p2), ('Class 0', 'Class 1'))


def classifyAndEvaluate(testData, w):
    right = 0
    wrong = 0
    for index, row in testData.iterrows():
        x1 = row['x1']
        x2 = row['x2']
        g0_out = g0(x1, x2, w)
        g1_out = g1(x1, x2, w)
        if g0_out >= g1_out:
            guessLabel = 0
        else:
            guessLabel = 1
        if guessLabel == row['y']:
            right = right + 1
        else:
            wrong = wrong + 1
    return right / test_data.shape[0]


def EvaluatePriorProbs(Ws, test_data):
    accuracyArray = []
```

```
    for i in range(len(Ws)):
        w = Ws[i]
        # print("w: ", w)
        accuracy = classifyAndEvaluate(test_data, w)
        accuracyArray.append(accuracy)
    accuracyArray = np.array(accuracyArray)
    return accuracyArray


def plotAccuracyCurve(Ws, ys, figureName):
    fig, ax = plt.subplots(figsize=(8, 8), dpi=100, facecolor='w', edgecolor='k')
    ax.plot(Ws, ys)
    ax.set(xlabel='Prior Probabbility (W)', ylabel='Accuracy',
           title=' ')
    # title='Finding the best accuracy')
    ax.grid()
    # plt.show()
    print("Maximun accuracy is provided is obtained when W= ", Ws[np.argmax(accuracy_array)])
    print("That corresponds to an accuracy of: ", np.amax(accuracy_array))
    print(" ")
    plt.savefig(figureName)


Ws = np.arange(0.1, 1.0, 0.01)  # Can't start at 0 because log(0) = infinity -- These are Prior Probs.

mewY0 = np.array([[y0x1Mean, y0x2Mean]])
mewY1 = np.array([[y1x1Mean, y1x2Mean]])
plotData()
plt.savefig('dataset.png')
plotTestData()
plt.savefig('TestData.png')

#****************************************************************
#This script prints the Recall (True Positive Rate (TPR))
# and the True Negative Rate (TNR)
#It calculates the values for all the different prior
# probabilities and graph them together.
#****************************************************************



def ConfusionMatrixWithPriors(testData,w,g0,g1):
    #Let class 0 = N; Class 1 = P;
    TP = 0
```

```python
    TN = 0
    FP = 0
    FN = 0
    for index,row in testData.iterrows():
        x1=row['x1']
        x2=row['x2']
        g0_out = g0(x1,x2,w)
        g1_out = g1(x1,x2,w)
        if g0_out>=g1_out:
            guessLabel=0
        else:
            guessLabel=1
        rightLabel=row['y']
        if guessLabel==rightLabel:
            if rightLabel == 1:
                TP+=1
            else:
                TN+=1
        else:
            if rightLabel==1:
                FN+=1
            else:
                FP+=1
    #print(TP)
    totalRowsInData = test_data.shape[0]
    confusion_matrix = [['TPR', TP/(TP+FN)], ["TNE", TN/(TN+FP)], ['FP',
    ↪  FP/totalRowsInData], ['FN', FN/totalRowsInData]]
    return confusion_matrix

def EvaluatePriorProbsConfusionMatrix(Ws,test_data,g0,g1):
    TPR_array =[]
    TNE_array =[]
    FP_array =[]
    FN_array =[]
    for i in range(len(Ws)):
        w=Ws[i]
        confusion_matrix = ConfusionMatrixWithPriors(test_data,w,g0,g1)
        TPR_array.append(confusion_matrix[0][1])
        TNE_array.append(confusion_matrix[1][1])
        FP_array.append(confusion_matrix[2][1])
        FN_array.append(confusion_matrix[3][1])
    plotRecallCurves(Ws,TPR_array,TNE_array)


def plotRecallCurves(Ws,TPR_array,TNE_array):
    fig, ax = plt.subplots(figsize=(8, 8), dpi=100, facecolor='w', edgecolor='k')
```

```python
    plt.plot(Ws,TPR_array, label= 'TPR (Class 1)')
    plt.plot(Ws,TNE_array, label= 'TNE (Class 0)' )
    ax.set(xlabel='Prior Probabbility (W)', ylabel='True Positive Rate (TPR) & True Negative Rate (TNR)',
    title=' ')
    plt.legend()
    ax.grid()
    print(" ")
    plt.savefig('RecallCurve')


#******************************************************************
# Declare two new functions to plot the missclassified points
# together with the correctly classified data points.
#******************************************************************

def classifyAndEvaluateWithWrongDisplay(testData, w, g0, g1):
    right = 0
    wrong = 0
    wrongIndexes = []
    for index, row in testData.iterrows():
        x1 = row['x1']
        x2 = row['x2']
        g0_out = g0(x1, x2, w)
        g1_out = g1(x1, x2, w)
        if g0_out >= g1_out:
            guessLabel = 0
        else:
            guessLabel = 1
        if guessLabel == row['y']:
            right = right + 1
        else:
            wrong = wrong + 1
            wrongIndexes.append(index)

    WrongIndexTable = test_data.iloc[wrongIndexes]
    test_dataMinusWrongs = test_data.drop(wrongIndexes)
    plotWrongDataPoints(test_dataMinusWrongs, WrongIndexTable)
    # return wrongIndexes #Returns a list of indexes of misclassified points


def plotWrongDataPoints(test_data, wrongPoints):
    plt.figure(num=None, figsize=(8, 8), dpi=100, facecolor='w', edgecolor='k')
    p1 = plt.scatter(wrongPoints[['x1']], wrongPoints[['x2']], color='red', marker='x')
```

```
    p2 = plt.scatter(test_data[['x1']], test_data[['x2']], color='yellow',
     ↪  marker='o')
    plt.xlim(-1.5, 1.5)
    plt.ylim(-0.25, 1.25)
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.legend((p1, p2), ('Misclassification', 'Correctly Classified'))
    plt.savefig('Miscclassification.png')

# classifyAndEvaluateWithWrongDisplay(test_data,.5,g0,g1)

#****************************************************************
#Case I
#Here is just a plot of g_0 just to see how it looks
#****************************************************************

simgaY0=(y0X1Std + y0X2Std + y1X2Std+ y1X1Std)/4
mewY0 = np.array([[y0x1Mean,y0x2Mean]])
def f(x, y):
    V= np.array([[x,y]])
    w=.5
    return  (((-1/(2*simgaY0)) * (np.dot(V,np.transpose(V)) \
    -2*np.dot(mewY0,np.transpose(V))+
     ↪  np.dot(mewY0,np.transpose(mewY0))))+math.log(w)).item()

x = np.linspace(-1, 1, 30)
y = np.linspace(-1, 1, 30)

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

Z = np.vectorize(f)
#Z = f(X, Y)
#print(Z)
fig = plt.figure(figsize=(8,8))
ax = plt.axes(projection='3d')
#ax.contour3D(X, Y, Z, 50, cmap='binary')
ax.contour3D(X, Y, Z(X,Y),50)
plt.savefig('discriminant.png')

ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('y');

#****************************************************************
#Case I
#Testing the performance of the Case I Classifier:
```

```python
#***************************************************************


simgaY0=(y0X1Std + y0X2Std + y1X2Std+ y1X1Std)/4  ## This should be arbitrary so I got the total avrage.
mewY0 = np.array([[y0x1Mean,y0x2Mean]])
mewY1 = np.array([[y1x1Mean,y1x2Mean]])

def g0(x, y, w):
    V= np.array([[x,y]])
    return (((-1/(2*simgaY0)) * (np.dot(V,np.transpose(V)) \
    -2*np.dot(mewY0,np.transpose(V))+ np.dot(mewY0,np.transpose(mewY0))))+math.log(w)).item()

def g1(x, y,w):
    V= np.array([[x,y]])
    return (((-1/(2*simgaY0)) * (np.dot(V,np.transpose(V)) \
    -2*np.dot(mewY1,np.transpose(V))+ np.dot(mewY1,np.transpose(mewY1))))+math.log(1-w)).item()

accuracy_array=EvaluatePriorProbs(Ws,test_data)

plotAccuracyCurve(Ws,accuracy_array,'CaseIAccuracy.png')

print("With equal prior probability, accuracy: ",classifyAndEvaluate(test_data,.5) )

classifyAndEvaluateWithWrongDisplay(test_data,.5,g0,g1)

#***************************************************************
#Plot the True Positive Rate/True Negative Rate for Case I
#***************************************************************
accuracy_array=EvaluatePriorProbsConfusionMatrix(Ws,test_data,g0,g1)


#***************************************************************
#Case I
#In order to find the decision boundary, I used equation 56 (page 21) and the fact that:
#"This equation definesa hyperplane through the point x0 and
#orthogonal to the vector w" Where w = u0-u1 (difference of means)
#***************************************************************

u1 = np.array([-0.22147023711999997, 0.32575494064000005])
u2 = np.array([0.07595431392, 0.6829689131999999])
x1 = sym.Symbol('x1')
```

```
w = u1-u2
x0 = (u1+u2)/2
a= w*x1
a2=a-x0

a3=a2*w
eq= np.sum(a3)

sol= sym.solve(eq,x1)
sol2 = sol[0]
point2=w*sol2

print("The line passes through these two points: ")
print(point2)
print(x0)
print("\n")

print("Therefore, the equation of the line is :  y= -0.8326x+0.4438")


#*************************************************************
#Case I
#Plotting Decision Boundary
#*************************************************************

def caseIdecisionRule():
    x= np.arange(-1.5,2,.1)
    y=[]
    for i in x:
        ii=-0.8326*i+0.4438
        y.append(ii)
    plt.plot(x,y)
    plt.savefig('CaseIDecisionBoundary.png')

#plotData()
#caseIdecisionRule()

plotTestData()
caseIdecisionRule()


#*************************************************************
#Case II
#Discriminat Function
#Evaluating the acuracy.
#*************************************************************
```

```python

E0E1Average1= (y0X1Std+y0X2Std)/2
E0E1Average2= (y1X1Std+y1X2Std)/2
Ex = np.array([(E0E1Average1,0), (0,E0E1Average2)])
Ex_inv = np.linalg.inv(Ex)

u0 = np.array([-0.22147023711999997, 0.32575494064000005])
u1 = np.array([0.07595431392,0.6829689131999999])
def g0(x, y, w):
    X = np.array([x,y])
    LHS1= np.dot(np.dot(Ex_inv,u0),X)
    LHS2= -.5 * np.dot(np.dot(u0,Ex_inv),u0)
    LHS3= math.log(w)
    LHS = LHS1 + LHS2 + LHS3
    return LHS

def g1(x, y, w):
    X = np.array([x,y])
    RHS1= np.dot(np.dot(Ex_inv,u1),X)
    RHS2= -.5 * np.dot(np.dot(u1,Ex_inv),u1)
    RHS3= math.log(1-w)
    RHS = RHS1 + RHS2 + RHS3
    return RHS



accuracy_array=EvaluatePriorProbs(Ws,test_data)
plotAccuracyCurve(Ws,accuracy_array,'CaseIIAcuracy.png')

print("With Equal Prior Probability, accuracy: ", classifyAndEvaluate(test_data,.5))
classifyAndEvaluateWithWrongDisplay(test_data,.5,g0,g1)

#*************************************************************
#True Postive Rate/ True Negative Rate for Case II
#*************************************************************

accuracy_array=EvaluatePriorProbsConfusionMatrix(Ws,test_data,g0,g1)


#*************************************************************
#Case II
#Finding Decision Boundary:
#*************************************************************


```

```
E0E1Average1= (y0X1Std+y0X2Std)/2
E0E1Average2= (y1X1Std+y1X2Std)/2
Ex = np.array([(E0E1Average1,0), (0,E0E1Average2)])
Ex_inv= np.linalg.inv(Ex)

u0 = np.array([-0.22147023711999997, 0.32575494064000005])
u1 = np.array([0.07595431392,0.6829689131999999])

x1 = sym.Symbol('x1')
x2 = sym.Symbol('x2')
X = np.array([x1,x2])

LHS1= np.dot(np.dot(Ex_inv,u0),X)
LHS2= -.5 * np.dot(np.dot(u0,Ex_inv),u0)
LHS = LHS1 + LHS2

RHS1= np.dot(np.dot(Ex_inv,u1),X)
RHS2= -.5 * np.dot(np.dot(u1,Ex_inv),u1)
RHS = RHS1 + RHS2

eq = LHS + (-1*RHS)

sol= sym.solve(eq,x2, set=True)
print("Rule 2 - Classification boundary: ", sol)


#*****************************************************************
# Case II
# Plotting Decision Boundary:
#*****************************************************************


def caseIIdecisionRule():
    x = np.arange(-1.5, 2, .1)
    y = []
    for i in x:
        ii = 0.467636629718861 - 0.504759841995623 * i
        y.append(ii)
    plt.plot(x, y)
    plt.savefig('CaseIIDecisionBoundary.png')


# plotData()
# caseIIdecisionRule()
plotTestData()
caseIIdecisionRule()
```

```
#*****************************************************************
#Case III
#Evaluating the classifiers:
#*****************************************************************


E0= y0Cov
E1 = y1Cov
E0_inv = np.linalg.inv(E0)
E1_inv = np.linalg.inv(E1)
u0 = np.array([y0x1Mean, y0x2Mean])
u1 = np.array([y1x1Mean,y1x2Mean])

def g0(x, y, w):
    X = np.array([x,y])
    LHS1A= -.5*E0_inv
    LHS1 = np.dot(np.dot(np.transpose(X),LHS1A),X)
    LHS2 = np.dot(np.transpose(np.dot(E0_inv,u0)),X)
    LHS3A= np.dot(np.dot(np.transpose(u0),E0_inv),u0)
    LHS3B= np.log(np.linalg.det(E0))
    LHS3 = -.5*(LHS3A+LHS3B)
    LHS = LHS1 + LHS2 + LHS3 + math.log(w)
    return LHS

def g1(x, y, w):
    X = np.array([x,y])
    RHS1A= -.5*E1_inv
    RHS1 = np.dot(np.dot(np.transpose(X),RHS1A),X)
    RHS2 = np.dot(np.transpose(np.dot(E1_inv,u1)),X)
    RHS3A= np.dot(np.dot(np.transpose(u1),E1_inv),u1)
    RHS3B= np.log(np.linalg.det(E1))
    RHS3 = -.5*(RHS3A+RHS3B)
    RHS = RHS1 + RHS2 + RHS3 + math.log(1-w)
    return RHS

accuracy_array=EvaluatePriorProbs(Ws,test_data)
plotAccuracyCurve(Ws,accuracy_array,'CaseIIIAccuracy.png')


print("With Equal Prior Probability, accuracy: ",
 ↪  classifyAndEvaluate(test_data,.5))
classifyAndEvaluateWithWrongDisplay(test_data,.5,g0,g1)

#*****************************************************************
#True Postive Rate/True Negative Rate for Case III
```

```
#*************************************************************


accuracy_array=EvaluatePriorProbsConfusionMatrix(Ws,test_data,g0,g1)

#*************************************************************
#Case 3
# Finding decision boundary:
#*************************************************************


E0= y0Cov
E1 = y1Cov
E0_inv = np.linalg.inv(E0)
E1_inv = np.linalg.inv(E1)
u0 = np.array([y0x1Mean, y0x2Mean])
u1 = np.array([y1x1Mean,y1x2Mean])
x1 = sym.Symbol('x1')
x2 = sym.Symbol('x2')
X = np.array([x1,x2])

LHS1A= -.5*E0_inv
LHS1 = np.dot(np.dot(np.transpose(X),LHS1A),X)
LHS2 = np.dot(np.transpose(np.dot(E0_inv,u0)),X)
LHS3A= np.dot(np.dot(np.transpose(u0),E0_inv),u0)
LHS3B= np.log(np.linalg.det(E0))
LHS3 = -.5*(LHS3A+LHS3B)

LHS = LHS1 + LHS2 + LHS3

RHS1A= -.5*E1_inv
RHS1 = np.dot(np.dot(np.transpose(X),RHS1A),X)
RHS2 = np.dot(np.transpose(np.dot(E1_inv,u1)),X)
RHS3A= np.dot(np.dot(np.transpose(u1),E1_inv),u1)
RHS3B= np.log(np.linalg.det(E1))
RHS3 = -.5*(RHS3A+RHS3B)

RHS = RHS1 + RHS2 + RHS3

eq1 = LHS + (-1*RHS)
sol= sym.solve(eq1,x2)

print("Rule 3 - Classification boundary: ", sol[0])

#Case III
#Ploting decision boundary:
```

```
def caseIIIdecisionRule():
    x= np.arange(-1.5,1.4,.1)
    y=[]
    for i in x:
        ii = sol[0].subs(x1,i)
        y.append(ii)
    plt.plot(x,y)
    plt.savefig('CaseIIIDecisionBoundary.png')

plotTestData()
caseIIIdecisionRule()

#################################################################
#Plotting all decision boundaries together:
#################################################################


plotData()

x= np.arange(-1.5,2,.1)
y=[]
for i in x:
    ii=-0.8326*i+0.4438
    y.append(ii)
plt.plot(x,y)

y=[]
for i in x:
    ii=0.467636629718861 - 0.504759841995623*i
    y.append(ii)
plt.plot(x,y)

x= np.arange(-1.5,1.4,.1)
y=[]
for i in x:
    ii = sol[0].subs(x1,i)
    y.append(ii)
plt.plot(x,y)
plt.savefig('AllCasesDecisionBoundary.png')


#################################################################
# Define the two gaussian discriminants and calcualte accuracy. (Equal Prior
↪  probability)
#################################################################
```

```python


def g0(x, y, w):
    mu1 = np.array([-0.75, 0.2]);
    mu2 = np.array([0.3, 0.3]);
    S1 = np.array([[0.25, 0], [0, 0.3]]);
    S2 = np.array([[0.1, 0], [0, 0.1]]);
    A1 = 0.8;
    A2 = 1 - A1;
    d = 2
    S1_inv = np.linalg.inv(S1)
    S2_inv = np.linalg.inv(S2)
    X = np.array([x, y])

    p1a = (2 * math.pi) ** (d / 2)
    p1b = A1 / ((np.linalg.det(S1)) ** (1 / 2) * p1a)
    p1c = np.exp((-1 / 2) * (np.dot(np.dot(np.transpose((X - mu1)), S1_inv), (X -
     ↪  mu1))))
    p1 = p1b * p1c

    p2a = (2 * math.pi) ** (d / 2)
    p2b = A2 / ((np.linalg.det(S2)) ** (1 / 2) * p2a)
    p2c = np.exp((-1 / 2) * (np.dot(np.dot(np.transpose((X - mu2)), S2_inv), (X -
     ↪  mu2))))
    p2 = p2b * p2c

    return p2 + p1


def g1(x, y, w):
    # mu1 = np.array([0.38, 0.70])
    # mu2 = np.array([-0.29, 0.69])
    mu1 = np.array([-0.31, 0.75])
    mu2 = np.array([0.48, 0.65])
    S1 = np.array([[0.03, 0], [0, 0.029]])
    S2 = np.array([[0.029, 0], [0, 0.28]])
    A1 = 0.8;
    A2 = 1 - A1;
    d = 2
    S1_inv = np.linalg.inv(S1)
    S2_inv = np.linalg.inv(S2)

    X = np.array([x, y])

    p1a = (2 * math.pi) ** (d / 2)
```

```python
    p1b = A1 / ((np.linalg.det(S1)) ** (1 / 2) * p1a)
    p1c = np.exp((-1 / 2) * (np.dot(np.dot(np.transpose((X - mu1)), S1_inv), (X -
     ↪  mu1))))
    p1 = p1b * p1c

    p2a = (2 * math.pi) ** (d / 2)
    p2b = A2 / ((np.linalg.det(S2)) ** (1 / 2) * p2a)
    p2c = np.exp((-1 / 2) * (np.dot(np.dot(np.transpose(X - mu2), S2_inv), (X -
     ↪  mu2))))
    p2 = p2b * p2c

    return p2 + p1


print("Two-Modal Gaussian, accuracy: ", classifyAndEvaluate(test_data, .5))

#****************************************************************
#Prints the Gaussian figure in 3d (P1)
#****************************************************************




fig = plt.figure(figsize=(12,12))
ax = fig.add_subplot(111, projection='3d')

x = np.linspace(-1.5, 1, 30)
y = np.linspace(-.5, 2.5, 30)
X, Y = np.meshgrid(x, y)
Z = np.vectorize(g0)
Z2= np.vectorize(g1)

ax.plot_surface(X, Y, Z(X,Y,.5))


ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('y');
plt.savefig('TwoModal0.png')

#****************************************************************
#Prints the Gaussian figure in 3d (P2)
#****************************************************************


fig = plt.figure(figsize=(12,12))
```

```
ax = fig.add_subplot(111, projection='3d')

x = np.linspace(-1.5, 1, 30)
y = np.linspace(-.5, 2.5, 30)
X, Y = np.meshgrid(x, y)
Z = np.vectorize(g0)
Z2= np.vectorize(g1)

ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('y');

ax.plot_surface(X, Y, Z2(X,Y,.5), color='r')
plt.savefig('TwoModal1.png')
```