Project 1: Two Category Classification Using Baysian Decision Rule

Adrian Cross

November 25, 2019

across19@vols.utk.edu
Machine learning
COSC 522

1 Abstract

In this project a decision rule was determined on a synthetic data set (Ripley and Hjort 1996) with two categories which were assumed to have Gaussian probability densities. Using a maximum likelihood estimation (MLE) method on a training data set the mean ($\overrightarrow{\mu}$) and covariance matrices ($\overrightarrow{\Sigma}$) were calculated for both categories. Using the maximum a posteriori probability (MAP) method on these parameters three decision rules were created, based on different mathematical assumptions, with different results on the data clarification. These different classifications were then extensively tested with a testing dataset and evaluated using classification accuracy as a performance metric. It was found that, depending on the assumptions made, the percentage of correctly classified points increased from 68.0% to 90.0% for class 0 and 74.9% increased to 88.4% for class 1 assuming a equal prior probability. The result was then compared to a two Gaussian peak distribution.

2 Introduction

In this project a Bayesian approach has been taken to apply decision rules to specific data sets. The data used is a collection of data sets used in Ripley and Hjort (1996) which has two real-valued coordinates,xs (x) and ys (y), which are used in a multivariate analysis alongside a class, xc, which can either be 0 or 1. Therefore this project is based upon a multivariate data set with two categories. the training data set, synth.tr, contains 250 data points used to train a decision rule whereas the testing data set, synth.te, is used to test and evaluate the decision rule for accuracy.

Thee data can be modelled in a variety of ways. A popular way of modelling data is using a Gaussian distribution. This distribution can be modelled from one up to d dimensions however, in this study, a multivariate Gaussian with two dimensions is modelled. Equation 1 gives the equation based on specific input parameters.

$$p(x|\overrightarrow{\mu}, \overrightarrow{\Sigma}) = \frac{1}{(2\pi)^{\frac{d}{2}}|\overrightarrow{\Sigma}|^{\frac{1}{2}}} exp(-\frac{1}{2}(\overrightarrow{x} - \overrightarrow{\mu})^T \overrightarrow{\Sigma}^{-1}(\overrightarrow{x} - \overrightarrow{\mu})) \tag{1}$$

Where $\overrightarrow{\mu}$ and $\overrightarrow{\Sigma}$ are measures of the mean and standard deviation for different coordinates in each class respectively, determined by a MLE method. By modelling each class as an independent, multivariate Gaussian function $\overrightarrow{\mu}$ and $\overrightarrow{\Sigma}$ can be modelled using a MLE method, which can then be used to calculate decision boundaries between the two classes.

3 Maximum likelihood estimation (MLE)

3.1 $\overrightarrow{\mu}$ Calculation

The $\overrightarrow{\mu}$ vector is simply a vector of the mean average of the coordinates. Therefore for the synth.tr data set it is simply a $2 * 1$ matrix containing the mean average of the x and y coordinate as shown in equation 2.

$$\overrightarrow{\mu} = \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix} \tag{2}$$

with each component being calculated by summing up all the x or y data points in that class and dividing by the total number of data points, N, as shown in equation 3.

$$\mu_c = \frac{1}{N}\sum_{j=1}^{N} c_{(j)} \tag{3}$$

Where $c = x$ or y and j refers to a specific data point. Applying equation 3 to the data gives two $\overrightarrow{\mu}$ vectors, one for each category.

3.2 $\overrightarrow{\Sigma}$ Calculation

The $\overrightarrow{\Sigma}$ matrix is a $d * d$ matrix, in this case a $2 * 2$ matrix due to being a two dimensional problem, which contains information on the standard deviations for each coordinate as well as the correlation between the coordinates. the

$\overrightarrow{\Sigma}$ matrix is shown below.

$$\overrightarrow{\Sigma} = \begin{pmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{yx} & \sigma_{yy} \end{pmatrix} \tag{4}$$

Where σ_{xx} and σ_{yy} are the standard deviations for the x/y coordinates and σ_{xy}/σ_{yx} are mixing factors which depend on the correlation between x and y. If x and y are statistically independent then $\sigma_{xy} = \sigma_{yx} = 0$. The general form to calculate this matrix is given in equation 7.

$$\overrightarrow{\Sigma} = \frac{1}{N} \sum_{j=1}^{N} (c_{(j)} - \overrightarrow{\mu})(c_{(j)} - \overrightarrow{\mu})^T \tag{5}$$

In order to calculate σ_{xx} and σ_{yy} it is simply a case of calculating the standard deviation for the x and y coordinates, essentially summing up the squared differences between the data point and the mean and dividing by the total number of data points.

$$\overrightarrow{\sigma_{cc}} = \frac{1}{N} \sum_{j=1}^{N} (c_{(j)} - \overrightarrow{\mu_c})^2 \tag{6}$$

To calculate σ_{xy} and σ_{yx} a similar approach is required, except now you sum the multiple of the differences between the data point and the mean for x and μ_x and y and μ_y (Duda et al. 2012).

$$\overrightarrow{\sigma_{ik}} = \frac{1}{N} \sum_{j=1}^{N} (c_{(j)} - \overrightarrow{\mu_c})(k_{(j)} - \overrightarrow{\mu_k}) \tag{7}$$

3.3 Results

Using the methods outlines in section 3.1 and 3.2 $\overrightarrow{\mu}$ and $\overrightarrow{\Sigma}$ matrices have been constructed for both categories and are shown below.

Class 0:

$$\overrightarrow{\mu} = \begin{pmatrix} -0.22147024 \\ 0.32575494 \end{pmatrix} \qquad \overrightarrow{\Sigma} = \begin{pmatrix} 0.27459508 & 0.01113883 \\ 0.01113883 & 0.03583011 \end{pmatrix}$$

Class 1:

$$\overrightarrow{\mu} = \begin{pmatrix} 0.07595431 \\ 0.68296891 \end{pmatrix} \qquad \overrightarrow{\Sigma} = \begin{pmatrix} 0.15846988 & -0.01545041 \\ -0.01545041 & 0.02971875 \end{pmatrix}$$

By using equation 1 a multivariate Gaussian has been plotted. This Gaussian contains the two coordinate axis with a probability density in the z direction. a multivariate Gaussian has been plotted for each graph as shown below. From this a clear Gaussian can be seen with a peak at $\overrightarrow{\mu}$ for each of the categories.

4 Decision boundaries

4.1 Maximum a posteriori probability (MAP) method

The maximum a posteriori probability method determines which class a data point should belong to. It does this by first calculating the posterior probability for a given coordinate c using equation 8.

$$P(\omega_j|c) = \frac{P(c|\omega_j)P(\omega_j)}{P(c)} \tag{8}$$

Where $P(c|\omega_j)$ is the conditional probability density function, $P(\omega_j)$ is the prior probability and $P(x)$ is a normalization constant. When the posterior probability is calculated the class with the larger number is the class which that data point belongs to. i.e. for a given c, if $P(\omega_0|c) > P(\omega_1|c)$, then c belongs to class 0, otherwise it belongs to class 1. The pattern classifier can be defined via a discriminant function given in equation 10.

$$g_i(\overrightarrow{c}) = lnp(\overrightarrow{c}|\omega_i) + lnP(\omega_i) \tag{9}$$

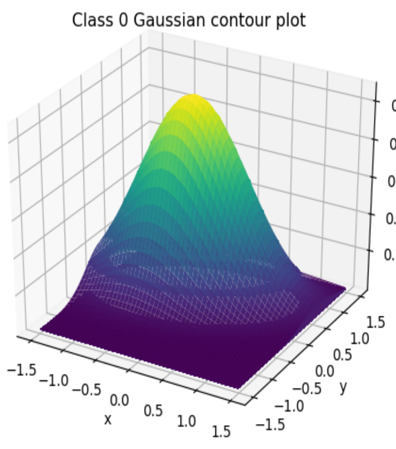


Figure 1: Class 0 Gaussian contour plot

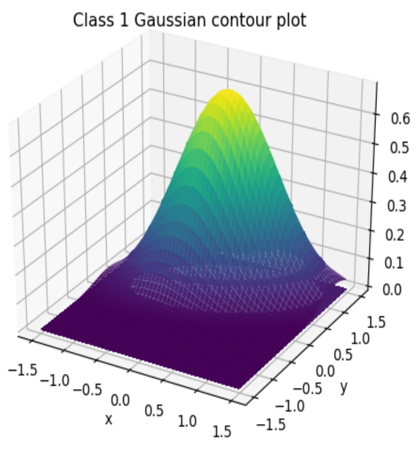


Figure 2: Class 1 Gaussian contour plot

Using equation 1 with the definition for the discriminant function a generalized $g_i(x, y)$ can be found.

$$g_i(\overrightarrow{c}) = -\frac{1}{2}(\overrightarrow{c} - \overrightarrow{\mu_i})^T \Sigma_i^{-1}(\overrightarrow{c} - \overrightarrow{\mu_i}) - \frac{1}{2}ln|\Sigma_i| + lnP(\omega_i) \tag{10}$$

The form of the classifier can be simplified depending upon assumption are made about the data points. Three different cases with different assumptions are considered in sections 4.2, 4.3 and 4.4.

4.2 Case 1: $(\overrightarrow{\Sigma_i} = \sigma^2 \overrightarrow{I})$

This case assumes that x and y are statistically independent and have the same variance. By taking a determinant of both Σ_i and taking the mean average a value is obtained which is equal to σ^2. Using this value equation 10 can be reduced down to equation 11.

$$g_i(\overrightarrow{c}) = \frac{\overrightarrow{\mu_i^T}}{\sigma^2}\overrightarrow{x} - \frac{\overrightarrow{\mu_i^T}\overrightarrow{\mu_i}}{2\sigma^2} + lnP(\omega_i) \tag{11}$$

Setting the $g_i(x, y)$ value for case 0 equal to case 1 defines a hyperplane between the two Gaussian plots. By observing the graph on only the x and y axis a straight line is defined by equation **??** (Duda et al. 2012).

$$y = -x\frac{\mu_{x0} - \mu_{x1}}{\mu_{y0} - \mu_{y1}} + c_x(\mu_{x0} - \mu_{x1}) + c_y(\mu_{y0} - \mu_{y1}) \tag{12}$$

Where c_x and c_y are defined as $\overrightarrow{c_0} = \begin{pmatrix} c_x \\ c_y \end{pmatrix} = \frac{1}{2}(\overrightarrow{\mu_0} + \overrightarrow{\mu_1}) - \frac{\sigma^2}{||\overrightarrow{\mu_0} - \overrightarrow{\mu_1}||^2}\frac{lnP(\omega_0)}{lnP(\omega_1)}(\overrightarrow{\mu_0} - \overrightarrow{\mu_1})$. From this definition a line with equation $y = -0.832622948392767x0.44378197841356015$ is found and is shown in figure **??**. The full derivation can be found in appendix B.1.

4.3 Case 2: $(\overrightarrow{\Sigma_i} = \overrightarrow{\Sigma})$

This case assumes that each coordinate has a different standard deviation, but that each class has the same standard deviation so that $\overrightarrow{\Sigma}$ is the same in all classes. In order to practically get the value of $\overrightarrow{\Sigma}$ a simple mean average is taken so that $\overrightarrow{\Sigma} = \frac{1}{2}(\overrightarrow{\Sigma_0} + \overrightarrow{\Sigma_1})$. Using this an equation for the classifier can be derived similarly to the classifier shown in equation 11.

$$g_i(\overrightarrow{c}) = -\frac{1}{2}(x - \overrightarrow{\mu_i})^t \overrightarrow{\Sigma_i^{-1}}(x - \overrightarrow{\mu_i}) + lnP(\omega_i) \tag{13}$$

Using this a similar linear equation to the one shown in section 4.2 can be derived (Duda et al. 2012).

$$y = -x\frac{\sigma_{xx}(\mu_{x0} - \mu_{x1}) + \sigma_{yx}(\mu_{y0} - \mu_{y1})}{\sigma_{yy}(\mu_{y0} - \mu_{y1}) + \sigma_{xy}(\mu_{x0} - \mu_{x1})} + c_y + \frac{\sigma_{xx}(\mu_{x0} - \mu_{x1}) + \sigma_{yx}(\mu_{y0} - \mu_{y1})}{\sigma_{yy}(\mu_{y0} - \mu_{y1}) + \sigma_{xy}(\mu_{x0} - \mu_{x1})}c_x \tag{14}$$

Where c_x and c_y are defined as $\overrightarrow{c_0} = \begin{pmatrix} c_x \\ c_y \end{pmatrix} = \frac{1}{2}(\overrightarrow{\mu_0} + \overrightarrow{\mu_1}) - \frac{1}{(\overrightarrow{\mu_0}-\overrightarrow{\mu_1})^t \overrightarrow{\Sigma_i^{-1}}(\overrightarrow{\mu_0}-\overrightarrow{\mu_1})} \frac{lnP(\omega_0)}{lnP(\omega_1)}(\overrightarrow{\mu_0} - \overrightarrow{\mu_1})$. Putting in values gives an equation of $y = -0.13486408662390312x + 0.4945494908841989$. The full derivation can be found in appendix B.2.

4.4 Case 3: $(\overrightarrow{\Sigma_i} = arbitrary)$

$$g_i(\overrightarrow{c}) = -\frac{1}{2}(x - \overrightarrow{\mu_i})^t \overrightarrow{\Sigma_i^{-1}}(x - \overrightarrow{\mu_i}) - \frac{1}{2}ln|\overrightarrow{\sigma_i}| + lnP(\omega_i) = \overrightarrow{c^t} W_i \overrightarrow{c} + w_i^t \overrightarrow{c} + c_{0i} \tag{15}$$

Where $W_i = -\frac{1}{2}\Sigma_i^{-1}$, $c_{0i} = -\frac{1}{2}\overrightarrow{\mu_i^t}\Sigma_i^{-1}\overrightarrow{\mu_i} - \frac{1}{2}ln|\Sigma_i| + lnP(\omega_i)$ and $w_i = \Sigma_i^{-1}\overrightarrow{\mu_i}$. Setting $g_0(\overrightarrow{c}) = g_1(\overrightarrow{c})$ and rearranging a quadratic boundary is found. This derivation is found in appendix B.3.
These three boundaries have been plotted in figure 3. From this you can see how the different assumptions affect the location of the boundary.

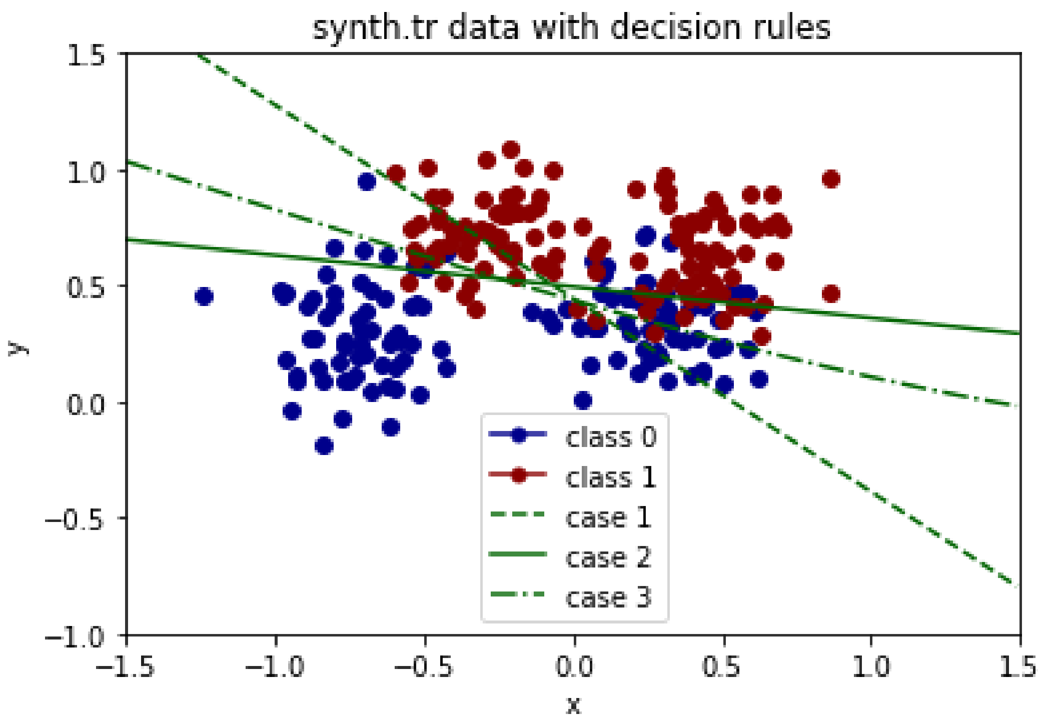


Figure 3: Plot showing the various boundary's on the training data set.

5 Evaluating accuracy of boundaries

5.1 Equal prior probabilities

By now applying the testing data file the efficiency of the different boundaries can be tested. The synth.te file has now been used to test the derived boundaries. figure 4 shows a graph of the same boundaries using testing data instead.

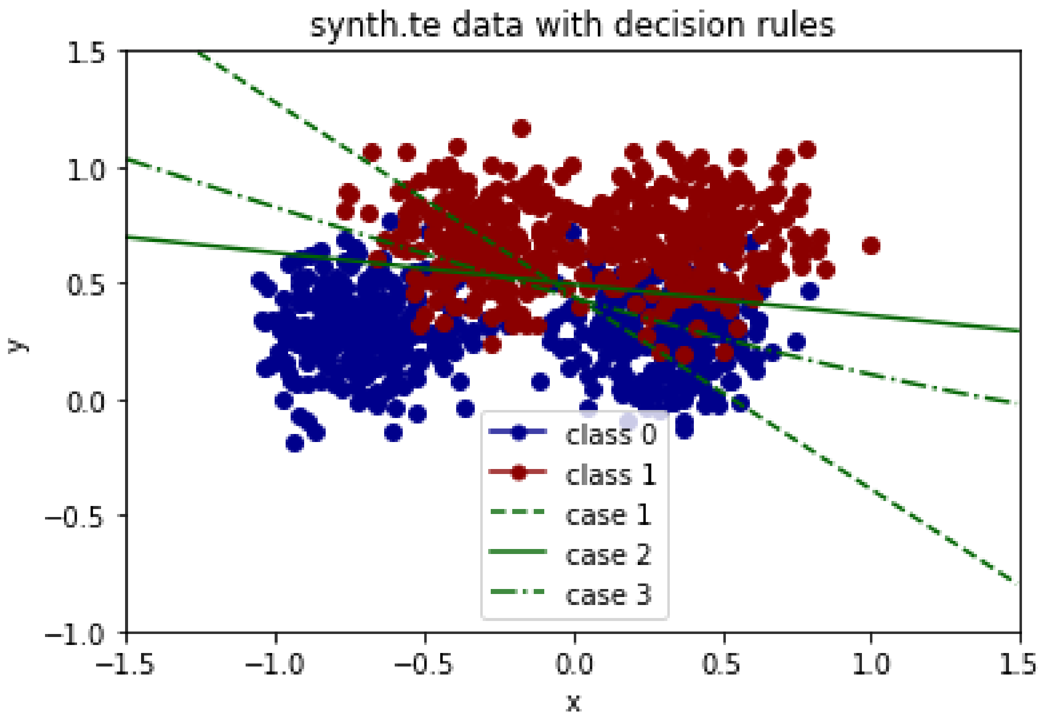
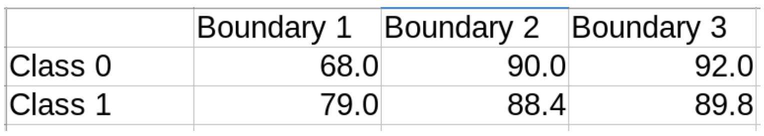


Figure 4: Plot showing the various boundary's on the testing data set with prior probabilities of 0.5.

Using these boundaries, the points can be sorted whether they are classified correctly or not. If a data point is taken it is checked to see whether it is above or below each boundary line. If it is above the line it is assumed to be in class 1, if it is below it is assumed to be in class 0. The data points are then compared to the actual class and if they are classified correctly they are included in the correct classification percentage. The classification percentages, for equal prior probabilities of 0.5, are shown in figure 5.



	Boundary 1	Boundary 2	Boundary 3
Class 0	68.0	90.0	92.0
Class 1	79.0	88.4	89.8

Figure 5: Classification percentages for testing data

From this table it can be seen that the percentage acceptance tends to increase with the fewer assumptions used in each case. In particularly you can see that Boundary three has the best classification accuracy and this method made the least assumptions.

5.2 Variable prior probabilities

By varying the prior probabilities the equations of the boundary lines as their equations are dependent upon these quantities. figure 6 shows the prior probabilities with a value of 0.9 for class 0 and 0.1 for class 1.

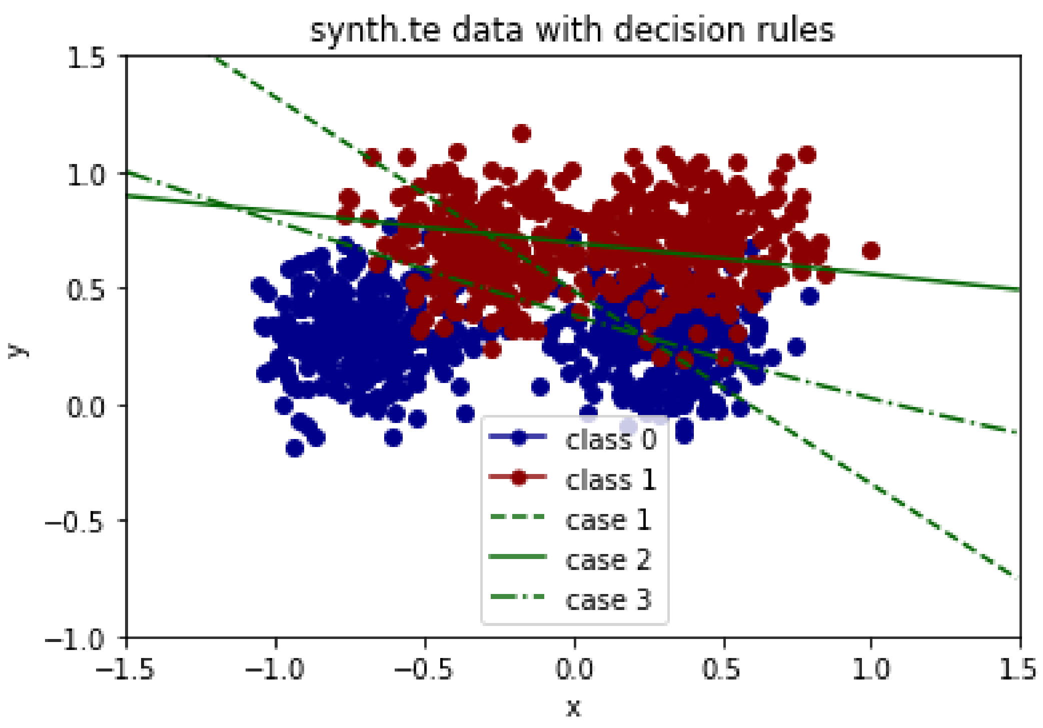
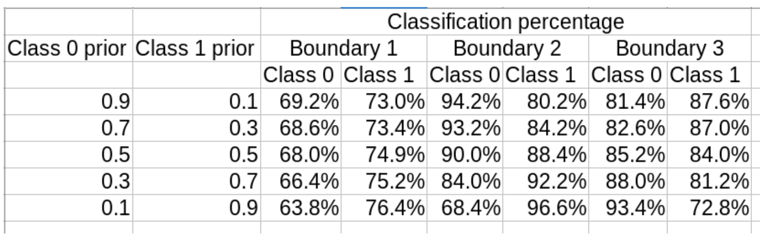


Figure 6: Plot showing the various boundary's on the testing data set with prior probabilities of 0.9 for class 0 and 0.1 for class.

By varying these quantities the classification percentages can also be looked at. figure 7 shows a table of these varied prior probabilities and their affect on the mean classification percentages between the two classes.



		Classification percentage					
Class 0 prior	Class 1 prior	Boundary 1		Boundary 2		Boundary 3	
		Class 0	Class 1	Class 0	Class 1	Class 0	Class 1
0.9	0.1	69.2%	73.0%	94.2%	80.2%	81.4%	87.6%
0.7	0.3	68.6%	73.4%	93.2%	84.2%	82.6%	87.0%
0.5	0.5	68.0%	74.9%	90.0%	88.4%	85.2%	84.0%
0.3	0.7	66.4%	75.2%	84.0%	92.2%	88.0%	81.2%
0.1	0.9	63.8%	76.4%	68.4%	96.6%	93.4%	72.8%

Figure 7: Table showing how the variation of prior probabilities affects the classification percentage

From this table a clear trend can be seen that the classification percentage increases with the prior probability. Therefore as one class classification percentage is increasing with its prior probability, the other classes classification percentage is decreasing with its class decreasing prior probability.

6 Two modal Gaussian

The data in this study has been modelled using a one modal Gaussian. By modelling the data using a two modal Gaussian. A two modal Gaussian includes the use of two separate Gaussian peaks. figure 8 shows a contour plot of a two modal Gaussian.

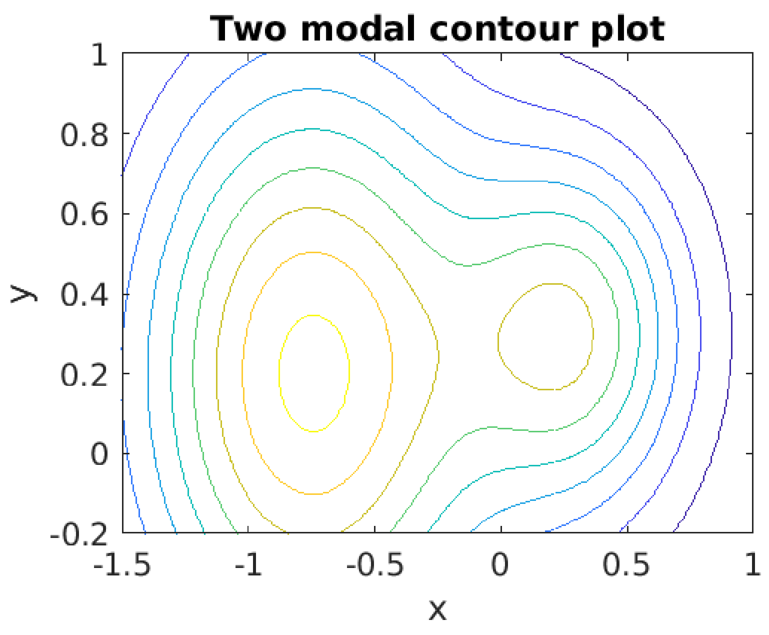


Figure 8: Contour plot of a two modal Gaussian

7 Conclusion

In this report a set of training data with two dimensions, x and y, been take with two different classes. Using the training dataset a calculation for the covariance matrix and the two dimension's means has been completed, using the MLE method. This method of calculation has provided an accurate result for the $\vec{\mu}$ vector and Σ matrix for both classes. Using these estimates, three different decision boundaries have been derived, based on assumptions made about the correlation between dimensions. From the classification percentages, which were used to evaluate the data, The three boundaries have been evaluated for their accuracy in classifying data. This classification has also been tested for variable prior probabilities.
Future studies of this nature could include a more depth study on how the classification percentages are affected by varying prior probabilities, including looking at cases where there are more than three classes and therefore how these classification percentages vary with different class numbers. This model also assumes a Gaussian distribution, however the data could follow different distributions more accurately. Therefore a similar study looking at the the data using different distributions could yield more accurate results.

A Code

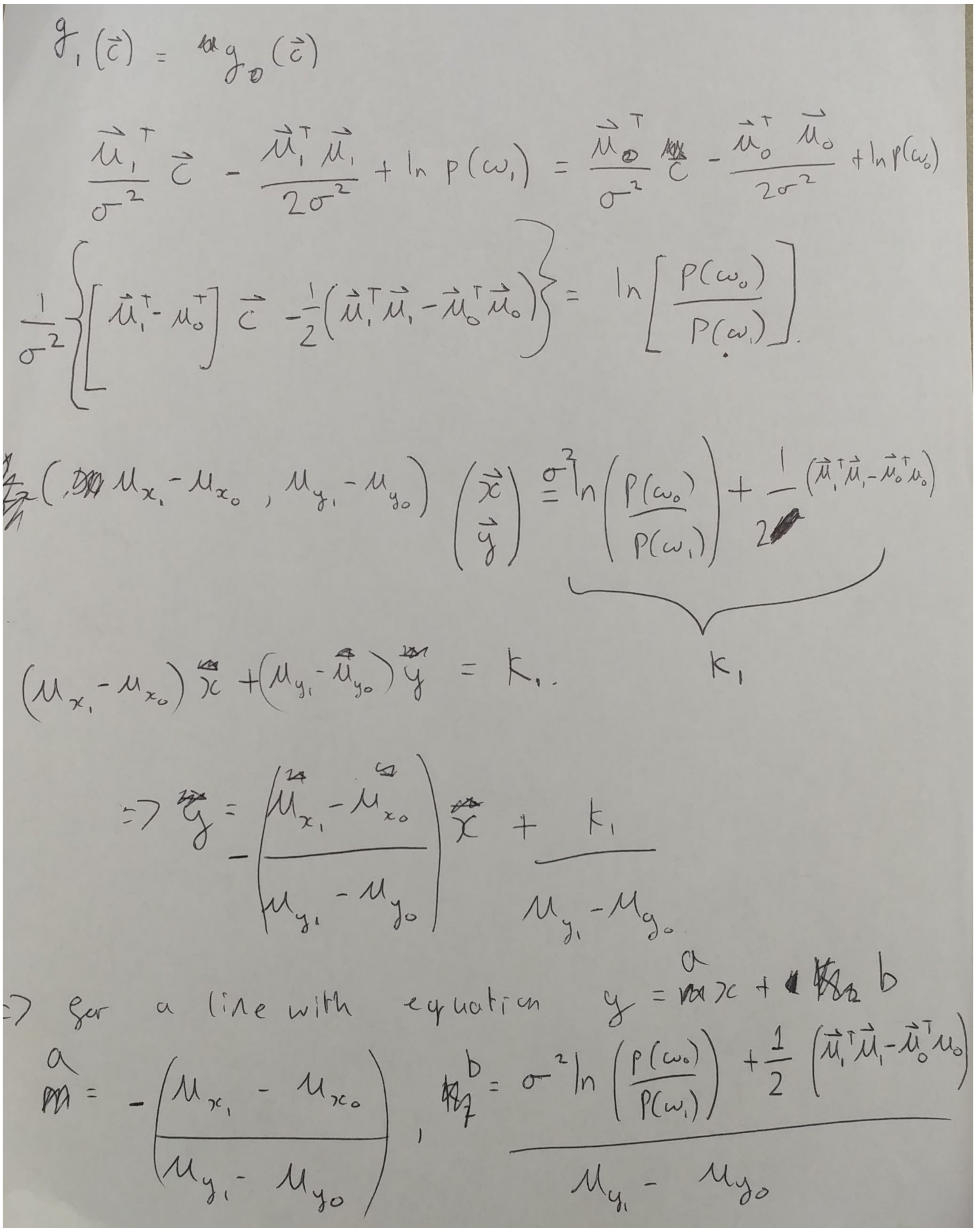
```
import sys
import math
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.lines as mlines
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits import mplot3d

pi= 3.14159265359
def calc_mu(data_tr,coordinate):
    n=count_row=data_tr.shape[0]#total number of data points calculation
    return (1/n)*data_tr[coordinate].sum()

def calc_sigma(data_tr,coordinate1,coordinate2):
    n=count_row=data_tr.shape[0]#total number of data points calculation
    a=data_tr[coordinate1]-calc_mu(data_tr,coordinate1)
    b=data_tr[coordinate2]-calc_mu(data_tr,coordinate2)
    #return (1/n)*((data_tr[coordinate1]-calc_mu(data_tr,coordinate1))*(data_tr[coordinate2]-
    calc_mu(data_tr,coordinate2))).sum()
    return (1/n)*(a*b).sum()

```

```
def gaussian_pdf(x,mu,sigma): #calculates p(x|mu,sigma) for x1 and x2
    x=x.reshape(-1,1)
    mu=mu.reshape(-1,1)
    p,_=sigma.shape

    sigma_inv=np.linalg.inv(sigma)#determinant of sigma
    a=np.sqrt((2*pi)**p*np.linalg.det(sigma)) #first part of gaussian
    b=-0.5*((x-mu).T @ sigma_inv @ (x-mu)) #second part of gaussian
    return float (1./a)*np.exp(b)

def input_data(filename):
    data = pd.read_csv(filename, sep='\s+', header = 0)
    return data

def boundary_1(x,mu_0,mu_1, sigma_c0, sigma_c1,P_w0,P_w1):
    xcoeff=-(mu_1[0]-mu_0[0])/(mu_1[1]-mu_0[1])
    sigma_c0_b1=([sigma_c0[0,0],0],[0,sigma_c0[1,1]])
    sigma_c1_b1=([sigma_c1[0,0],0],[0,sigma_c1[1,1]])
    sigma=0.5*(np.linalg.det(sigma_c0_b1)+np.linalg.det(sigma_c1_b1))#sigma is the average of
    2 determinants
    y_intercept=(1/(mu_1[1]-mu_0[1]))*(sigma*np.log(P_w0/P_w1)+0.5*(np.transpose(mu_1)@mu_1-np
    .transpose(mu_0)@mu_0))
    print('Boundary 1: y=',xcoeff,'x+',y_intercept)
    return xcoeff*x+y_intercept

def boundary_2(x,mu_0,mu_1, sigma_c0, sigma_c1,P_w0,P_w1):
    sigma=0.5*(sigma_c0+sigma_c1)
    sigma=np.linalg.inv(sigma)
    a=sigma[0,0]*(mu_1[0]-mu_0[0])+sigma[1,0]*(mu_1[1]-mu_0[1])
    b=sigma[0,1]*(mu_1[0]-mu_0[0])+sigma[1,1]*(mu_1[1]-mu_0[1])
    xcoeff=-a/b
    w_x=sigma[0,0]*(mu_0[0]-mu_1[0])+sigma[0,1]*(mu_0[1]-mu_1[1])
    w_y=sigma[1,1]*(mu_0[1]-mu_1[1])+sigma[1,0]*(mu_0[0]-mu_1[0])
    m_minus=mu_0-mu_1
    factor=1/((np.transpose(m_minus))@sigma@m_minus)
    c0=0.5*(mu_0+mu_1)-m_minus*factor*np.log(P_w0/P_w1)
    y_intercept=c0[1]+c0[0]*(w_x/w_y)
    #y_intercept=(w_x/w_y)*c0[0]
    print('Boundary 2: y=',xcoeff,'x+',y_intercept)
    return xcoeff*x+y_intercept

def boundary_3(x,mu_c0,mu_c1, sigma_c0, sigma_c1,P_w0,P_w1):
    sigma_minus=np.linalg.inv(sigma_c1)-np.linalg.inv(sigma_c0)
    k=0.5*np.transpose(mu_c1)@np.linalg.inv(sigma_c1)@mu_c1-0.5*np.transpose(mu_c0)@np.linalg.
    inv(sigma_c0)@mu_c0+0.5*np.log(np.linalg.det(sigma_c1)/np.linalg.det(sigma_c0))-np.log(
    P_w1/P_w0)
    mu_sigma_0=np.transpose(mu_c0)@np.linalg.inv(sigma_c0)
    mu_sigma_1=np.transpose(mu_c1)@np.linalg.inv(sigma_c1)
    m=(mu_sigma_1-mu_sigma_0)
    A=m[0]
    B=m[1]
    numerator=-np.sqrt(4*sigma_minus[1][1]*(A*x-sigma_minus[0][0]*x**2+k)+(B-2*sigma_minus
    [1][0]*x)**2)+B-2*x*sigma_minus[1][0]
    #numerator=-np.sqrt(4*sigma_minus[1][1]*A*x-4*sigma_minus[1][1]*sigma_minus[0][0]*x**2+4*
    sigma_minus[1][1]*k+B**2-4*B*x+4*x**2)+B-2*x
    return numerator/(2*sigma_minus[1][1])

def class_check(boundary_type,df,mu_0,mu_1, sigma_c0, sigma_c1,P_w0,P_w1=0.5): # checks which
    class a point belongs to
    yline=0
    if boundary_type=='bound_1':
        yline=boundary_1(df['xs'],mu_0,mu_1, sigma_c0, sigma_c1,P_w0,P_w1=0.5)
    if boundary_type=='bound_2':
        yline=boundary_2(df['xs'],mu_0,mu_1, sigma_c0, sigma_c1,P_w0,P_w1=0.5)
    if boundary_type=='bound_3':
        yline=boundary_3(df['xs'],mu_0,mu_1, sigma_c0, sigma_c1,P_w0,P_w1=0.5)

    cond = df['ys'] > yline #if true then belongs to class 1, if false then class 0
    df[boundary_type] = np.where(cond, 1, 0)

data_tr=input_data('synth.tr')
data_tr_c0=data_tr.loc[data_tr.yc==0]# splits data into 2 classes
data_tr_c1=data_tr.loc[data_tr.yc==1]
```

```python
#data_tr_c1=data_tr_c1.reset_index()
data_tr_c1.drop(columns='yc',axis=1,inplace=True)

mu_c0=np.array([calc_mu(data_tr_c0,'xs'),calc_mu(data_tr_c0,'ys')])#mu calculation
mu_c1=np.array([calc_mu(data_tr_c1,'xs'),calc_mu(data_tr_c1,'ys')])

sigma_c0=np.array([[calc_sigma(data_tr_c0,'xs','xs') , calc_sigma(data_tr_c0,'xs','ys')], [calc_sigma(data_tr_c0,'ys','xs'),  calc_sigma(data_tr_c0,'ys','ys')]])#sigma calculation
sigma_c1=np.array([[calc_sigma(data_tr_c1,'xs','xs') , calc_sigma(data_tr_c1,'xs','ys')], [calc_sigma(data_tr_c1,'ys','xs'),  calc_sigma(data_tr_c1,'ys','ys')]])

x=np.arange(-1.5,1.5,0.01)

print('mu_c0= ',mu_c0)
print('mu_c1= ',mu_c1)
print('sigma_c0= \n',sigma_c0)
print('sigma_c1= \n',sigma_c1)

data_tr=input_data('synth.te')
data_tr_c0=data_tr.loc[data_tr.yc==0]# splits data into 2 classes
data_tr_c1=data_tr.loc[data_tr.yc==1]

P_w0=0.1
P_w1=0.9
class_check('bound_1',data_tr_c0,mu_c0,mu_c1, sigma_c0, sigma_c1,P_w0,P_w1) #counts points above/below boundary
class_check('bound_2',data_tr_c0,mu_c0,mu_c1, sigma_c0, sigma_c1,P_w0,P_w1)
class_check('bound_3',data_tr_c0,mu_c0,mu_c1, sigma_c0, sigma_c1,P_w0,P_w1)
class_check('bound_1',data_tr_c1,mu_c0,mu_c1, sigma_c0, sigma_c1,P_w0,P_w1)
class_check('bound_2',data_tr_c1,mu_c0,mu_c1, sigma_c0, sigma_c1,P_w0,P_w1)
class_check('bound_3',data_tr_c1,mu_c0,mu_c1, sigma_c0, sigma_c1,P_w0,P_w1)

print('Class 0 correct classification percentage')
print('boundary 1: ',100*data_tr_c0.loc[data_tr_c0.bound_1==0].shape[0]/data_tr_c0.shape[0])#calculates % acceptence
print('boundary 2: ',100*data_tr_c0.loc[data_tr_c0.bound_2==0].shape[0]/data_tr_c0.shape[0])
print('boundary 3: ',100*data_tr_c0.loc[data_tr_c0.bound_3==0].shape[0]/data_tr_c0.shape[0])

print('Class 1 correct classification percentage')
print('boundary 1: ',100*data_tr_c1.loc[data_tr_c1.bound_1==1].shape[0]/data_tr_c1.shape[0])#calculates % acceptence
print('boundary 2: ',100*data_tr_c1.loc[data_tr_c1.bound_2==1].shape[0]/data_tr_c1.shape[0])
print('boundary 3: ',100*data_tr_c1.loc[data_tr_c1.bound_3==1].shape[0]/data_tr_c1.shape[0])

plt.plot(x,boundary_1(x,mu_c0,mu_c1, sigma_c0, sigma_c1,P_w0,P_w1),c='DarkGreen',linestyle='--')
plt.plot(x, boundary_2(x,mu_c0,mu_c1, sigma_c0, sigma_c1,P_w0,P_w1),c='DarkGreen',linestyle='-')
plt.plot(x,boundary_3(x,mu_c0,mu_c1, sigma_c0, sigma_c1,P_w0,P_w1),c='DarkGreen',linestyle='-.')
plt.scatter(data_tr_c0['xs'],data_tr_c0['ys'],c='DarkBlue')# plot data
plt.scatter(data_tr_c1['xs'],data_tr_c1['ys'],c='DarkRed')
plt.xlabel('x')
plt.ylabel('y')
plt.xlim(-1.5,1.5)
plt.ylim(-1,1.5)
c_1_leg= mlines.Line2D([], [], color='DarkBlue', marker='o',markersize=5)
c_2_leg= mlines.Line2D([], [], color='DarkRed', marker='o',markersize=5)
b_1_leg= mlines.Line2D([], [], color='DarkGreen', linestyle='--',markersize=5)
b_2_leg= mlines.Line2D([], [], color='DarkGreen',linestyle='-',markersize=5)
b_3_leg= mlines.Line2D([], [], color='DarkGreen',linestyle='-.',markersize=5)
plt.legend((c_1_leg,c_2_leg,b_1_leg,b_2_leg,b_3_leg),('class 0', 'class 1','case 1','case 2','case 3'))
plt.title('synth.te data with decision rules')
plt.show()
```

B Derivation

B.1 Case 1 boundary



$$g_1(\vec{c}) = g_0(\vec{c})$$

$$\frac{\vec{\mu}_1^T}{\sigma^2}\vec{c} - \frac{\vec{\mu}_1^T\vec{\mu}_1}{2\sigma^2} + \ln p(\omega_1) = \frac{\vec{\mu}_0^T}{\sigma^2}\vec{c} - \frac{\vec{\mu}_0^T\vec{\mu}_0}{2\sigma^2} + \ln p(\omega_0)$$

$$\frac{1}{\sigma^2}\left\{\left[\vec{\mu}_1^T - \mu_0^T\right]\vec{c} - \frac{1}{2}\left(\vec{\mu}_1^T\vec{\mu}_1 - \vec{\mu}_0^T\vec{\mu}_0\right)\right\} = \ln\left[\frac{P(\omega_0)}{P(\omega_1)}\right].$$

$$\left(\mu_{x_1}-\mu_{x_0}, \mu_{y_1}-\mu_{y_0}\right)\begin{pmatrix}\vec{x}\\ \vec{y}\end{pmatrix} = \underbrace{\sigma^2\ln\left(\frac{P(\omega_0)}{P(\omega_1)}\right) + \frac{1}{2}\left(\vec{\mu}_1^T\vec{\mu}_1 - \vec{\mu}_0^T\vec{\mu}_0\right)}_{k_1}$$

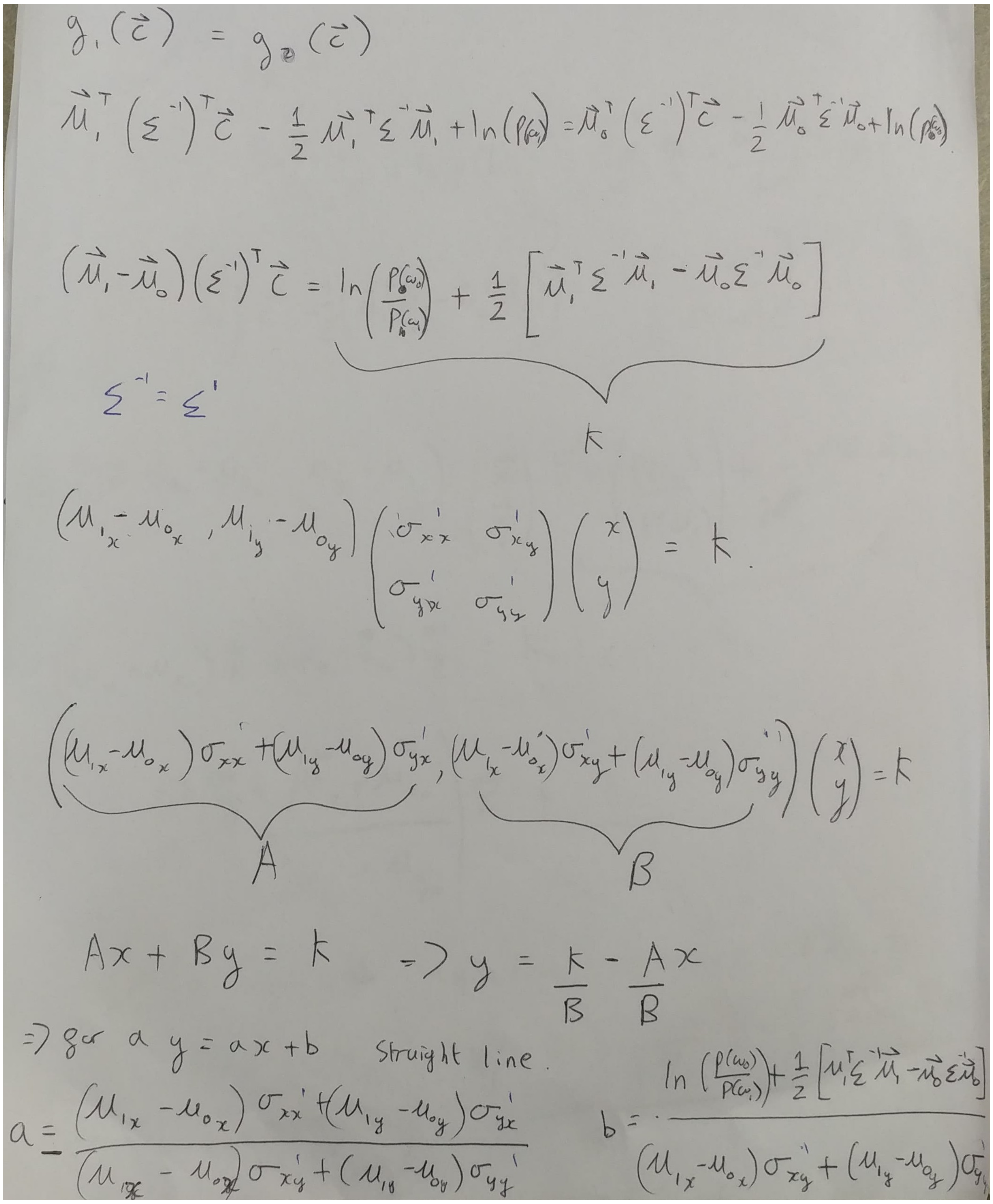
$$\left(\mu_{x_1}-\mu_{x_0}\right)\vec{x} + \left(\mu_{y_1}-\mu_{y_0}\right)\vec{y} = k_1.$$

$$\Rightarrow \vec{y} = -\left(\frac{\mu_{x_1}-\mu_{x_0}}{\mu_{y_1}-\mu_{y_0}}\right)\vec{x} + \frac{k_1}{\mu_{y_1}-\mu_{y_0}}$$

$\Rightarrow$ for a line with equation $y = ax + b$

$$a = -\left(\frac{\mu_{x_1}-\mu_{x_0}}{\mu_{y_1}-\mu_{y_0}}\right), \quad b = \frac{\sigma^2\ln\left(\frac{P(\omega_0)}{P(\omega_1)}\right) + \frac{1}{2}\left(\vec{\mu}_1^T\vec{\mu}_1 - \vec{\mu}_0^T\mu_0\right)}{\mu_{y_1}-\mu_{y_0}}$$

B.2 Case 2 boundary



$$g_1(\vec{c}) = g_0(\vec{c})$$

$$\vec{\mu}_1^T (\Sigma^{-1})^T \vec{c} - \frac{1}{2}\vec{\mu}_1^T \Sigma^{-1} \vec{\mu}_1 + \ln(P(\omega_1)) = \vec{\mu}_0^T (\Sigma^{-1})^T \vec{c} - \frac{1}{2}\vec{\mu}_0^T \Sigma^{-1} \vec{\mu}_0 + \ln(P(\omega_0))$$

$$(\vec{\mu}_1 - \vec{\mu}_0)(\Sigma^{-1})^T \vec{c} = \underbrace{\ln\left(\frac{P(\omega_0)}{P(\omega_1)}\right) + \frac{1}{2}\left[\vec{\mu}_1^T \Sigma^{-1} \vec{\mu}_1 - \vec{\mu}_0 \Sigma^{-1} \vec{\mu}_0\right]}_{k.}$$

$$\Sigma^{-1} = \Sigma'$$

$$(\mu_{1_x} - \mu_{0_x}, \mu_{1_y} - \mu_{0_y}) \begin{pmatrix} \sigma'_{xx} & \sigma'_{xy} \\ \sigma'_{yx} & \sigma'_{yy} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = k.$$

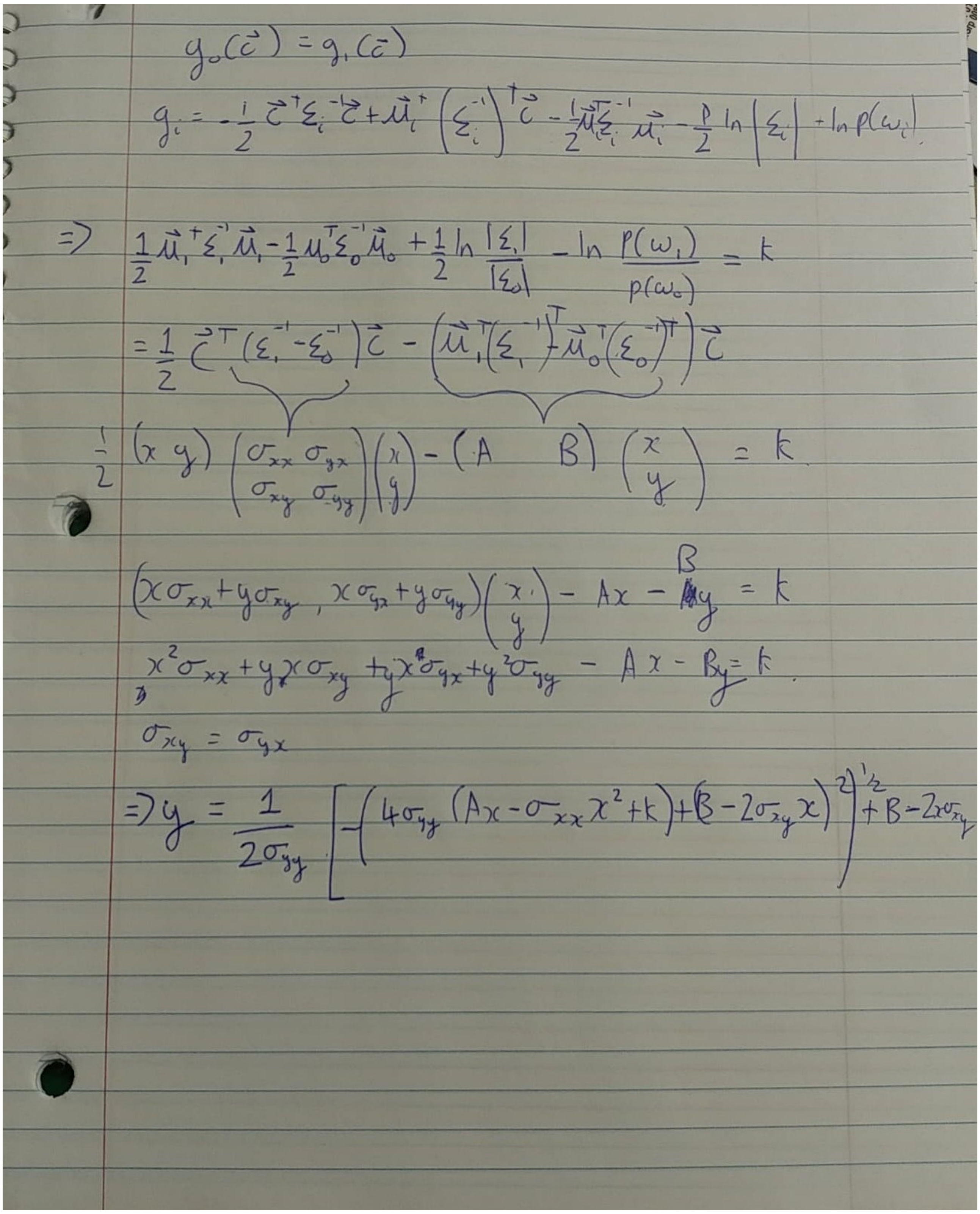
$$\Big(\underbrace{(\mu_{1_x} - \mu_{0_x})\sigma'_{xx} + (\mu_{1_y} - \mu_{0_y})\sigma'_{yx}}_{A}, \underbrace{(\mu_{1_x} - \mu'_{0_x})\sigma'_{xy} + (\mu_{1_y} - \mu_{0_y})\sigma'_{yy}}_{B}\Big)\begin{pmatrix} x \\ y \end{pmatrix} = k$$

$$Ax + By = k \Rightarrow y = \frac{k}{B} - \frac{A}{B}x$$

$\Rightarrow$ for a $y = ax + b$ straight line.

$$a = \frac{(\mu_{1_x} - \mu_{0_x})\sigma'_{xx} + (\mu_{1_y} - \mu_{0_y})\sigma'_{yx}}{(\mu_{1_x} - \mu_{0_x})\sigma'_{xy} + (\mu_{1_y} - \mu_{0_y})\sigma'_{yy}} \qquad b = \frac{\ln\left(\frac{P(\omega_0)}{P(\omega_1)}\right) + \frac{1}{2}\left[\mu_1^T \Sigma^{-1} \vec{\mu}_1 - \vec{\mu}_0 \Sigma^{-1} \vec{\mu}_0\right]}{(\mu_{1_x} - \mu_{0_x})\sigma'_{xy} + (\mu_{1_y} - \mu_{0_y})\sigma'_{yy}}$$

B.3 Case 3 boundary



$$g_0(\vec{c}) = g_1(\vec{c})$$

$$g_i = -\frac{1}{2}\vec{c}^{\,T}\Sigma_i^{-1}\vec{c} + \vec{\mu}_i^{\,T}\left(\Sigma_i^{-1}\right)^{T}\vec{c} - \frac{1}{2}\vec{\mu}_i^{\,T}\Sigma_i^{-1}\vec{\mu}_i - \frac{p}{2}\ln\left|\Sigma_i\right| + \ln p(\omega_i)$$

$$\Rightarrow \frac{1}{2}\vec{\mu}_1^{\,T}\Sigma_1^{-1}\vec{\mu}_1 - \frac{1}{2}\vec{\mu}_0^{\,T}\Sigma_0^{-1}\vec{\mu}_0 + \frac{1}{2}\ln\frac{|\Sigma_1|}{|\Sigma_0|} - \ln\frac{p(\omega_1)}{p(\omega_0)} = k$$

$$= \frac{1}{2}\vec{c}^{\,T}\underbrace{\left(\Sigma_1^{-1} - \Sigma_0^{-1}\right)}\vec{c} - \underbrace{\left(\vec{\mu}_1^{\,T}\left(\Sigma_1^{-1}\right)^{T} - \vec{\mu}_0^{\,T}\left(\Sigma_0^{-1}\right)^{T}\right)}\vec{c}$$

$$\frac{1}{2}\begin{pmatrix} x & y \end{pmatrix}\begin{pmatrix} \sigma_{xx} & \sigma_{yx} \\ \sigma_{xy} & \sigma_{yy} \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} A & B \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = k$$

$$\begin{pmatrix} x\sigma_{xx} + y\sigma_{xy}, & x\sigma_{yx} + y\sigma_{yy} \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} - Ax - \overset{B}{\cancel{A}}y = k$$

$$x^2\sigma_{xx} + yx\sigma_{xy} + yx\sigma_{yx} + y^2\sigma_{yy} - Ax - By = k$$

$$\sigma_{xy} = \sigma_{yx}$$

$$\Rightarrow y = \frac{1}{2\sigma_{yy}}\left[-\left(4\sigma_{yy}\left(Ax - \sigma_{xx}x^2 + k\right) + \left(B - 2\sigma_{xy}x\right)^2\right)^{1/2} + B - 2x\sigma_{xy}\right.$$

References

Brian D Ripley and NL Hjort. *Datasets used in Pattern recognition and neural networks*. Cambridge university press, 1996. Online data sets found at http://www.stats.ox.ac.uk/pub/PRNN/.

Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.