EECE 5644: Machine Learning Project 2 Report

1. Given the following data:

$$p(\mathbf{x}) = p(\mathbf{x}|L=0)p(L=0) + p(\mathbf{x}|L=1)p(L=1)$$
$$p(L=0) = 0.65$$
$$p(L=1) = 0.35$$

$$p(\mathbf{x}|L=0) = w_1 g(\mathbf{x}|m_{01}, C_{01}) + w_2 g(\mathbf{x}|m_{02}, C_{02})$$
$$p(\mathbf{x}|L=1) = g(\mathbf{x}|m_1, C_1)$$
$$w_1 = w_2 = 0.5$$

class conditional parameters:

$$m_{01} = \begin{bmatrix} 3 \\ 0 \end{bmatrix} C_{01} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} m_{02} = \begin{bmatrix} 0 \\ 3 \end{bmatrix} C_{02} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} m_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix} C_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

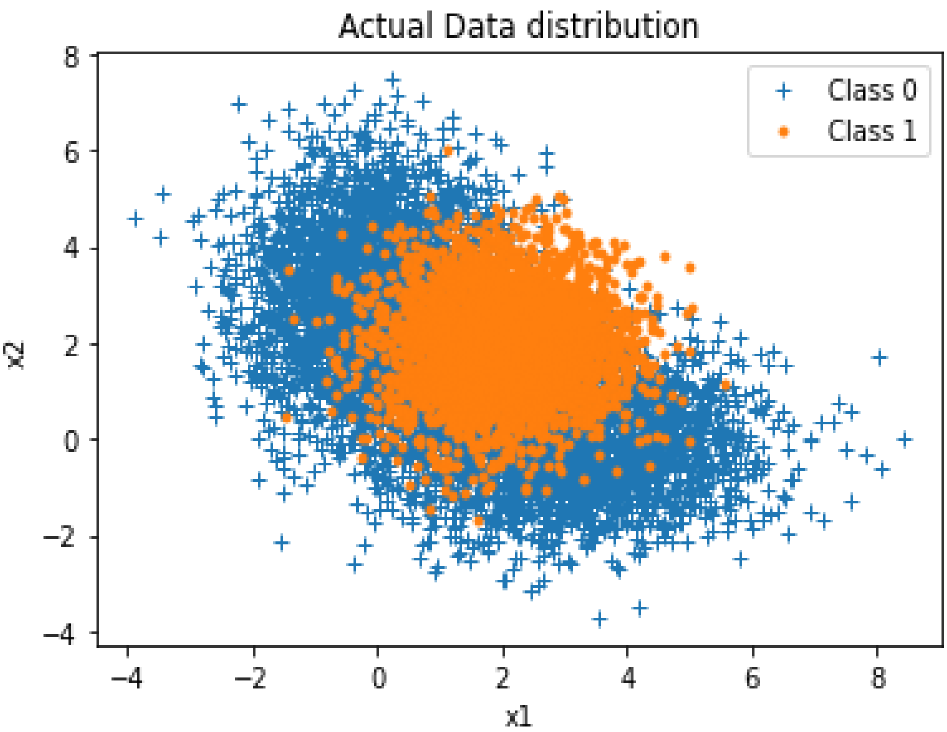


Figure 1: True data distribution for the 10k validation dataset

A ERM classification (Bayes Classifier)

i Minimum expected risk classification rule
Likelihood ratio test:

$$\frac{p(x|L=1)}{p(x|L=0} \underset{L=0}{\overset{L=1}{\gtrless}} \frac{p(L=0)}{p(L=1} * \frac{(\lambda_{10}-\lambda_{00})}{(\lambda_{01}-\lambda_{11})} = \gamma$$

For $0-1$ loss classification, the above reduces to:

$$\frac{p(x|L=1)}{p(x|L=0} \underset{L=0}{\overset{L=1}{\gtrless}} = \frac{0.7}{0.3} * \frac{(1-0)}{(1-0)} = \frac{0.65}{0.35}$$

In the above formula, $\gamma = 2.33$ is the theoretical threshold. Classify $L = 1$ if

$$\frac{p(x|L=1)}{p(x|L=0} > 1.86$$

and classify $L = 0$ otherwise.

ii ROC curve
The minimum Probability error classifier is implemented on the 10K validation set samples. The discriminant scores were compared to the threshold values (which calculated as the mid-point value of the discriminant scores), and the ROC curve plotted based on this threshold value. From the figure, the black dot indicates the point probability of error is minimum. The true positive and false positive rates at this minimum probability error is 0.55 and 0.15 respectively.

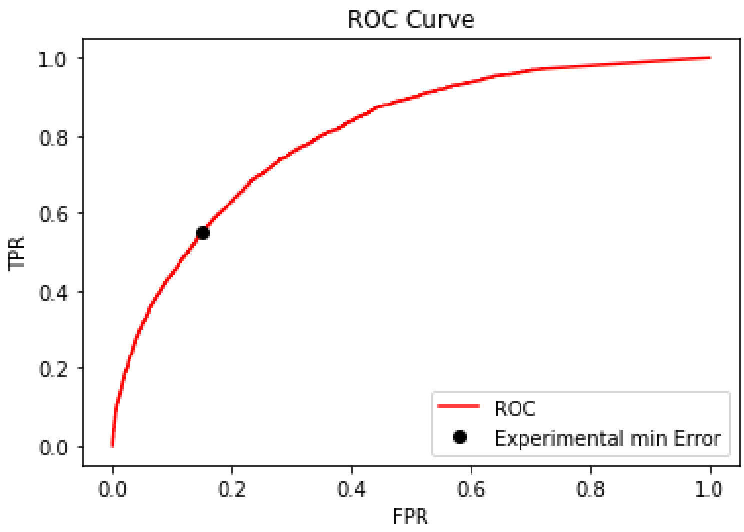


Figure 2: ROC curve with minimum probability error

types	γ	minimum probability error
Theoretical	1.86	0.41
Experimental	2.341	0.25

The following plot shows the distribution after classification:

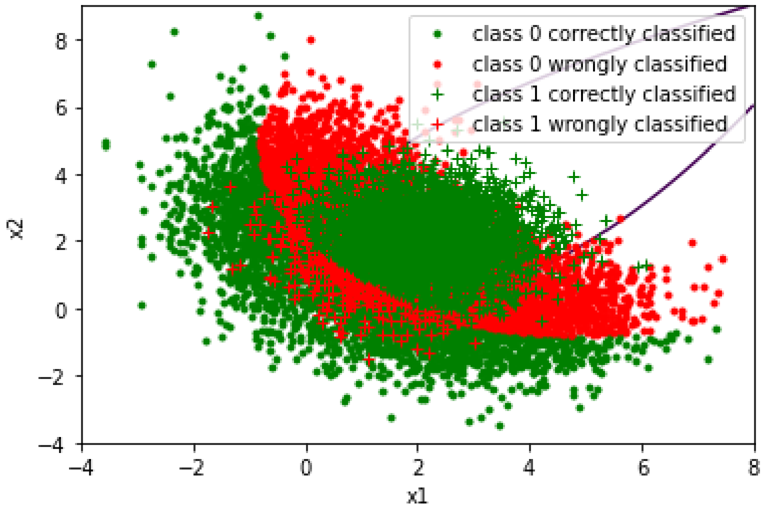


Figure 3: Data distributions after classification

B MLE for logistic-linear-function-based approximation: This approximator is based on 3 separate training datasets and later implemented on 10K test datasets. The theta parameters were trained by minimizing the cost function using gradient descent optimization technique. The cost function is the negative-log-likelihood of each training data:

$$cost = -\frac{1}{N}\Sigma_{i=1}^{N}[y_i ln(h(x_i, w)) + (1 - y_i) ln(1 - h(x_i, w))]$$

where, the sigmoid function $h(x_i, w)$ is

$$h(x_i, w) = \frac{1}{1 + e^{-w^T z(x)}}$$

$$z(x) = [1, x^T]^T$$

i Using the 20 training samples:

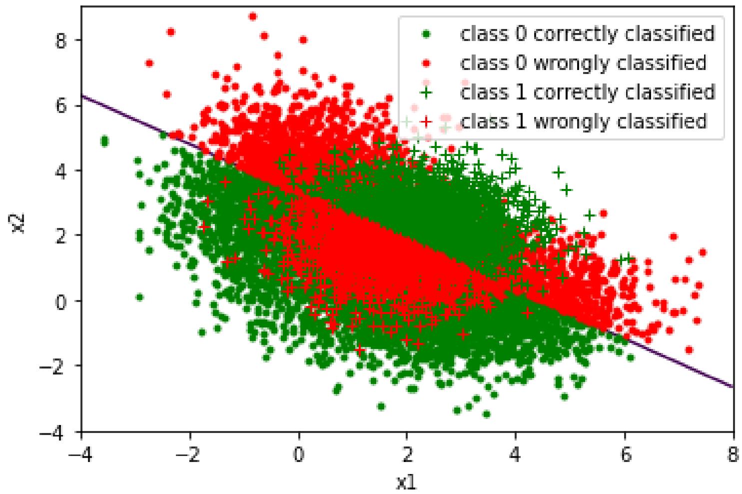


Figure 4: Classification of 10,000 validation samples based on 20 training samples

ii using the 200 training samples

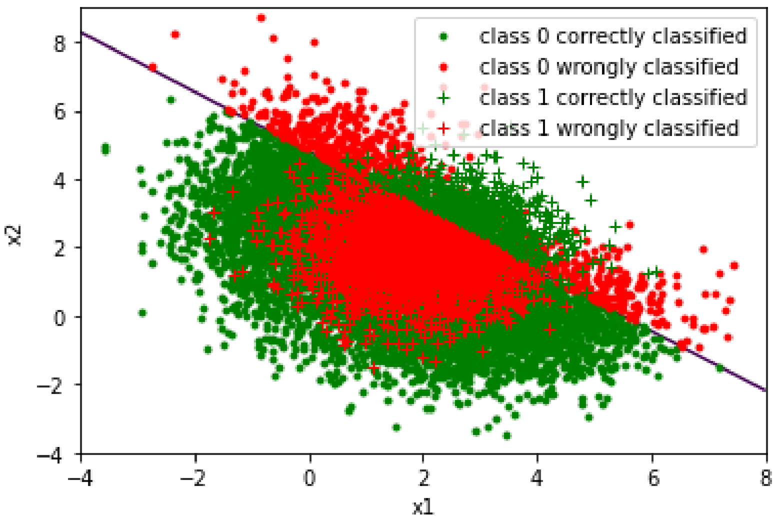


Figure 5: Classification of 10,000 validation samples based on 200 training samples

iii using the 2000 training samples

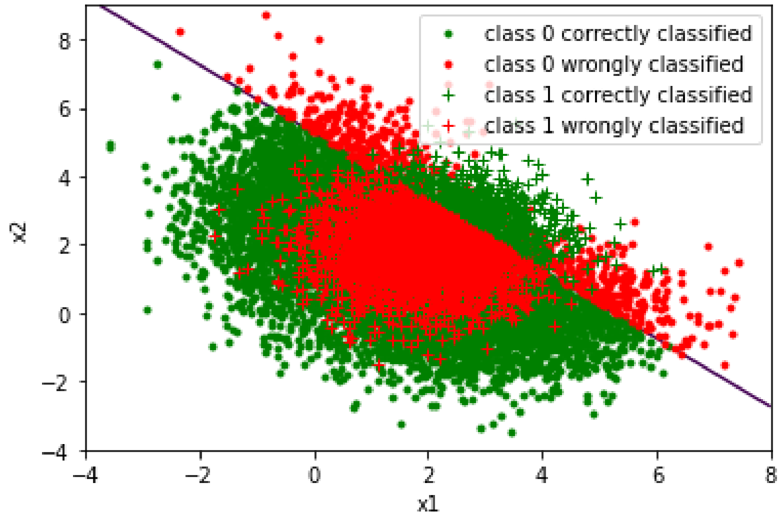


Figure 6: Classification of 10,000 validation samples based on 2000 training samples

Training samples	minimum probability
20	0.367
200	0.346
2000	0.343

Observation:
As the number of the training samples increases, the minimum probability error decreases. Also as the number of training samples increases, we can see from the above figures that, the mis-classification labels decreases (which is shown in red labels).

C MLE for logistic-quadratic-function-based approximation:
The cost and sigmoid function for this approximator is given below:

$$cost = -\frac{1}{N}\Sigma_{i=1}^{N}[y_i ln(h(x_i, w)) + (1 - y_i)ln(1 - h(x_i, w))]$$

where, the sigmoid function $h(x_i, w)$ is

$$h(x_i, w) = \frac{1}{1 + e^{-w^T z(x)}}$$

$$z(x) = [1, x_1, x_2, x_1^2, x_1 x_2, x_2^2]^T$$

i Using the 20 training samples:
After training the 20 samples data, the classifier was applied to the 10k validation datasets.

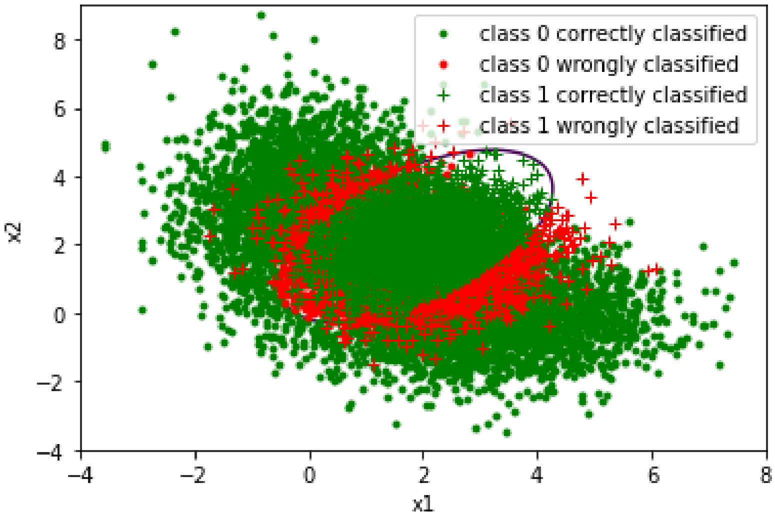


Figure 7: Classification of 10,000 validation samples based on 20 training samples

ii using the 200 training samples

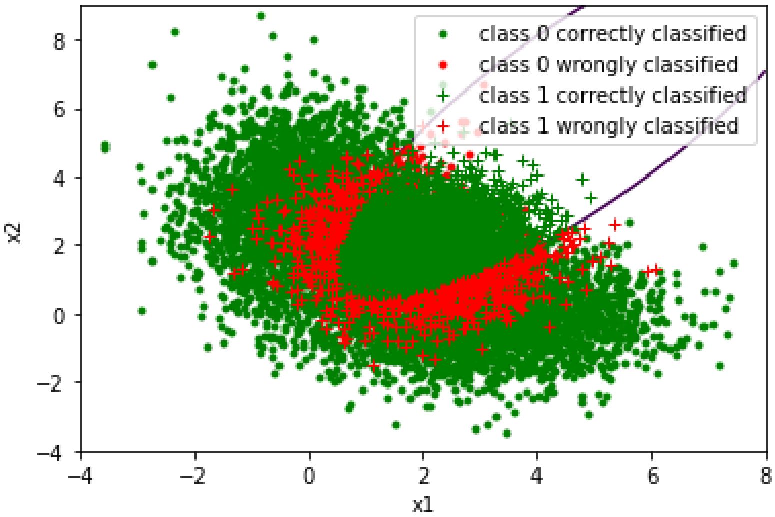


Figure 8: Classification of 10,000 validation samples based on 200 training samples

iii using the 2000 training samples

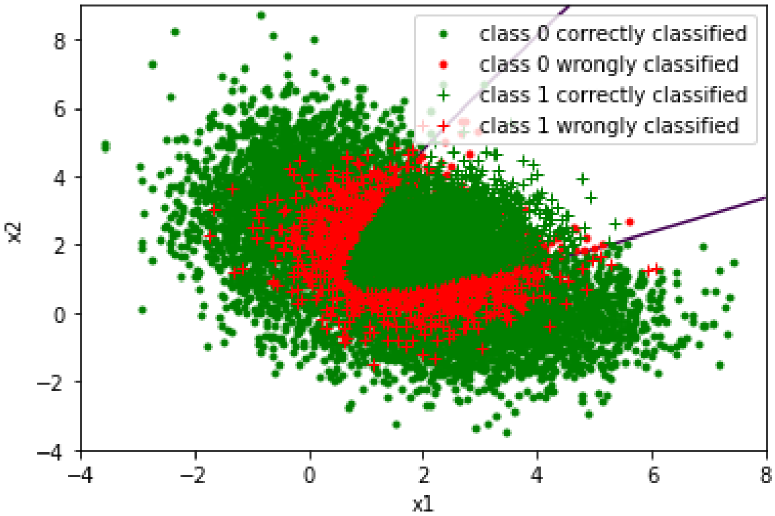


Figure 9: Classification of 10,000 validation samples based on 2000 training samples

Training samples	minimum probability
20	0.223
200	0.171
2000	0.168

The following table shows the minimum probability between the logistic linear and quadratic classifiers.

Training samples	logistic-linear	logistic-quadratic
20	0.367	0.223
200	0.346	0.171
2000	0.343	0.168

Observation:
Similarly, in the quadratic approximation too, as the number of training samples increase the minimum probability error decreases. Furthermore, from the last table shown above, we can see that the quadratic approximation is better performing than the linear approximation, this is due to the overlapping Gaussian data distributions.

2. Given

$$y = c(x, w) + v;\ v \sim N(0, \sigma_{noise});\ \gamma = [10^{-4} : 10^{4}];\ z = [x^3, x^2, x, 1]^T$$

A **ML estimate:** The ML estimate is derived as follows and later implemented to estimate the coefficients of the cubic polynomial using 100 samples of data provided.

ML Estimator

$$w_{ML} = \underset{w}{\arg\max}\ P(y|w,x)$$

$$-\log P(y|w,X) = -\log \prod_{n=1}^{N} P(y_n|w_n,x_n)$$

$$= -\sum_{n=1}^{N} \log P(y|w,x)$$

* In linear regression, the likelihood is Gaussian due to the Gaussian noise term $\varepsilon \sim N(0, \sigma_\varepsilon^2)$.

$$\Rightarrow \log P(y|w,X,\sigma^2) = \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_n - X_n^T w)^2}{2\sigma^2}\right)\right)$$

$$= -\frac{1}{2\sigma^2}(y_n - X_n^T w)^2 + k$$

$\Rightarrow$ the loss function becomes;

$$L(w) = \frac{1}{2\sigma^2} \sum_{n=1}^{N} (y_n - Xw)^2$$

$$= \frac{1}{2\sigma^2} (y - Xw)^T (y - Xw)$$

$$= \frac{1}{2\sigma^2} \|y - Xw\|^2$$

$$\frac{dL}{dw} = 0$$

$$\Rightarrow \frac{d}{dw}\left[\frac{1}{2\sigma^2}(y - Xw)^T(y - Xw)\right] = 0$$

$$\Rightarrow \frac{1}{2\sigma^2}\left(-2y^T X + 2w^T X^T X\right) = 0$$

$$\Rightarrow w^T X^T X = y^T X$$

$$\Rightarrow w^T = y^T X (X^T X)^{-1}$$

$$\Rightarrow \underline{\underline{w_{ML} = (X^T X)^{-1} X^T y}}$$

<u>Estimating Noise Variance</u>

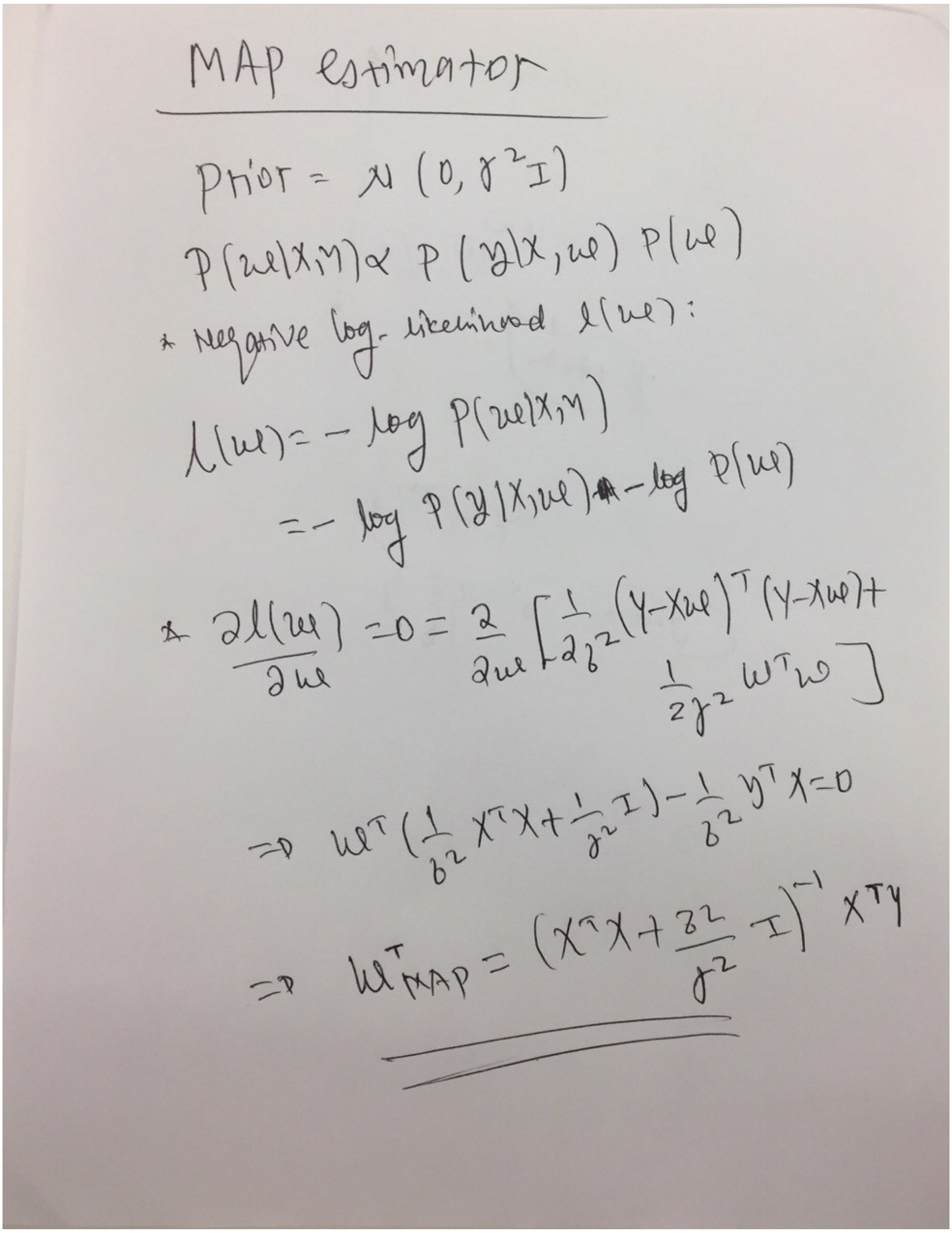
$$\log P(y|X, w, \sigma^2) = \sum_{n=1}^{N} \log \mathcal{N}(y_n | X_n w, \sigma^2)$$

$$= \sum_{n=1}^{N} \left(-\frac{1}{2}\log(2\pi) - \frac{1}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}(y_n - X_n w)^2 \right)$$

$$\Rightarrow \frac{\partial \log P(y|X, w, \sigma^2)}{\partial \sigma^2} = -\frac{N}{2\sigma^2} + \frac{1}{2\sigma^4}\sum_{n=1}^{N}(y_n - X_n w)^2 = 0$$

$$\Rightarrow \underline{\underline{\sigma^2_{ML} = \frac{1}{N}\sum_{n=1}^{N}(y_n - X_n w)^2}}$$

B **MAP estimate:** Similarly, the MAP estimate was derived for estimating the polynomial coefficients and later implemented using 100 training samples.



MAP estimator

Prior $= \mathcal{N}(0, \gamma^2 I)$

$P(w|X,y) \propto P(y|X,w)\, P(w)$

* Negative log-likelihood $l(w)$:

$$l(w) = -\log P(w|X,y)$$

$$= -\log P(y|X,w) - \log P(w)$$

* $$\frac{\partial l(w)}{\partial w} = 0 = \frac{\partial}{\partial w}\left[\frac{1}{2\sigma^2}(y - Xw)^T(y - Xw) + \frac{1}{2\gamma^2} w^T w\right]$$

$$\Rightarrow w^T\left(\frac{1}{\sigma^2} X^T X + \frac{1}{\gamma^2} I\right) - \frac{1}{\sigma^2} y^T X = 0$$

$$\Rightarrow w^T_{MAP} = \left(X^T X + \frac{\sigma^2}{\gamma^2} I\right)^{-1} X^T y$$

After estimating the parameters /coefficients of the cubic polynomial, the model was implemented on 1000 validation samples and different metrics were used to test the performance of both estimators.

After estimating theta parameters for the approximator, I calculated the mean square error for both estimates.

The below graph shows the log-scale plot of mean square error of the validation dataset for ML and MAP estimate. In this case, the noise parameter v has a variance of 3.11 after training it separately, which has a closed form solution of

$$\frac{1}{N}\Sigma_{i=1}^{N}(y_i - xw)^2$$

.

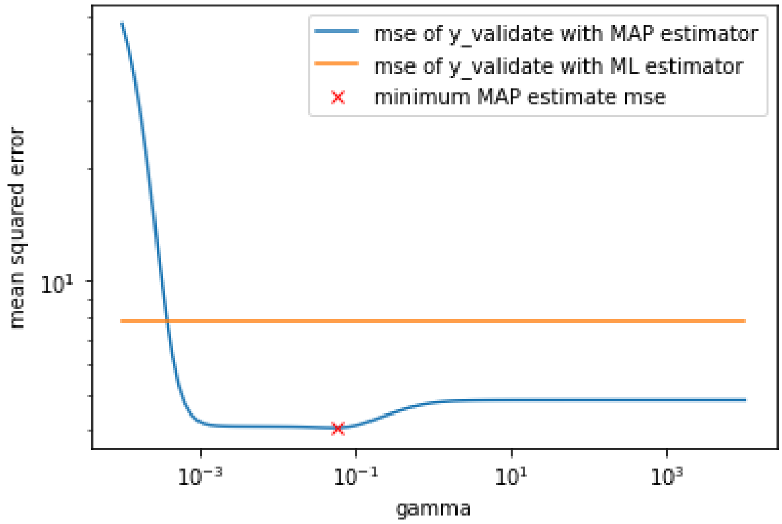


Figure 10: MSE of ML and MAP parameter estimations

The average MSE for using MAP estimator for different gamma values is 6.28, while MSE for ML was found to be 7.83. The minimum MSE for MAP estimator was 4.06 for $\gamma = 0.06$. From this, we can conclude that the optimal gamma value for the best fit is $\gamma = 0.06$. The MSE of MAP estimate after this optimal gamma value will be close to the ML estimate.

The l2 norm between the two θ parameter was also plotted:

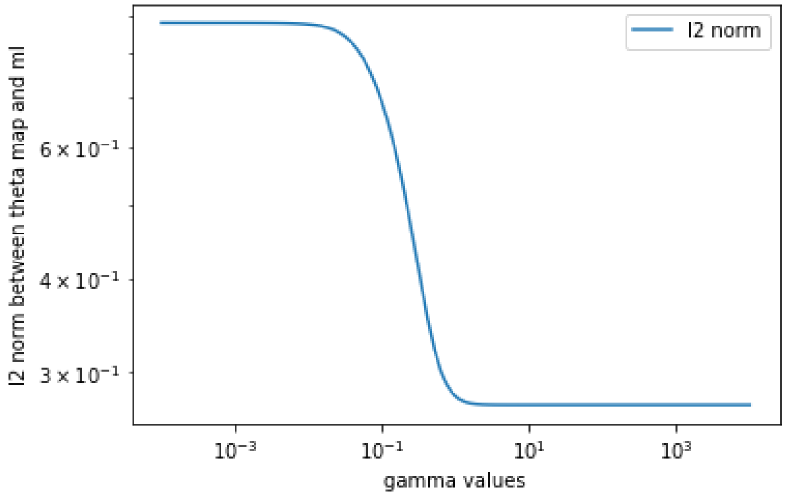


Figure 11: l2 norm of $\theta_{ml} - \theta_{map}$

From the above figure, as the value of gamma increases, the l2 norm of $\theta_{ml} - \theta_{map}$ comes to a constant value (0.271), which implies for large value of gamma the MAP estimate becomes the same as the ML estimate. It can also be seen from the formula that, as the value of gamma increases the MAP estimate becomes ML estimate.

To see the effect of γ values on the training parameters (θ), I plotted the components of θ with γ values. For low values of γ, the θ components has same values, but as the value of γ increases, the parameter estimates starts to diverge.

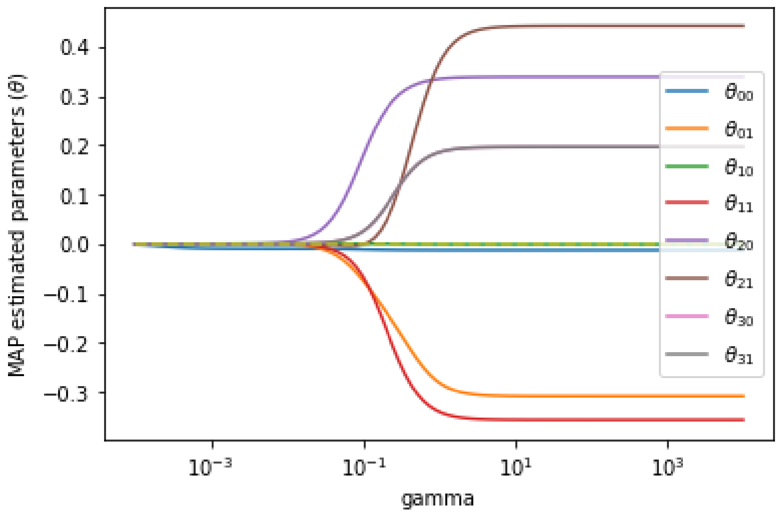


Figure 12: θ_{map} components

3. $Z = [z_1, z_2, \dots, z_K]^T$

$\Theta = [\theta_1, \dots, \theta_K]^T$

$P(z_k = 1) = \theta_k$

$D\{z_1, \dots, z_N\} \Rightarrow$ iid samples

a) ML estimator

$$\hat{\Theta}_{ML} = \arg\max_{\theta} l(\theta)$$

where
$$\begin{aligned} l(\theta) &= \ln P(Z_1 = z_1, \dots, Z_K = z_K; \theta_1, \dots, \theta_K) \\ &\overset{iid}{=} \ln \prod_{i=1}^{N} P(Z_i = z_i; \theta_1, \dots, \theta_K) \\ &= \sum_{i=1}^{N} \ln P(Z_i = z_i; \theta_1, \dots, \theta_K) \end{aligned}$$

$$\Rightarrow \hat{\Theta}_{ML} = \arg\max_{\theta} \sum_{i=1}^{N} \ln P(Z_i = z_i | \theta) \quad \text{s.t.}$$

$$\text{constraints} \begin{cases} \theta \succeq 0 \\ \theta^T \mathbf{1} = 1 \end{cases}$$

$$= \arg\min_{\theta} -\frac{1}{N} \sum_{i=1}^{N} \ln \theta_{z_i} \quad \text{s.t.:} \quad \theta \succeq 0, \; \theta^T \mathbf{1} = 1$$

where $\mathbf{1} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$

assume $\theta \succeq 0$ will be inactive $\Rightarrow$

$$\theta_* \succ 0$$

$$\Rightarrow \theta_* = \arg\min_{\theta} -\frac{1}{N} \sum_{i=1}^{M} \ln \theta_{z_i} \quad \text{s.t. } \theta^T \mathbf{1} - 1 = 0$$

lagrangian of the above optimization becomes

$$\mathcal{L}(\theta, \lambda) = -\frac{1}{N} \sum_{i=1}^{N} \ln \theta_{z_i} + \lambda \left(\theta^T \mathbf{1} - 1\right)$$

KKT conditions:

$$\nabla_{\theta_l} \mathcal{L}(\theta, \lambda) = 0$$

$$\Rightarrow -\frac{1}{N} \sum_{i=1}^{N} \frac{\partial}{\partial \theta_l} \ln \theta_{z_i} + \lambda \frac{\partial}{\partial \theta_l} \left(\theta^T \mathbf{1}\right) = 0$$

$$\Rightarrow \lambda - \frac{1}{N} \sum_{i=1}^{M} \frac{1}{\theta_l} \delta_{z_i l} = 0,$$

$$\text{where } \delta_{z l} = \begin{cases} 0 & \text{if } z \neq l \\ 1 & \text{if } z = l \end{cases}$$

$$\Rightarrow -\frac{1}{N\theta_l}\sum_{i=1}^{N}\delta_{z_i l} + \lambda = \frac{N_l}{N\theta_l} - \lambda = 0$$

$$\Rightarrow \quad \lambda = \frac{N_l}{N\theta_l}$$

, $N_l \rightarrow$ number of samples with $z_i = l$

$$N_1 + N_2 + \cdots + N_K = N$$

$$\lambda\theta_l = \frac{N_l}{N}$$

$$\Rightarrow \lambda\theta_1 = \frac{N_1}{N}$$

$$\lambda\theta_2 = \frac{N_2}{N}$$

$$\vdots$$

$$\lambda\theta_K = \frac{N_K}{N}$$

Summing up the above:

$$\lambda(\theta_1 + \theta_2 + \cdots + \theta_K) = \frac{N_1 + N_2 + \cdots + N_K}{N} = 1$$

$$\lambda = 1$$

$$\Rightarrow \underline{\underline{\theta_l = \frac{N_l}{N}}}$$

b) MAP estimator

Prior: $P(\theta|\alpha) = \frac{1}{B(\alpha)} \prod_{k=1}^{K} \theta_k^{\alpha_k - 1}$

$$B(\alpha) = \frac{\prod_{k=1}^{K} \Gamma(\alpha_k)}{\Gamma\left(\sum_{k=1}^{K} \alpha_k\right)}$$

$$\hat{\Theta}_{MAP} = \underset{\theta}{\operatorname{argmax}} \ln P(\theta|D)$$

$$= \underset{\theta}{\operatorname{argmax}} \ln \frac{P(D|\theta)\,P(\theta)}{P(D)}$$

$$= \underset{\theta}{\operatorname{argmax}} \frac{1}{N} \sum_{i=1}^{N} \ln \theta_{z_i} + \ln\left(\frac{\prod_{k=1}^{K} \theta_k^{\alpha_k - 1}}{B(\alpha)}\right)$$

$$\text{s.t.}: \ \theta \geq 0$$

$$\theta^T 1 - 1 = 0$$

$$* \ \mathcal{L}(\theta, \lambda) = \frac{1}{N} \sum_{i=1}^{N} \ln \theta_{z_i} + \sum_{k=1}^{K} (\alpha_k - 1) \ln \theta_k - \ln B(\alpha) - \lambda(\theta^T 1 - 1)$$

$$\frac{\partial \mathcal{L}}{\partial \theta_k} = 0 = \frac{N_k}{N\theta_k} + \sum_{k=1}^{K} \frac{(\alpha_k - 1)}{\theta_k} - \lambda,$$

$$\Rightarrow \lambda \theta_k = \frac{N_k}{N} + \sum_k \alpha_k - K$$

$$= \frac{N_k}{N} + (\alpha^T \mathbb{1} - K)$$

$\Rightarrow$ adding up the above expression for each k becomes:

$$\lambda(\theta_1 + \dots + \theta_K) = 1 + \alpha^T \mathbb{1} - K$$

$$\Rightarrow \lambda = (\alpha^T \mathbb{1} + 1 - K)$$

$$\Rightarrow \quad \theta_k = \frac{(N_k/N) + (\alpha^T \mathbb{1} - K)}{(\alpha^T \mathbb{1} - K + 1)}$$

A Appendix

```python
#Question 1
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import multivariate_normal as mvn
from collections import Iterable


features = 2
samples = [20,200,2000,10000]
mean_01=np.array([3,0])
mean_02=np.array([0,3])
mean_1=np.array([2,2])

cov_01=np.array([[2,0],[0,1]])
cov_02=np.array([[1,0],[0,2]])
cov_1=np.array([[1,0],[0,1]])

prior = [0.65, 0.35]
fpr = []
tpr = []

P_error = []
gamma_list = []

#part one

def data(n):
  label = np.zeros ((3, n ))
  label [0 , :] = (np.random.uniform (0,1,n)>= prior[0]).astype (int)
  for i in range(n):
     p=np.random.uniform(0,1)
     if p>=prior[0]:
       label[0,i]=1
     else:
       label[0,i]=0
  x = np.zeros ((features , n))
  for index in range (n) :
    if label [0,index] == 0:
      w=np.random.uniform (0,1)
      if w>=0.5:
        x [:,index] = mvn(mean_01.reshape(2,),cov_01).rvs(1)
      else:
        x [:,index] = mvn(mean_02.reshape(2,),cov_02).rvs(1)
    else:
      x [:,index] = mvn(mean_1.reshape(2,),cov_1).rvs(1)
  return x,label

Data_train_20,label_20=data(samples[0])
Data_train_200,label_200=data(samples[1])
Data_train_2000,label_2000=data(samples[2])
Data_validate_10000,label_10000=data(samples[3])

logValpdf0=np.zeros(samples[3])

for i in range(samples[3]):
```

```
  pdf01= np.log(mvn.pdf(Data_validate_10000[:,i].T,mean = mean_01,cov = cov_01))
  pdf02= np.log(mvn.pdf(Data_validate_10000[:,i].T,mean = mean_02,cov = cov_02))
  logValpdf0[i] = 0.5*pdf01+0.5*pdf02

logValpdf1= np.log(mvn.pdf(Data_validate_10000.T,mean=mean_1, cov = cov_1))
discriminant_score = logValpdf1 - logValpdf0
class0_count = float(list(label_10000[0,:]).count(0)) # number of samples for class 0
class1_count = float(list(label_10000[0,:]).count(1)) # number of samples for class 1

tau=(sorted(discriminant_score[np.array(discriminant_score[:].astype(float) >=0)]))
mid_tau=np.array([tau[0]-100,(tau[0:(len(tau)-1)]+(np.diff(tau))/2).tolist(),tau[len(tau)-1]+100])

def flatten(lis):
     for item in lis:
         if isinstance(item, Iterable) and not isinstance(item, str):
             for x in flatten(item):
                 yield x
         else:
             yield item
mid_tau=list(flatten(mid_tau.tolist()))


for gamma in mid_tau:#gamma_list
 label_10000[1,:] = (discriminant_score>=gamma)#.astype(int)#np.log(gamma)

 x01 = [i for i in range(label_10000.shape[1]) if (label_10000[1,i] == 1 and label_10000[0,i] == 0)]
 x11 = [i for i in range(label_10000.shape[1]) if (label_10000[1,i] == 1 and label_10000[0,i] == 1)]
 tpr.append(len(x11)/class1_count)
 fpr.append(len(x01)/class0_count)
 P_error.append(  (len(x01)/class0_count)*prior[0] + (1-len(x11)/class1_count)*prior[1])


minimum_Perror=min(P_error)
min_idx=np.argmin(P_error)
fpr_theory=[]
tpr_theory=[]
p_thry=[]

label_10000[2,:] = (discriminant_score>= np.log(prior[0]/prior[1])).astype(int)
x00t = [i for i in range(label_10000.shape[1]) if (label_10000[0,i] == 0 and label_10000[2,i] == 0)]
x01t = [i for i in range(label_10000.shape[1]) if (label_10000[0,i] == 0 and label_10000[2,i] == 1)]#fp
x10t = [i for i in range(label_10000.shape[1]) if (label_10000[0,i] == 1 and label_10000[2,i] == 0)]#fN
x11t = [i for i in range(label_10000.shape[1]) if (label_10000[0,i] == 1
```

```
    and label_10000[2,i] == 1)]
fpr_theory . append (len( x01t ) / class0_count )
tpr_theory . append (len( x11t ) / class1_count )

p_thry.append(  (len(x01t)/class0_count)*prior[0] + (1-len(x11t)/class1_count)*prior[1])

print('Optimal threshold {}'.format(mid_tau[min_idx]))#change here
print('TPR at minimum probability :{}'.format ( tpr [ min_idx ]))
print('FPR at minimum probability :{}'.format ( fpr [ min_idx ]))
print('Minimum Probability Error:{}'.format(minimum_Perror))
print('Minimum Theoretical Probability Error:{}'.format(min(p_thry)))

#plot ROC

plt.plot(fpr,tpr,color = 'red' )
plt.plot(fpr[min_idx],tpr[min_idx],'o',color='k')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC Curve')
plt.legend(['ROC','Experimental min Error'])
plt.show()



#plot the predicted labels
plt.plot(Data_validate_10000[0,x00t],Data_validate_10000[1,x00t],'.',
    color ='g', markersize = 6)
plt.plot(Data_validate_10000[0,x01t],Data_validate_10000[1,x01t],'.',
    color = 'r', markersize = 6)
plt.plot(Data_validate_10000[0,x11t],Data_validate_10000[1,x11t],'+',
    color ='g', markersize = 6)
plt.plot(Data_validate_10000[0,x10t],Data_validate_10000[1,x10t],'+',
    color = 'r', markersize = 6)
plt.legend(['class 0 correctly classified','class 0 wrongly classified',
    'class 1 correctly classified','class 1 wrongly classified'])
plt.xlabel("x1")
plt.ylabel("x2")

horizontalGrid = np.linspace(np.floor(min(Data_validate_10000[0,:])),np.
    ceil(max(Data_validate_10000[0,:])),100)
verticalGrid = np.linspace(np.floor(min(Data_validate_10000[1,:])),np.
    ceil(max(Data_validate_10000[1,:])),100)

dsg = np.zeros((100,100))
a = np.array(np.meshgrid(horizontalGrid,verticalGrid))
for i in range(100):
  for j in range(100):
    ww=np.random.uniform(0,1)
    p = mvn.pdf(np.array([a[0][i][j], a[1][i][j]]),mean=mean_1, cov =
    cov_1)
    q1=mvn.pdf(np.array([a[0][i][j], a[1][i][j]]),mean=mean_01, cov =
    cov_01)
    q2=mvn.pdf(np.array([a[0][i][j], a[1][i][j]]),mean=mean_02, cov =
    cov_02)
    q=0.5*q1+0.5*q2
    dsg[i][j] = np.log(q) - np.log(p) - np.log(prior[0]/prior[1])
plt.contour(a[0],a[1],dsg,levels=[0])
```

```
plt.show()


x0 = [i for i in range(label_10000.shape[1]) if (label_10000[0,i] == 0)]
x1 = [i for i in range(label_10000.shape[1]) if (label_10000[0,i] == 1 )
    ]
plt.plot(Data_validate_10000[0,x0],Data_validate_10000[1,x0],'+')
plt.plot(Data_validate_10000[0,x1],Data_validate_10000[1,x1],'.')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title("Actual Data distribution")
plt.legend(['Class 0','Class 1'])
plt.show()


#part 2
def linear_reg(xdata,alpha,max_iteration,ydata,xtest):
  z=np.vstack((np.ones(xdata.shape[1]),xdata))
  w=np.zeros((3,1))

  for i in range(max_iteration):
    h=1/(1+np.exp(-(np.dot(w.T,z))))
    grad=(1/float(z.shape[1])) * np.dot(z,(h-ydata[0]).T)
    w=w-alpha*grad
  z=np.vstack((np.ones(xtest.shape[1]),xtest))
  decisions=np.zeros((1,xtest.shape[1]))
  h=1/(1+np.exp(-(np.dot(w.T,z))))
  decisions[0,:]=(h[0,:]>=0.5).astype(int)
  return w,decisions
def quadratic_reg(xdata,alpha,max_iteration,ydata,xtest):

  z = np.vstack((np.ones(xdata.shape[1]),xdata[0],xdata[1],xdata[0]*
   xdata[0],xdata[0]*xdata[1], xdata[1]*xdata[1]))
  w = np.zeros((6,1))
  for i in range(max_iteration):
    h=1/(1+np.exp(-(np.dot(w.T,z))))
    grad=(1/float(z.shape[1])) * np.dot(z,(h-ydata[0]).T)
    w=w-alpha*grad
  z = np.vstack((np.ones(xtest.shape[1]),xtest[0],xtest[1],xtest[0]*
   xtest[0],xtest[0]*xtest[1], xtest[1]*xtest[1]))
  decisions = np.zeros((1,xtest.shape[1]))
  h=1/(1+np.exp(-(np.dot(w.T,z))))
  decisions[0,:]=(h[0,:]>=0.5).astype(int)
  return w,decisions


alpha=0.05
max_iter=2000

w_20,decisions_20=linear_reg(Data_train_20,alpha,max_iter,label_20,
    Data_validate_10000)
w_200,decisions_200=linear_reg(Data_train_200,alpha,max_iter,label_200,
    Data_validate_10000)
w_2000,decisions_2000=linear_reg(Data_train_2000,alpha,max_iter,
    label_2000,Data_validate_10000)

w_20q,decisions_20q=quadratic_reg(Data_train_20,alpha,max_iter,label_20,
    Data_validate_10000)
```

```
w_200q,decisions_200q=quadratic_reg(Data_train_200,alpha,max_iter,label_200,Data_validate_10000)
w_2000q,decisions_2000q=quadratic_reg(Data_train_2000,alpha,max_iter,label_2000,Data_validate_10000)

def contour_plot(typ,w_par):
  horizontalGrid = np.linspace(np.floor(min(Data_validate_10000[0,:])),np.ceil(max(Data_validate_10000[0,:])),100)
  verticalGrid = np.linspace(np.floor(min(Data_validate_10000[1,:])),np.ceil(max(Data_validate_10000[1,:])),100)
  dsg = np.zeros((100,100))
  a = np.array(np.meshgrid(horizontalGrid,verticalGrid))
  for i in range(100):
    for j in range(100):
      x1 = a[0][i][j]
      x2 = a[1][i][j]
      if typ==0:
        z = np.c_[1,x1,x2].T
      else:
        z = np.c_[1,x1,x2,pow(x1,2),x1*x2,pow(x2,2)].T
      dsg[i][j] = np.sum(np.dot(w_par.T,z))
  plt.contour(a[0],a[1],dsg,levels=[0])
  plt.show()

def plots(Data_train,label_train,decision,w_val,typ):
  P_err=[]

  #actual data plots
  x0 = [i for i in range(label_train.shape[1]) if (label_train[0,i] == 0)]
  x1 = [i for i in range(label_train.shape[1]) if (label_train[0,i] == 1)]

  plt.plot(Data_train[0,x0],Data_train[1,x0],'+')
  plt.plot(Data_train[0,x1],Data_train[1,x1],'.')
  plt.xlabel('x1')
  plt.ylabel('x2')
  plt.title("Actual Data distribution")
  plt.legend(['Class 0','Class 1'])
  if typ==0:#linear
    contour_plot(0,w_val)
  else:
    contour_plot(1,w_val)

  #plot the classified data

  x00 = [i for i in range(label_10000.shape[1]) if (label_10000[0,i] == 0 and decision[0,i] == 0)]
  x01 = [i for i in range(label_10000.shape[1]) if (label_10000[0,i] == 0 and decision[0,i] == 1)]#FP
  x10 = [i for i in range(label_10000.shape[1]) if (label_10000[0,i] == 1 and decision[0,i] == 0)]
  x11 = [i for i in range(label_10000.shape[1]) if (label_10000[0,i] == 1 and decision[0,i] == 1)]#TP
  P_err.append(  (len(x01)/class0_count)*prior[0] + (1-len(x11)/class1_count)*prior[1])
  print(min(P_err))

```

```
  plt.plot(Data_validate_10000[0,x00],Data_validate_10000[1,x00],'.', color ='g', markersize = 6)
  plt.plot(Data_validate_10000[0,x01],Data_validate_10000[1,x01],'.', color = 'r', markersize = 6)
  plt.plot(Data_validate_10000[0,x11],Data_validate_10000[1,x11],'+', color ='g', markersize = 6)
  plt.plot(Data_validate_10000[0,x10],Data_validate_10000[1,x10],'+', color = 'r', markersize = 6)
  plt.legend(['class 0 correctly classified','class 0 wrongly classified','class 1 correctly classified','class 1 wrongly classified'])
  plt.xlabel("x1")
  plt.ylabel("x2")

  if typ==0:#linear
    contour_plot(0,w_val)
  else:
    contour_plot(1,w_val)


plots(Data_train_20,label_20,decisions_20,w_20,0)
plots(Data_train_20,label_20,decisions_20q,w_20q,1)
plots(Data_train_200,label_200,decisions_200,w_200,0)
plots(Data_train_200,label_200,decisions_200q,w_200q,1)
plots(Data_train_2000,label_2000,decisions_2000,w_2000,0)
plots(Data_train_2000,label_2000,decisions_2000q,w_2000q,1)
```

Listing 1: Question 1

```
#Question 2
import numpy as np
import pandas as pd
from numpy.matlib import repmat
import matplotlib.pyplot as plt


def generateData(N):
    gmmParameters = {}
    gmmParameters['priors'] = [.3,.4,.3] # priors should be a row vector
    gmmParameters['meanVectors'] = np.array([[-10, 0, 10], [0, 0, 0], [10, 0, -10]])
    gmmParameters['covMatrices'] = np.zeros((3, 3, 3))
    gmmParameters['covMatrices'][:,:,0] = np.array([[1, 0, -3], [0, 1, 0], [-3, 0, 15]])
    gmmParameters['covMatrices'][:,:,1] = np.array([[8, 0, 0], [0, .5, 0], [0, 0, .5]])
    gmmParameters['covMatrices'][:,:,2] = np.array([[1, 0, -3], [0, 1, 0], [-3, 0, 15]])
    x,labels = generateDataFromGMM(N,gmmParameters)
    return x

def generateDataFromGMM(N,gmmParameters):
#     Generates N vector samples from the specified mixture of Gaussians
#     Returns samples and their component labels
#     Data dimensionality is determined by the size of mu/Sigma parameters
    priors = gmmParameters['priors'] # priors should be a row vector
    meanVectors = gmmParameters['meanVectors']
    covMatrices = gmmParameters['covMatrices']
    n = meanVectors.shape[0] # Data dimensionality
```

```
    C = len(priors) # Number of components
    x = np.zeros((n,N))
    labels = np.zeros((1,N))
    # Decide randomly which samples will come from each component
    u = np.random.random((1,N))
    thresholds = np.zeros((1,C+1))
    thresholds[:,0:C] = np.cumsum(priors)
    thresholds[:,C] = 1
    for l in range(C):
        indl = np.where(u <= float(thresholds[:,l]))
        Nl = len(indl[1])
        labels[indl] = (l+1)*1
        u[indl] = 1.1
        x[:,indl[1]] = np.transpose(np.random.multivariate_normal(
   meanVectors[:,l], covMatrices[:,:,l], Nl))

    return x,labels

def plot3(a,b,c,mark="o",col="b"):
  from matplotlib import pyplot
  import pylab
  from mpl_toolkits.mplot3d import Axes3D
  pylab.ion()
  fig = pylab.figure()
  ax = Axes3D(fig)
  ax.scatter(a, b, c,marker=mark,color=col)
  ax.set_xlabel("x1")
  ax.set_ylabel("x2")
  ax.set_zlabel("y")
  ax.set_title('Training Dataset')

def mse(x,y):
  error=0
  for i in range(x.shape[0]):
    error=error+pow(x[i]-y[i],2)

  return (1/x.shape[0])*error

Ntrain = 100
data1 = generateData(Ntrain)
#plot3(data1[0,:],data1[1,:],data1[2,:])#plot the training data
xTrain= data1[0:2,:]
yTrain = data1[2,:]

Ntrain = 1000
data2 = generateData(Ntrain)
#plot3(data2[0,:],data2[1,:],data2[2,:])#plot the validation data
xValidate = data2[0:2,:]
yValidate = data2[2,:]
sigm=0.1

z=np.vstack((pow(xTrain,3),pow(xTrain,2),pow(xTrain,1),pow(xTrain,0)))
   #design matrix

gamma_list=[]

for i in np.logspace(-4,4,100):
  gamma_list.append(i)
```

```python

zzt=np.zeros((8,8))
b=np.zeros(8).reshape(8,1)
for j in range(z.shape[1]):
  temp=z[:,j].reshape(8,1)
  zzt=zzt+np.matmul(temp,temp.T)
  b=b+temp*(yTrain[j])
theta_map_list=[]

theta_ml=np.matmul(np.linalg.inv(zzt+0.0001*np.random.normal(0,sigm
    ,(8,8))),b)# I added the small error to make the matrix non-singular

sigmaa=0
for i in range(xTrain.shape[1]):
  temp=np.matmul(theta_ml[0:2,0].reshape(1,2), pow(xTrain[:,i],3)) + np.
    matmul(theta_ml[2:4,0].reshape(1,2), pow(xTrain[:,i],2))+np.matmul(
    theta_ml[4:6,0].reshape(1,2), pow(xTrain[:,i],1))\
  +np.matmul(theta_ml[6:8,0].reshape(1,2), pow(xTrain[:,i],0))
  sigmaa=sigmaa+pow((yTrain[i]-temp),2)

sigmaa=(1/xTrain.shape[1])*sigmaa


for gamma in gamma_list:
  A=zzt+(sigmaa/pow(gamma,2))*np.eye(8)
  theta_map=np.matmul(np.linalg.inv(A),b)

  theta_map_list.append(theta_map)

v = np.random.normal(0,np.sqrt(sigmaa),1000)
y_validate_ml=np.matmul(theta_ml[0:2,0].reshape(1,2),pow(xValidate,3)) +
     np.matmul(theta_ml[2:4,0].reshape(1,2),pow(xValidate,2))\
  +np.matmul(theta_ml[4:6,0].reshape(1,2),pow(xValidate,1))+np.matmul(
    theta_ml[6:8,0].reshape(1,2),pow(xValidate,0))+v


y_validate_map=[]
l2_norm=[]
l2a,l2b,l2c,l2d=[],[],[],[]
y_train_map=[]

for i in theta_map_list:
  y_val=np.matmul(i[0:2,0].reshape(1,2),pow(xValidate,3)) + np.matmul(i
    [2:4,0].reshape(1,2),pow(xValidate,2))\
  +np.matmul(i[4:6,0].reshape(1,2),pow(xValidate,1))+np.matmul(i[6:8,0].
    reshape(1,2),pow(xValidate,0))

  y_validate_map.append(y_val)
  l2_norm.append(pow(np.linalg.norm(theta_ml-i),2))
  y_train_map.append(np.matmul(i[0:2,0].reshape(1,2),pow(xTrain,3)) + np
    .matmul(i[2:4,0].reshape(1,2),pow(xTrain,2))\
  +np.matmul(i[4:6,0].reshape(1,2),pow(xTrain,1))+np.matmul(i[6:8,0].
    reshape(1,2),pow(xTrain,0)))


mse_list_map=[]
mse_list_map_train=[]
for y in y_validate_map:
```

```
    rr=mse(y.T,yValidate.T)
    mse_list_map.append(rr)

a=[]
b=[]
c=[]
d=[]
for i in theta_map_list:
    a.append(i[0:2,0])
    b.append(i[2:4,0])
    c.append(i[4:6,0])
    d.append(i[6:8,0])

for xx in y_train_map:
    mse_list_map_train.append(mse(xx.T,yTrain.T))

mse_list_ml=mse(yValidate.T,y_validate_ml.T)

from numpy.lib.function_base import average
print('mse for using the ml estimator:{}'.format(mse_list_ml))
print('avergae mse for using the map estimator for different gamma
    values:{}'.format(average(mse_list_map)))
print('minimum map mse:{}'.format(min(mse_list_map)))
print(gamma_list[np.argmin(mse_list_map)])

plt.plot(gamma_list,l2_norm,label='l2 norm')
plt.plot(gamma_list,np.zeros(len(gamma_list)),'--',label='reference')
plt.xscale(value='log')
plt.yscale(value='log')
plt.xlabel('gamma values')
plt.ylabel('l2 norm between theta map and ml')
plt.legend()
plt.show()

plt.plot(gamma_list,mse_list_map,label='mse of y_validate with MAP
    estimator')
#plt.plot(gamma_list,mse_list_map_train,label='mse of y_train map with
    gamma values')
plt.plot(gamma_list,[mse_list_ml for i in range(len(gamma_list))],label=
    'mse of y_validate with ML estimator ')
plt.plot(gamma_list[np.argmin(mse_list_map)],min(mse_list_map),'xr',
    label='minimum MAP estimate mse')
plt.xscale(value='log')
plt.yscale(value='log')
plt.xlabel('gamma')
plt.ylabel('mean squared error')
plt.legend()
plt.show()


plt.plot(gamma_list,a)
plt.plot(gamma_list,b)
plt.plot(gamma_list,c)
plt.plot(gamma_list,d)
plt.plot(gamma_list,np.zeros(len(gamma_list)),'--',label='reference')
plt.xscale(value='log')
#plt.yscale(value='log')
plt.xlabel('gamma')
```

```
plt.ylabel(r'MAP estimated parameters ($\theta$)')
plt.legend([r'${\theta}_{00}$',r'${\theta}_{01}$',r'${\theta}_{10}$',r'${\theta}_{11}$',r'${\theta}_{20}$',r'${\theta}_{21}$',r'${\theta}_{30}$',r'${\theta}_{31}$'])
```

Listing 2: Question 2