EECE5644
Assignment 2

By Dhanush Balakrishna

Question 1

Plots for the datasets are shown below:

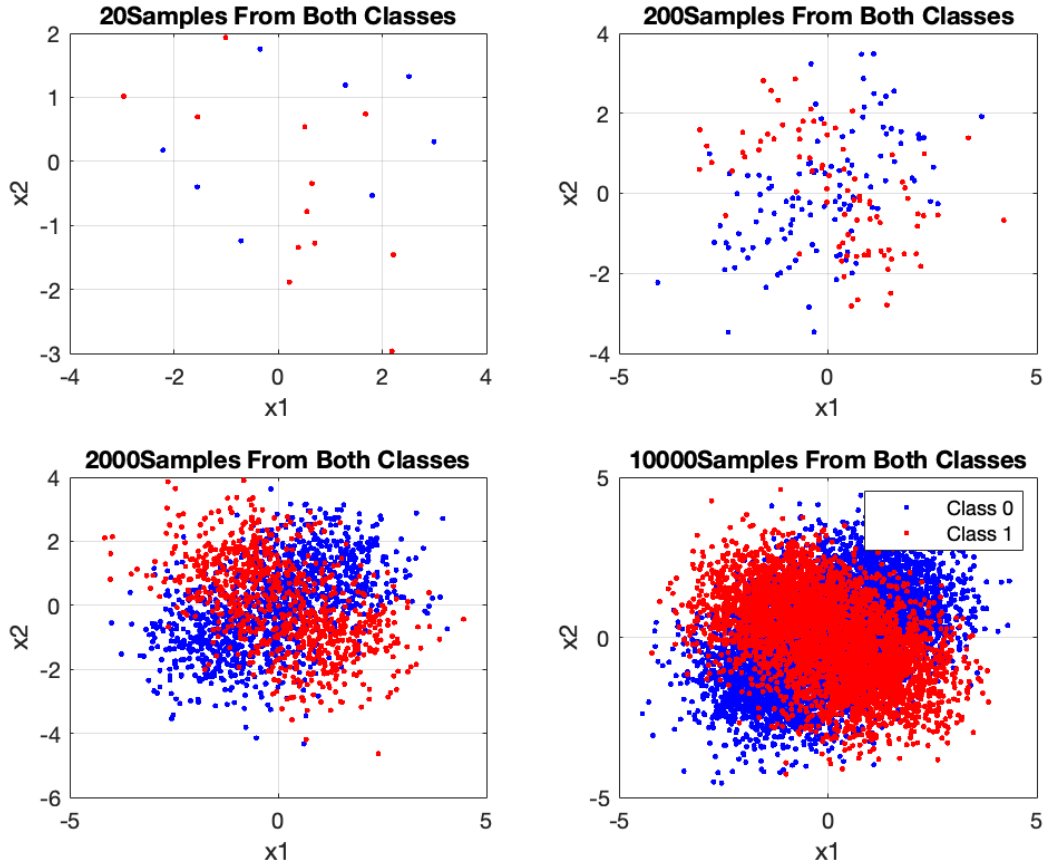


Fig.1.1: Datasets

$$(D = 1) = \frac{P(x|L1)}{P(x|L0)} \geq \frac{\lambda_{10} - \lambda_{00}}{\lambda_{01} - \lambda_{11}} * \frac{P(L0)}{P(L1)} = \gamma \ \ (D = 0)$$

To minimize probability of misclassifications the cost for incorrect classification should be 1 and the cost for correct classifications should be 0 which results in the gamma shown below.

$$(D = 1) = \frac{P(x|L1)}{P(x|L0)} \geq \frac{1 - 0}{1 - 0} * \frac{0.6}{0.4} = 1.86 = \gamma \ \ (D = 0)$$

Plots of the ROC curve with the calculated ideal minimum error point as well as the minimum error point estimated from the generated validation data is shown in Figure 1.2. The probability of error versus Gamma with the calculated and estimated minimum error points marked are shown in Figure 1.3.

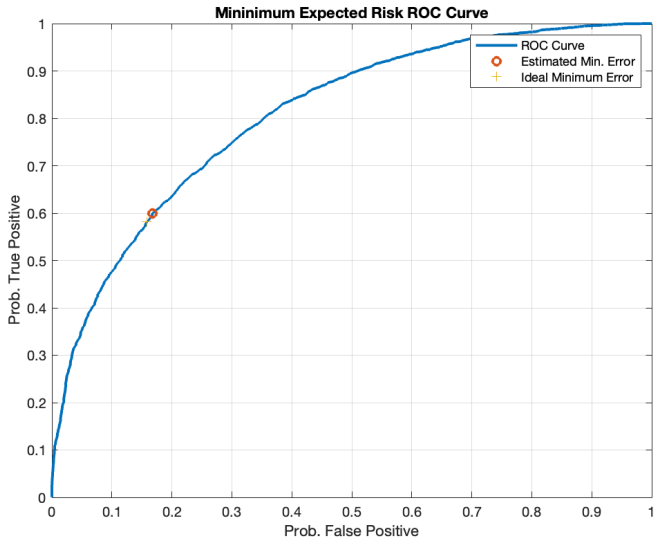


Fig 1.2: ROC Curve for Known Ideal Classification Case

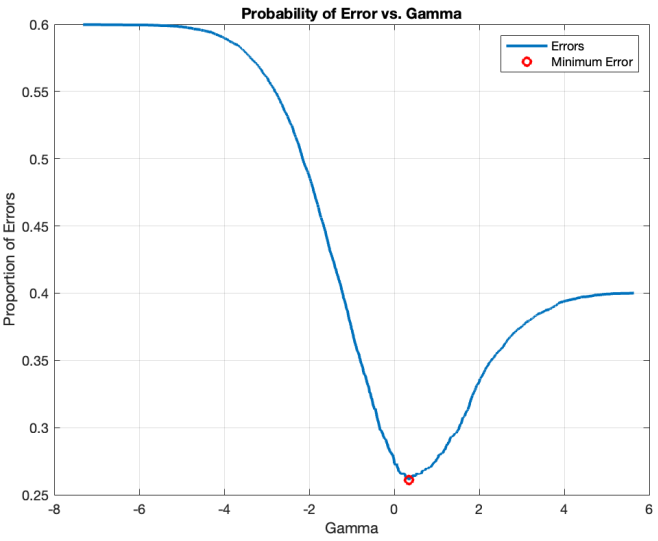


Fig 1.3: Probability of Error Curve for Ideal Classification

Figure 1.4 shows the decision space for each distribution along with equipluve contours of the discriminant function.

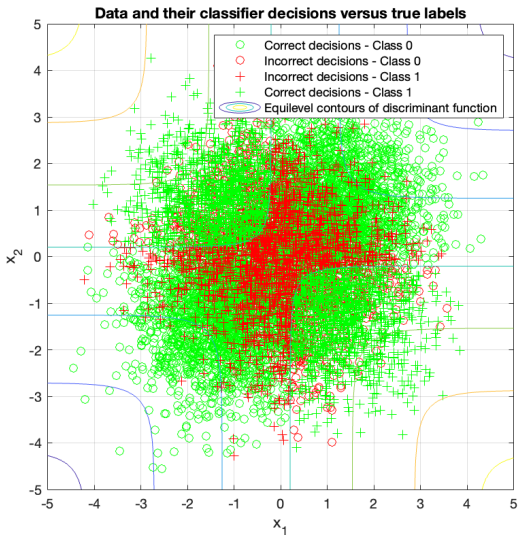


Fig 1.4: Decision Boundary of Ideal Classifier

For maximum likelihood parameter estimation techniques were used to train logistic linear and logistic quadratic based approximation of class label posterior functions given as a sample. This training was performed on each of the three separate training datasets consisting of 20, 200 and 2000 samples respectively and was then used to classify samples from the 10000 sample validation data set.

The logistic function is defined as follows:

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

For the linear logistic function $z(x) = [1\ x_1 x_2]^T$
For the quadratic logistic function $z(x) = [1\ x_1 x_2 x_1^2 x_1 * x_2 x_2^2]^T$
The **w** vectors are estimated using numerical optimization techniques with the cost function.
$\widehat{\theta_{ML}} = -\frac{1}{N}\sum_1^N l_n \ln\big(h(x_n, \theta)\big) + (1 - l_n)\ln\,(1 - h(x_n, \theta))$

The minimum expected risk classification criteria are then.
(l_n=1) $\quad \widehat{w}^T z(x) \gtrless 0 \quad$ (l_n=0)

Table 1.1 contains a summary of the resulting probability of errors from classifying the 10000 sample validation data set using each of the 3 training data sets. The data shows that for both the linear and quadratic estimation functions the probabilities of error decrease as the number of points in the training datasets increase. Additionally, the quadratic logistic function significantly outperformed the linear logistic function in all cases.

Training Dataset	Linear	Quadratic
20	0.4920	0.3108
200	0.4348	0.2829
2000	0.4007	0.2741

Table 1.1: Logistic Function Probabilities of Error

Figures 1.5.1, 1.5.2, 1.6.1, 1.6.2, 1.7.1 and 1.7.2 show the data points of X with classifier decisions and true labels marked.

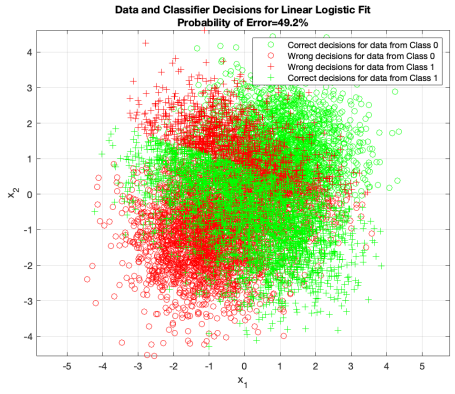


Fig 1.5.1: Classifier for Linear Logistic Fit on D20 training data

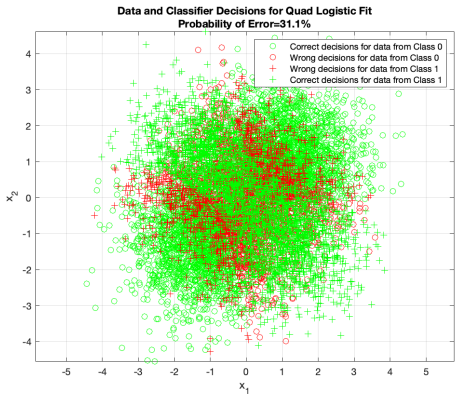


Fig 1.5.1: Classifier for Quadratic Logistic Fit on D20 training data

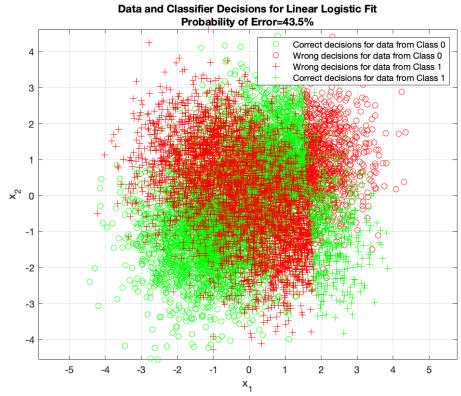


Fig 1.5.1: Classifier for Linear Logistic Fit on D200 training data

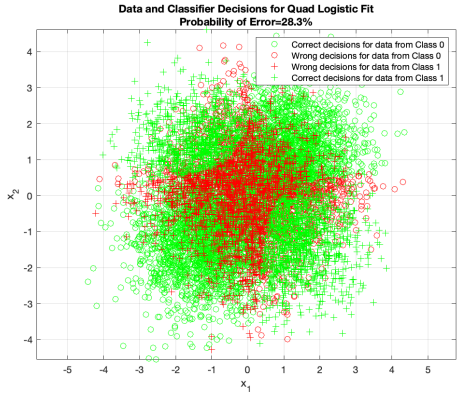


Fig 1.5.1: Classifier for Quadratic Logistic Fit on D200 training data

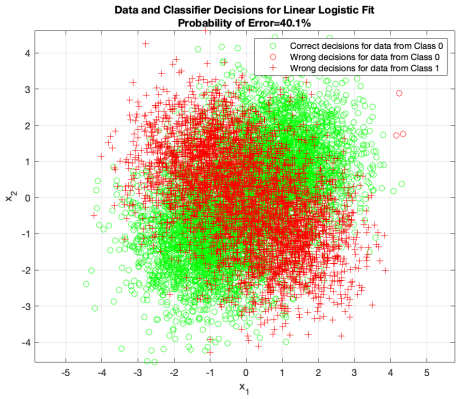


Fig 1.5.1: Classifier for Linear Logistic Fit on D2000 training data

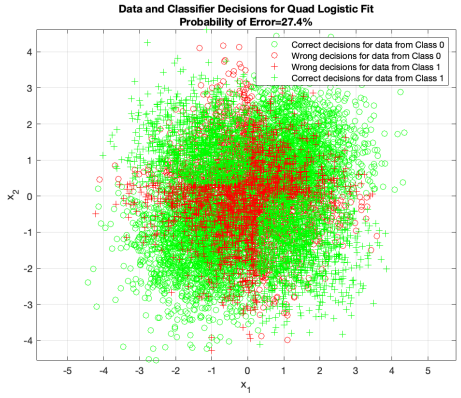


Fig 1.5.1: Classifier for Quadratic Logistic Fit on D2000 training data

Question 2

Assume that scalar-real $y \in \Re$ and two-dimensional real vector $x \in \Re^2$ are related to each other according to y=c (x, w) +∈, where c (., w) is a cubic polynomial in x with coefficients w, and ∈ is a random Gaussian scalar with mean zero and σ^2- variance (∈~N (0, σ^2)).

Given a dataset $D = \{(x^{(1)}, y^{(1)}), \dots, (x^{(N)}, y^{(N)})\}$ with N samples of (x, y) pairs that are independent and identically distributed according to the model, derive two estimators for w using maximum-likelihood (ML) and maximum-a-posteriori(MAP) parameter estimation approaches.

ML Estimation

Let's begin by defining the augmented features $\tilde{x}$ fir a two – dimensional real input vector and the corresponding regression weights w after applying a cubic polynomial c (x, w) transformation:

$$\tilde{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_1 x_2 \\ x_2^2 \\ x_1^3 \\ x_1^2 x_2 \\ x_1 x_2^2 \\ x_2^3 \end{bmatrix} \in \Re^{10}, \; w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \\ w_7 \\ w_8 \\ w_9 \end{bmatrix} \in \Re^{10}$$

Now given our polynomial regression model $y = w^T \tilde{x} + \in$, out first step to performing ML parameter estimation is choosing a suitable parametric model, which we know for linear regression (still "linear" in the parameter space) is the conditional univariate Gaussian $p(y|\tilde{x}; \theta) = N(y|w^T \tilde{x}, \sigma^2)$. Given a dataset D of N iid samples, this can be expressed as:

$$p(D|\theta) = \prod_{i=1}^{N} p\left(y^{(i)} \middle| \tilde{x}^{(i)}; \theta\right) = \prod_{i=1}^{N} N\left(y^{(i)} \middle| w^T \tilde{x}^{(i)}, \sigma^2\right),$$

Where θ = [w, σ^2]^T. Assuming this model for our dataset, the ML estimator for the parameters $\widehat{\theta_{MLE}}$ is obtained by maximizing the log-likelihood of the data, or equivalently, minimizing negative log-likelihood of the data, or equivalently, minimizing negative log-likelihood (NLL):

$$\begin{aligned} \widehat{\theta_{MLE}} &= \underbrace{argmin}_{\theta} \text{NLL}(\theta) = \underbrace{argmin}_{\theta} - \ln p(D|\theta) \\ &= \underbrace{argmin}_{\theta} - \textstyle\sum_{i=1}^{N} \ln p(y^{(i)}|\tilde{x}^{(i)}; \theta) \\ &= \underbrace{argmin}_{\theta} - \textstyle\sum_{i=1}^{N} \ln[\left(\left(\frac{1}{2\pi\sigma^2}\right)^{\frac{1}{2}} \exp\left(-\frac{1}{2\sigma^2}\left(y^{(i)} - w^T \tilde{x}^{(i)}\right)^2\right)\right)] \end{aligned}$$

$$= \underbrace{argmin}_{\theta} \frac{N}{2}\ln(2\pi\sigma^2) + \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y^{(i)} - w^T\tilde{x}^{(i)})^2$$

Assuming fixed variance, e.g., $\sigma^2 = 1$, we will focus on estimating w alone, resulting in an expression equal to the residual sum of squares (RSS) loss. For NLL (θ) where the regression weights **w** are our only parameters, then the derivative is:

$$\nabla NLL_w(\theta) = \sum_{i=1}^{N}(y^{(i)} - w^T\tilde{x}^{(i)})\,\tilde{x}^{(i)} = X^Ty - X^TXw,$$

With $X \in \Re^{N*10}$ as the design matrix containing the transformed input features. Hence the final ML estimator can be solved for by setting this gradient to zero and rearranging (assuming X^TX is invertible)

MAP Estimation

For the MAP estimator, assume that w has a zero-mean Gaussian prior with covariance matrix γI.

We follow a similar routine to ML parameter estimation, except that the MAP parameter estimates $\widehat{\theta_{MAP}}$ adds a regularization term to the log-likelihood term, namely the log-prior:

$$\begin{aligned}\hat{\boldsymbol{\theta}}_{\text{MAP}} &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \ln p(\boldsymbol{\theta} \mid \mathcal{D}) \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \ln p(\mathcal{D} \mid \boldsymbol{\theta}) + \ln p(\boldsymbol{\theta}),\end{aligned}$$

Where prior θ=w, p(θ)=N(w|0,γI). The log of the pdf of this evaluates to:

$$\begin{aligned}\ln p(\boldsymbol{\theta}) &= \ln \mathcal{N}(\mathbf{w} \mid \mathbf{0}, \gamma\mathbf{I}) \\ &= \ln\left[\frac{1}{(2\pi)^{\frac{n}{2}}|\gamma\mathbf{I}|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(\mathbf{w}-\mathbf{0})^{\intercal}(\gamma\mathbf{I})^{-1}(\mathbf{w}-\mathbf{0})\right)\right] \\ &= -\frac{n}{2}\ln(2\pi) - \frac{1}{2}\ln|\gamma\mathbf{I}| - \frac{1}{2}\mathbf{w}^{\intercal}(\gamma\mathbf{I})^{-1}\mathbf{w},\end{aligned}$$

$$\begin{aligned}\hat{\boldsymbol{\theta}}_{\text{MAP}} &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} - [\ln p(\mathcal{D} \mid \boldsymbol{\theta}) + \ln p(\boldsymbol{\theta})] \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{2}\sum_{i=1}^{N}(y^{(i)} - \mathbf{w}^{\intercal}\tilde{\mathbf{x}}^{(i)})^2 + \frac{1}{2\gamma}\mathbf{w}^{\intercal}\mathbf{w}.\end{aligned}$$

$$\begin{aligned}\nabla(-\ln p(\boldsymbol{\theta} \mid \mathcal{D}))_{\mathbf{w}} &= \frac{\delta}{\delta\mathbf{w}^{\intercal}}\left[\frac{1}{2}\sum_{i=1}^{N}(y^{(i)} - \mathbf{w}^{\intercal}\tilde{\mathbf{x}}^{(i)})^2 + \frac{1}{2\gamma}\mathbf{w}^{\intercal}\mathbf{w}\right] \\ &= \sum_{i=1}^{N}(y^{(i)} - \mathbf{w}^{\intercal}\tilde{\mathbf{x}}^{(i)})\tilde{\mathbf{x}}^{(i)} + \frac{1}{\gamma}\mathbf{w} \\ &= \mathbf{X}^{\intercal}\mathbf{X}\mathbf{w} - \mathbf{X}^{\intercal}\mathbf{y} + \frac{1}{\gamma}\mathbf{w},\end{aligned}$$

$$\mathbf{X}^\intercal\mathbf{X}\mathbf{w} - \mathbf{X}^\intercal\mathbf{y} + \frac{1}{\gamma}\mathbf{w} = \mathbf{0},$$

$$(\mathbf{X}^\intercal\mathbf{X} + \frac{1}{\gamma}\mathbf{I})\mathbf{w} = \mathbf{X}^\intercal\mathbf{y},$$

$$\mathbf{w}^* = (\mathbf{X}^\intercal\mathbf{X} + \frac{1}{\gamma}\mathbf{I})^{-1}\mathbf{X}^\intercal\mathbf{y} = \left(\sum_{i=1}^{N}\tilde{\mathbf{x}}^{(i)}\tilde{\mathbf{x}}^{(i)\intercal} + \frac{1}{\gamma}\mathbf{I}\right)^{-1}\left(\sum_{i=1}^{N}\tilde{\mathbf{x}}^{(i)}y^{(i)}\right) = \hat{\boldsymbol{\theta}}_{\text{MAP}}.$$

Using the *training dataset* (Ntrain=100), obtain the ML estimator and the MAP estimator for a variety of γ values ranging from $10^{-4}$to 10^{4} (span a log scale). Evaluate each *trained* model by calculating the mean squared error (MSE) between the y values in the *validation samples* (Nvalid=1000) and model estimates of these using c (., wtrain).

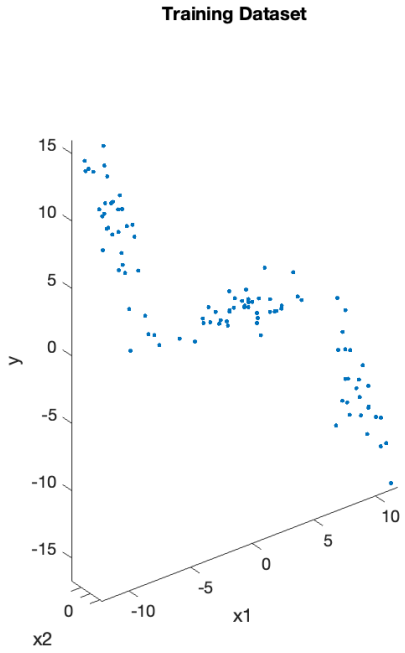
How does your MAP-trained model perform on the validation set as γ is varied? How is the MAP estimate related to the ML estimate? Describe your experiments, visualize, and quantify your analyses with data from these experiments.

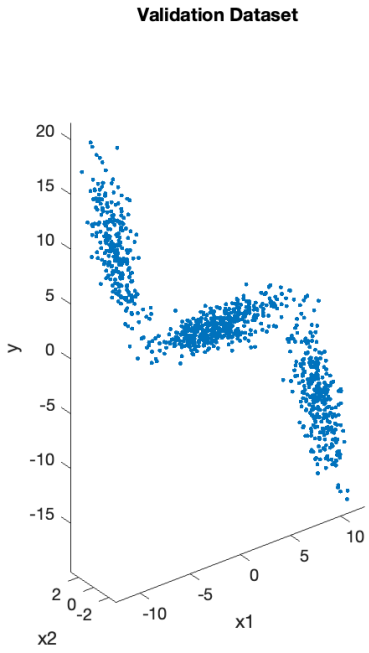
A MAP-trained linear regression model with a zero-mean Gaussian prior on the weights w is also known as ridge regression. The benefit of employing this parameter estimation technique is that it combats the precarious issue of overfitting, which MLE is susceptible to, especially in low N sample size settings like ours (N = 100 for model fitting is not a lot). It achieves this by adding an explicit term to penalize large weight magnitudes, thereby constraining the solution weights vector from taking on arbitrarily complex values. Consider the penalized NLL objective:
Regarding performance of the two estimators, as measured by MSE, we find that the MAP estimator for the polynomial regression model can lead to a much better solution when provided with an appropriate Γ value.

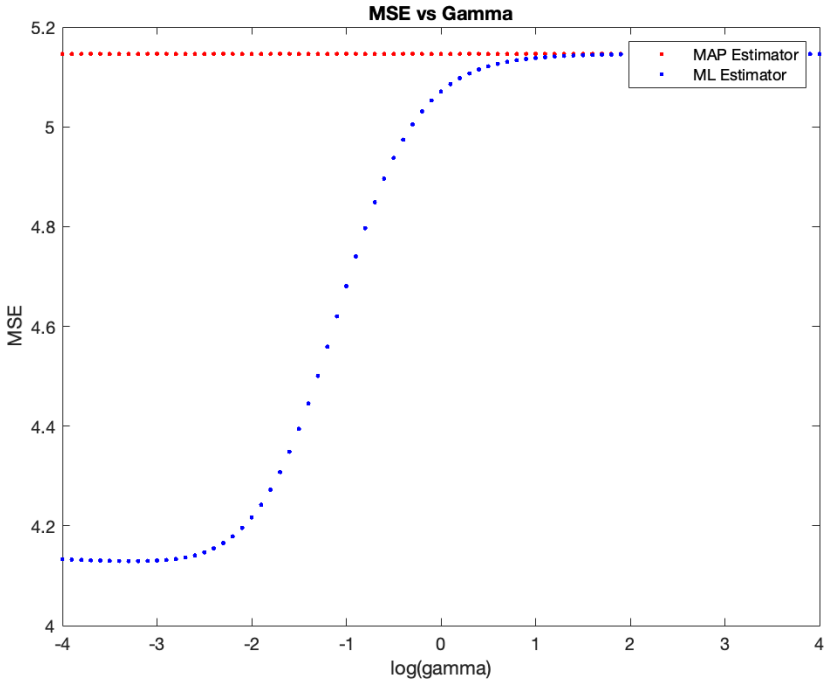
So, what is the relationship between these two estimators and how does the regularization parameter? γ plays a role?
From our plot of MSE vs γ, we observe that at larger values.
γ>10 the MAP estimator converges to a similar result as that of the ML estimator. This is because as γ tends towards infinity, the prior's variance approaches zero and the posterior distribution over the parameters becomes like an infinitely narrow "spike" around the ML point estimate. In other words, the posterior distribution is entirely determined by the likelihood of the data and MAP becomes equivalent to ML. When γ is too large (or λ too small), the optimization objective focuses entirely on maximizing likelihood of the data and may result in overfitting... but what about when γ is too small?
There is a drastic increase in MSE. This is because the posterior distribution learnt now focuses on staying too close to the prior and is not considering any of the available data, causing underfitting.

Although we found a suitable value of
γ through this experiment, we may not always have enough data to create a validation set for hyperparameter selection. Note that if γ was selected based on this validation set, then we could no longer use this same subset of data to evaluate generalization performance, as our model would provide a biased prediction estimate. Instead, one can stick to only using the training set

by employing cross-validation to select this regularization strength parameter. This is exactly the task of one of the questions in the following assignment.






MSE vs Gamma
MAP Estimator
ML Estimator
MSE
log(gamma)
5.2
5
4.8
4.6
4.4
4.2
4
-4
-3
-2
-1
0
1
2
3
4

Question 3

Question 3:

The objective is to find the $[x, y]^T$ coordinate position with the highest probability given the prior distribution as well as the range measurements from each of the K reference coordinates.

$$\begin{bmatrix} x_{MAP} \\ y_{MAP} \end{bmatrix} = \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmax}}\, p\left(\begin{bmatrix} x \\ y \end{bmatrix} \mid \{r_1 \dots r_K\}\right)$$

$$= \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmax}} \left((2\pi\sigma_x\sigma_y)^{-1} e^{-\frac{1}{2}[x\ y]\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix}} \right) + \sum_{i=1}^{K} \ln p\left(\begin{bmatrix} x \\ y \end{bmatrix} \mid r_i\right)$$

$$= \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmax}} -\frac{1}{2}[x\ y]\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix} + \sum_{i=1}^{K} \ln N(r_i \mid 0, \sigma_i^2)$$

$$= \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmax}} -\frac{1}{2}[x\ y]\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix} + \sum_{i=1}^{K} \ln\left((2\pi\sigma_i^2)^{-\frac{1}{2}} e^{-\frac{((r_i - d_i) - 0)^2}{2\sigma_i^2}} \right)$$

$$= \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmax}} -\frac{1}{2}[x\ y]\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix} + \sum_{i=1}^{K} \ln\left((2\pi\sigma_i^2)^{-\frac{1}{2}} \right) + \ln\left(e^{-\frac{(r_i - d_i)^2}{2\sigma_i^2}} \right)$$

$$= \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmax}} -\frac{1}{2}[x\ y]\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix} + \sum_{i=1}^{K} -\frac{(r_i - d_i)^2}{2\sigma_i^2}$$

$$= \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmin}} [x\ y]\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix} + \sum_{i=1}^{K} \frac{(r_i - d_i)^2}{\sigma_i^2}$$

$$\underset{[x\ y]}{\operatorname{argmin}} [x\ y] \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1} \begin{bmatrix} x \\ y \end{bmatrix} + \sum_{i=1}^{k} \frac{(\lambda_i - \| [x\ y] - [x_i\ y_i] \|)^2}{\sigma_i^2} .$$

Now let's look at the contours that we obtained with varying values for K (1,2,3,4):

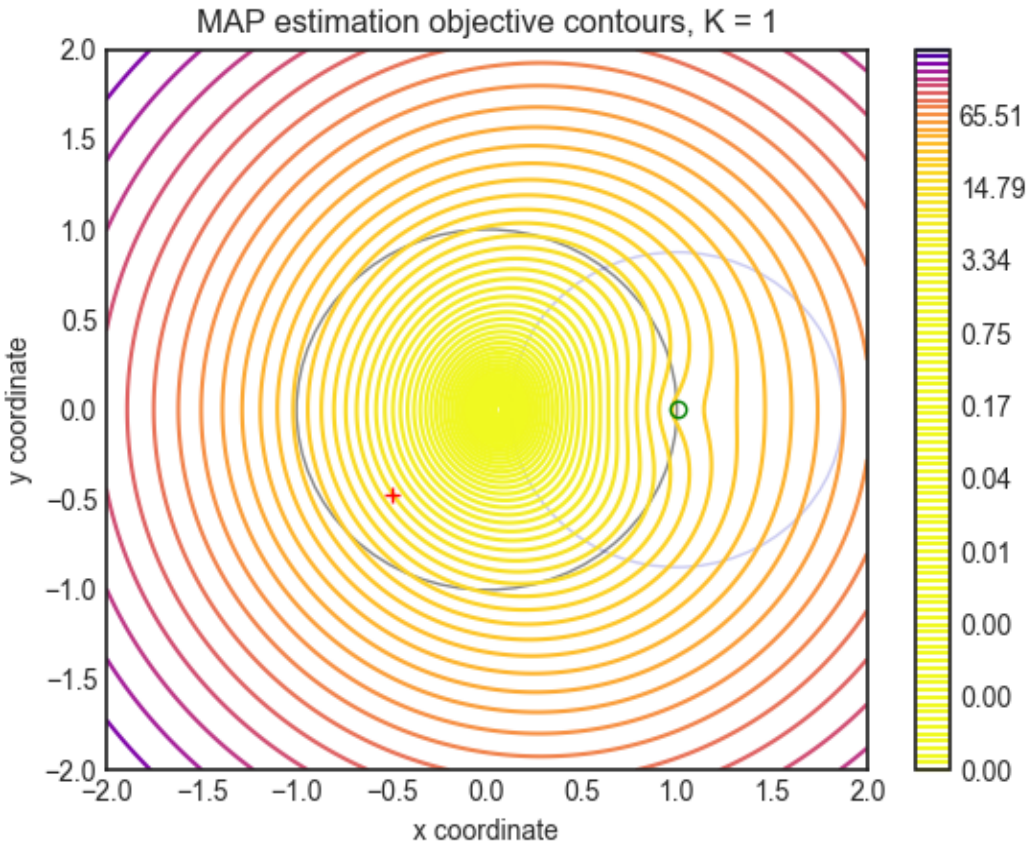


Fig.3.1: MAP Estimation Objective Contours, K=1

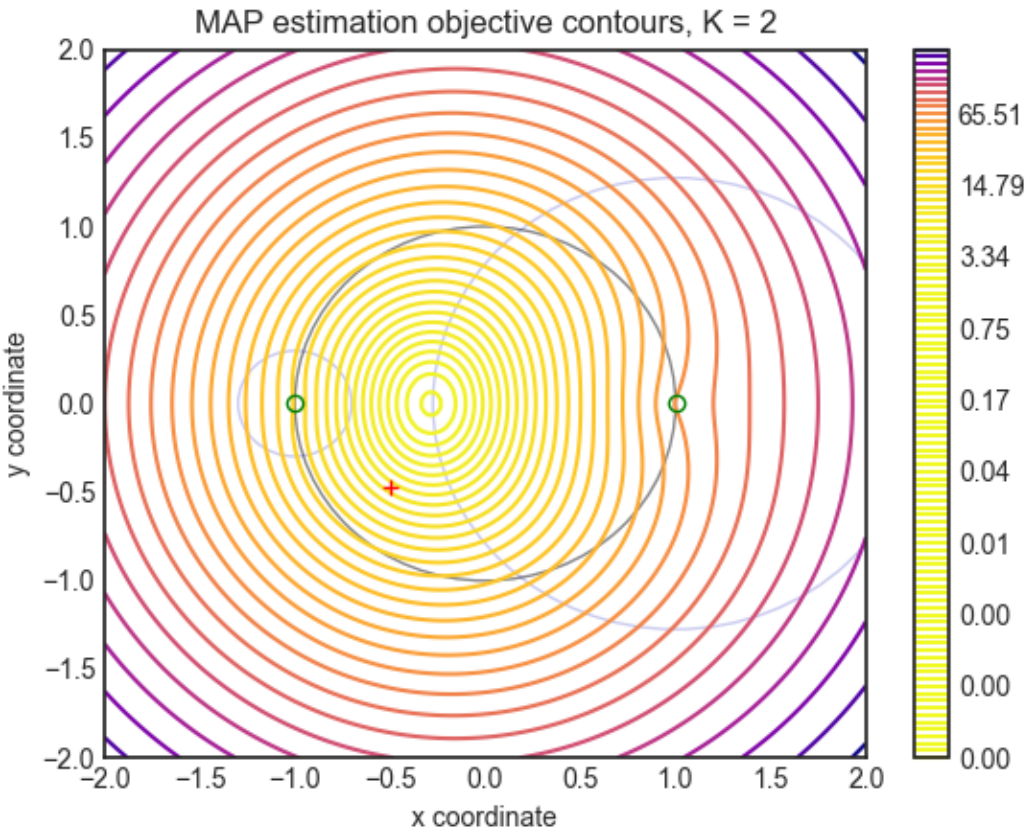


Fig.3.2: MAP Estimation Objective Contours, K=2

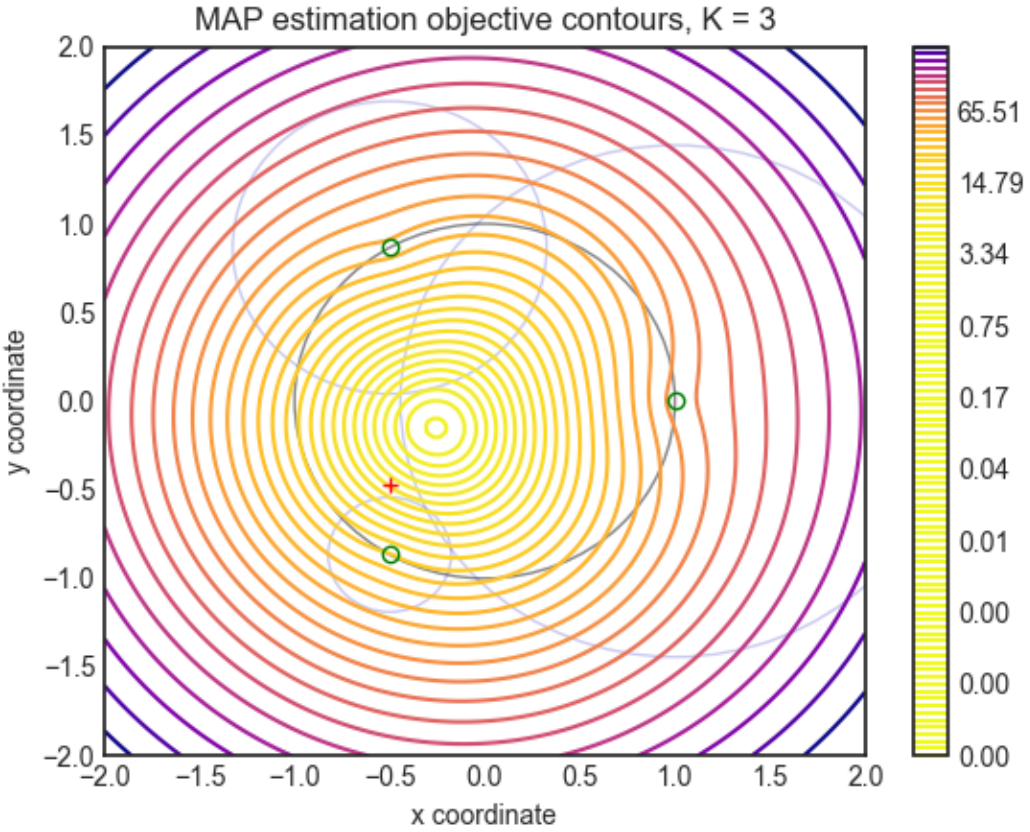


Fig.3.3: MAP Estimation Objective Contours, K=3

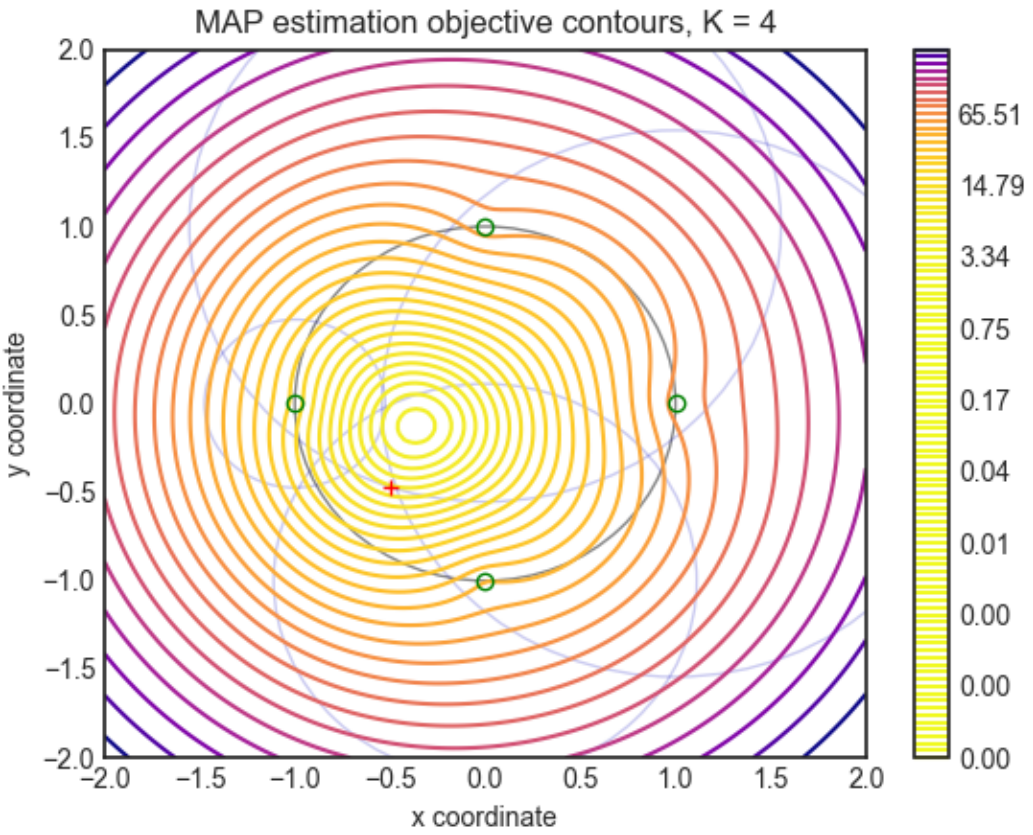


Fig.3.4: MAP Estimation of Objective Contours, K=4

The true vehicle coordinates are generated by the code as a random pair of coordinates inside the unit circle at the origin. It then calculates the distances from the true vehicle location for each value of K using K evenly spaced landmarks around the circle, truncates those measurements with additive Gaussian noise, computes the MAP estimation objective function for every point on a 128x128 mesh grid, and finally plots those values as equipluve contours. Also, it maps the landmark locations (represented by green circles), the ranges provided by each landmark, the unit circle (represented by a gray dot), and the correct location (shown by a red "+"). (Shown as faint blue circles around their respective landmarks).

Since all landmarks and the previous bias have a y-coordinate of 0, the MAP estimate of position is not very accurate for $K < 3$, where all estimates are symmetric around the x-axis. However, the estimate is significantly more precise for $K = 3$ and $K = 4$. The contour graph illustrates this, showing that the true location is only two and one contour levels, respectively, from the central estimate contour. In general, as K rises, the MAP estimate becomes more precise. While this is not always true, such as in the transition from $K = 1 \rightarrow 2$, where one of the landmark's range measurements happened to be low enough to overcome the more accurate estimate by the new second landmark, it becomes a stronger trend as K becomes very large. The accuracy of the estimator can be determined from the contour graph by measuring the distance between the true location and the point with the lowest contour, which is roughly in the center of the innermost contour.

In general, as K increases, so does the estimator's certainty. On the contour graphs, the estimator's certainty can be seen as a shrinking of the area of locations with a high probability. It is difficult to notice for such small values of K but following a single contour level as K increases (to about 40) clearly demonstrates this phenomenon.

Question 4

Question 4:

We have a classification problem with a choice for rejection:

$$\lambda(\alpha_i \mid \omega_j) = \begin{cases} 0, & i=j, \; i,j = 1,\ldots,c \\ \lambda_r, & i = c+1, \\ \lambda_s, & \text{otherwise} \end{cases}$$

The risk of taking action α_i is:

$$R(a_i \mid x) = \sum_{j=1}^{c} \lambda(\alpha_i \mid \omega_j) p(\omega_j \mid x)$$

So we have:

$$R(a_i \mid x) = \sum_{j=1}^{c} \lambda(\alpha_i \mid \omega_j) p(\omega_j \mid x)$$

$$= \sum_{j=1, j\neq i}^{c} \lambda_s p(\omega_j \mid x)$$

$$= \lambda_s \sum_{j=1, j\neq i}^{c} p(\omega_j \mid x)$$

$$= \lambda_s (1 - p(\omega_i \mid x))$$

For taking the rejection decision we have:

$$R(a_{c+1} \mid x) = \sum_{j=1}^{c} \lambda(a_{c+1} \mid \omega_j) p(\omega_j \mid x)$$

$$= \sum_{j=1}^{c} \lambda_r p(\omega_j \mid x)$$

$$= \lambda_r \sum_{j=1}^{c} p(\omega_j \mid x) = \lambda_r.$$

Action α_i is taken if its risk is smaller than the risk of taking another action α_j, $i \neq j$:

$$R(\alpha_i|x) \leq R(\alpha_j|x), \forall j = 1, \dots, C, j \neq i \Rightarrow$$

$$\lambda_s (1 - p(w_i|x)) \leq \lambda_s (1 - p(w_j|x)), \forall j = 1, \dots, C, j \neq i \Rightarrow$$

$$p(w_i|x) \geq p(w_j|x), \forall j = 1, \dots, C, j \neq i. \quad \hookrightarrow (1)$$

Also, the risk of action α_i has to be less than the risk of rejection. That is:

$$R(\alpha_i|x) \leq R(\alpha_{C+1}|x) \Rightarrow$$

$$\lambda_s (1 - p(w_i|x)) \leq \lambda_r \Rightarrow$$

$$p(w_i|x) \geq 1 - \frac{\lambda_r}{\lambda_s}. \quad \hookrightarrow (2)$$

If $\lambda_r = 0$, then for all the values of x, the system always rejects the decision, since this action has a cost of zero.

If $\lambda_r = \lambda_s$, then eq (2) is always true and we choose the class with the maximum posterior probability, according to eq (1).

The more λ_r is greater than λ_s, the less chances are that a feature be rejected by the system. If λ_r is much greater than λ_s, then the rejection decision will never be taken.

Question 5

Question 5:

Part 1:

The likelihood function for the observed data is given by:

$$P(D|\theta) = \pi[i = 1 \text{ to } N] P(z_i|\theta).$$

Since each sample is represented using a one hot encoded scheme, we have:

$$P(z_i|\theta) = \theta^T z_i$$

$$P(z_i|\theta) = \theta_k^{z_k} * \theta_k^{z_{2k}} * \theta_k^{z_{3k}} \ldots$$

Substituting this expression into the likelihood function and taking the logarithm, we get:

$$\log P(D|\theta) = \sum_{i=1}^{N} \sum_{k=1}^{K} z_{ik} \log \theta_k.$$

where z_{ik} is the k-th component of z_i, and is either 0 or 1 depending on whether the variable is in state k or not.

Taking the derivative of $\log L$ with respect to θ_k, we get:

$$\frac{\partial \log L(\theta|D)}{\theta_k} = \sum_{i=1}^{N} \frac{z_{i,k}}{\theta}.$$

Setting this derivative to zero, we get:

$$\theta_k = \frac{1}{N} \sum_{i=1}^{N} z_{i,k}$$

Thus, the maximum likelihood estimate for the parameter vector is, $\theta_{ML} = [\theta_1, \ldots, \theta_k]^T$.

Part 2 I

Maximum-a-Posteriori Estimation:

The prior distribution for the parameter vector is given by:

$$p(\theta) = \text{Dirichlet}(\theta|\alpha) = \frac{1}{B(\alpha)} \prod_{k=1}^{K} \theta_k^{\alpha_k - 1}$$

The posterior distribution given the observed data D is proportional to the product of the likelihood function and the priori distribution:

$$p(\theta|D) \propto P(D|\theta)p(\theta)$$

Substituting the expressions for the likelihood and prior and taking the logarithm, we get:

$$\log p(\theta|D) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} \log \theta_k + \sum_{k=1}^{K} (\alpha_k - 1) \log \theta_k.$$

Taking the partial derivative of the sum with respect to θ_k, we get:

$$\frac{\partial \log p(\theta|D)}{\theta_k} = \left(\sum_{n=1}^{N} z_{nk} + \alpha_k - 1 \right) / \theta_k.$$

Setting this derivative to zero, we get

$$\theta_k = \left(\sum_{n=1}^{N} z_{nk} + \alpha_k - 1 \right) / \left(N + \sum_{k'=1}^{K} \alpha_{k'} - K \right).$$

which is MAP estimate of the given distribution.

Appendix

Codes for all questions are affixed here:

Question 1:

```
dim=2; % Dimension of data

%Define datasets used for training and validation
D.d20.N=20;
D.d200.N=200;
D.d2k.N=2000;
D.d10k.N=10e3;
dType=fieldnames(D);

p=[0.6 0.4]; %prior probabilities

%Label 0

mu0=[-1 -1 ;1 1]';
Sigma0(:,:,1)=[1 0;0 1];
Sigma0(:,:,2)=[1 0;0 1];
alpha0=[0.5 0.5];

%Label 1
mu1=[-1 1 ;1 -1]';
Sigma1(:,:,1)=[1 0;0 1];
Sigma1(:,:,2)=[1 0;0 1];
alpha1=[0.5 0.5];
figure;

% Generate data

for index=1:length(dType)
    D.(dType{index}).x=zeros(dim,D.(dType{index}).N); %Initialize Data

    %Determine posteriors
    D.(dType{index}).labels = rand(1,D.(dType{index}).N)>=p(1);
    D.(dType{index}).N0=sum(~D.(dType{index}).labels);
    D.(dType{index}).N1=sum(D.(dType{index}).labels);

    D.(dType{index}).phat(1)=D.(dType{index}).N0/D.(dType{index}).N;
    D.(dType{index}).phat(2)=D.(dType{index}).N1/D.(dType{index}).N;


[D.(dType{index}).x(:,~D.(dType{index}).labels),D.(dType{index}).dist(:,~D.(d
Type{index}).labels)]=randGMM(D.(dType{index}).N0,alpha0,mu0,Sigma0);

[D.(dType{index}).x(:,D.(dType{index}).labels),D.(dType{index}).dist(:,D.(dTy
pe{index}).labels)]=randGMM(D.(dType{index}).N1,alpha1,mu1,Sigma1);

    subplot(2,2,index);

plot(D.(dType{index}).x(1,~D.(dType{index}).labels),D.(dType{index}).x(2,~D.(
dType{index}).labels),'b.','DisplayName','Class 0');
    hold all;
```

```
plot(D.(dType{index}).x(1,D.(dType{index}).labels),D.(dType{index}).x(2,D.(dT
ype{index}).labels),'r.','DisplayName','Class 1');

    grid on;
    xlabel('x1');ylabel('x2');
    title([num2str(D.(dType{index}).N) 'Samples From Both Classes']);
end

legend 'show';

%Part 1: Optimal Classifier

px0=evalGMM(D.d10k.x,alpha0,mu0,Sigma0);
px1=evalGMM(D.d10k.x ,alpha1,mu1,Sigma1);
discScore=log(px1./px0);
sortScore=sort(discScore);

% Generate vector of gammas

logGamma=[min(discScore)-eps sort(discScore)+eps];
prob=CalcProb(discScore,logGamma,D.d10k.labels,D.d10k.N0,D.d10k.N1,D.d10k.pha
t);
logGamma_ideal=log(p(1)/p(2));
decision_ideal=discScore>logGamma_ideal;
p10_ideal=sum(decision_ideal==1 & D.d10k.labels==0)/D.d10k.N0;
p11_ideal=sum(decision_ideal==1 & D.d10k.labels==1)/D.d10k.N1;
pFE_ideal=(p10_ideal*D.d10k.N0+(1-
p11_ideal)*D.d10k.N1)/(D.d10k.N0+D.d10k.N1);

% Estimate Minimum Error

[prob.min_pFE, prob.min_pFE_ind]=min(prob.pFE);
if length(prob.min_pFE_ind)>1
    [~,minDistTheory_ind]=min(abs(logGamma(prob.min_pFE_ind)-
logGamma_ideal));
    prob.min_pFE_ind=prob.min_pFE_ind(minDistTheory_ind);
end

% Find minimum gamma and corresponding false and true positive rates

minGAMMA=exp(logGamma(prob.min_pFE_ind));
prob.min_FP=prob.p10(prob.min_pFE_ind);
prob.min_TP=prob.p11(prob.min_pFE_ind);

plotROC(prob.p10,prob.p11,prob.min_FP,prob.min_TP);
hold all;
plot(p10_ideal,p11_ideal,'+','DisplayName','Ideal Minimum Error');
plotMinPFE(logGamma,prob.pFE,prob.min_pFE_ind);
plotDecisions(D.d10k.x,D.d10k.labels,decision_ideal);
plotERMContours(D.d10k.x,alpha0,mu0,Sigma0,alpha1,mu1,Sigma1,logGamma_ideal);

%Part 2: Maximum Likelihood Estimate

option=optimset('MaxFunEvals',3000,'MaxIter',1000);
for index=1:length(dType)
    lin.x=[ones(1,D.(dType{index}).N); D.(dType{index}).x];
```

```
    lin.init=zeros(dim+1,1);

[lin.theta,lin.cost]=fminsearch(@(theta)(costFun(theta,lin.x,D.(dType{index})
.labels)),lin.init,option);
    lin.discScore=lin.theta'*[ones(1,D.d10k.N); D.d10k.x];
    gamma=0;

lin.prob=CalcProb(lin.discScore,gamma,D.d10k.labels,D.d10k.N0,D.d10k.N1,D.d10
k.phat);

    plotDecisions(D.d10k.x,D.d10k.labels,lin.prob.decisions);
    lin.prob.pFE
    title(sprintf('Data and Classifier Decisions for Linear Logistic
Fit\nProbability of Error=%1.1f%%',100*lin.prob.pFE));

    %Using a Quadratic Fit
    quad.x=[ones(1,D.(dType{index}).N);
D.(dType{index}).x;D.(dType{index}).x(1,:).^2;
        D.(dType{index}).x(1,:).*D.(dType{index}).x(2,:);
        D.(dType{index}).x(2,:).^2];
    quad.init= zeros(2*(dim+1),1);

[quad.theta,quad.cost]=fminsearch(@(theta)(costFun(theta,quad.x,D.(dType{inde
x}).labels)),quad.init,option);
    quad.xScore=[ones(1,D.d10k.N); D.d10k.x; D.d10k.x(1,:).^2;
        D.d10k.x(1,:).*D.d10k.x(2,:); D.d10k.x(2,:).^2];
    quad.discScore=quad.theta'*quad.xScore;
    gamma=0;

quad.prob=CalcProb(quad.discScore,gamma,D.d10k.labels,D.d10k.N0,D.d10k.N1,D.d
10k.phat);

    plotDecisions(D.d10k.x,D.d10k.labels,quad.prob.decisions);
    title(sprintf('Data and Classifier Decisions for Quad Logistic
Fit\nProbability of Error=%1.1f%%',100*quad.prob.pFE));
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Functions used
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function cost=costFun(theta,x,labels)
h=1./(1+exp(-x'*theta));
cost=-1/length(h)*sum((labels'.*log(h)+(1-labels)'.*(log(1-h))));
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [x,labels] = randGMM(N,alpha,mu,Sigma)
d = size(mu,1); % dimensionality of samples
cum_alpha = [0,cumsum(alpha)];
u = rand(1,N); x = zeros(d,N); labels = zeros(1,N);
for m = 1:length(alpha)
ind = find(cum_alpha(m)<u & u<=cum_alpha(m+1));
x(:,ind) = randGaussian(length(ind),mu(:,m),Sigma(:,:,m));
labels(ind)=m-1;
end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function x = randGaussian(N,mu,Sigma)
% Generates N samples from a Gaussian pdf with mean mu covariance Sigma
n = length(mu);
z = randn(n,N);
A = Sigma^(1/2);
x = A*z + repmat(mu,1,N);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function gmm = evalGMM(x,alpha,mu,Sigma)
gmm = zeros(1,size(x,2));
for m = 1:length(alpha) % evaluate the GMM on the grid
gmm = gmm + alpha(m)*evalGaussian(x,mu(:,m),Sigma(:,:,m));
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function g = evalGaussian(x,mu,Sigma)
[n,N] = size(x);
C = ((2*pi)^n * det(Sigma))^(-1/2);
E = -0.5*sum((x-repmat(mu,1,N)).*(inv(Sigma)*(x-repmat(mu,1,N))),1);
g = C*exp(E);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function prob=CalcProb(discScore,logGamma,labels,N0,N1,phat)
for ind=1:length(logGamma)
prob.decisions=discScore>=logGamma(ind);
Num_pos(ind)=sum(prob.decisions);
prob.p10(ind)=sum(prob.decisions==1 & labels==0)/N0;
prob.p11(ind)=sum(prob.decisions==1 & labels==1)/N1;
prob.p01(ind)=sum(prob.decisions==0 & labels==1)/N1;
prob.p00(ind)=sum(prob.decisions==0 & labels==0)/N0;
prob.pFE(ind)=prob.p10(ind)*phat(1) + prob.p01(ind)*phat(2);
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function plotROC(p10,p11,min_FP,min_TP)
figure;
plot(p10,p11,'DisplayName','ROC Curve','LineWidth',2);
hold all;
plot(min_FP,min_TP,'o','DisplayName','Estimated Min. Error','LineWidth',2);
xlabel('Prob. False Positive');
ylabel('Prob. True Positive');
title('Mininimum Expected Risk ROC Curve');
legend 'show';
grid on; box on;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function plotMinPFE(logGamma,pFE,min_pFE_ind)
figure;
plot(logGamma,pFE,'DisplayName','Errors','LineWidth',2);
hold on;
plot(logGamma(min_pFE_ind),pFE(min_pFE_ind),...
'ro','DisplayName','Minimum Error','LineWidth',2);
xlabel('Gamma');
ylabel('Proportion of Errors');
title('Probability of Error vs. Gamma')
grid on;
```

```
legend 'show';
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function plotDecisions(x,labels,decisions)
ind00 = find(decisions==0 & labels==0);
ind10 = find(decisions==1 & labels==0);
ind01 = find(decisions==0 & labels==1);
ind11 = find(decisions==1 & labels==1);
figure; % class 0 circle, class 1 +, correct green, incorrect red

plot(x(1,ind00),x(2,ind00),'og','DisplayName','Class 0, Correct'); hold on,
plot(x(1,ind10),x(2,ind10),'or','DisplayName','Class 0, Incorrect'); hold on,
plot(x(1,ind01),x(2,ind01),'+r','DisplayName','Class 1, Correct'); hold on,
plot(x(1,ind11),x(2,ind11),'+g','DisplayName','Class 1, Incorrect'); hold on,
axis equal,
grid on;
title('Data and their classifier decisions versus true labels');
xlabel('x_1'), ylabel('x_2');
legend('Correct decisions for data from Class 0',...
'Wrong decisions for data from Class 0',...
'Wrong decisions for data from Class 1',...
'Correct decisions for data from Class 1');
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function
plotERMContours(x,alpha0,mu0,Sigma0,alpha1,mu1,Sigma1,logGamma_ideal)
horizGrid = linspace(floor(min(x(1,:))),ceil(max(x(1,:))),101);
vertGrid = linspace(floor(min(x(2,:))),ceil(max(x(2,:))),91);
[h,v] = meshgrid(horizGrid,vertGrid);
discrimScoreGridVals =log(evalGMM([h(:)';v(:)'],alpha1,mu1,Sigma1))-
log(evalGMM([h(:)';v(:)'],alpha0,mu0,Sigma0)) - logGamma_ideal;
minDSGV = min(discrimScoreGridVals);
maxDSGV = max(discrimScoreGridVals);
discrimScoreGrid = reshape(discrimScoreGridVals,91,101);
contour(horizGrid,vertGrid,discrimScoreGrid,[minDSGV*[0.9,0.6,0.3],0,[0.3,0.6
,0.9]*maxDSGV]); % plot equilevel contours
% including the contour at level 0
legend('Correct decisions - Class 0',...
    'Incorrect decisions - Class 0',...
    'Incorrect decisions - Class 1',...
    'Correct decisions - Class 1',...
    'Equilevel contours of discriminant function' ),
end
```

Question 2:

```
%% Question 2

%generate data using the given function
nTr = 100;
nVal = 1000;
[xTr,yTr,xVal,yVL] = hw2q2(nTr,nVal);

%get the x1 and x2 training and validation samples
x1Tr = xTr(1,:);
x2Tr = xTr(2,:);
x1Val = xVal(1,:);
x2Val = xVal(2,:);

%cubic vectors for validation and training set
xTrVect = [ones(1,nTr); x1Tr; x2Tr; x1Tr.^2; x1Tr.*x2Tr; x2Tr.^2; x1Tr.^3;
x1Tr.^2 .* x2Tr;x1Tr.*x2Tr.^2;x2Tr.^3];
xValVect = [ones(1,nVal); x1Val; x2Val; x1Val.^2; x1Val.*x2Val; x2Val.^2;
x1Val.^3; x1Val.^2 .* x2Val;x1Val.*x2Val.^2;x2Val.^3];

%calculate terms derived from textbook
R = (xTrVect * xTrVect')/nTr;
Q = (xTrVect * yTr')/nTr;
wmle = inv(R)*Q;

%calculate mse
MSEmle = mean((yVL - wmle'*xValVect).^2);

%MAP Estimate with gamma
gamma = 10^(-4):.1:10^4;
wmap = zeros(10,length(gamma));
MSEmap=zeros(10,length(gamma));
parameter = zeros(1, length(gamma));
sigma  = 1;
figure;
for i = 1 : length(gamma)
    parameter(i) = (sigma^2)/(nTr*(10^gamma(i)));
    wmap(:,i) = inv(R+parameter(i)*eye(10))*Q;
    MSEmap(:,i) = mean((yVL-wmap(:,i)'*xValVect).^2);
    plot(gamma(i), MSEmle,'r.', gamma(i), MSEmap(:,i),'b.');
    hold on;
    title('MSE vs Gamma')
    xlabel('log(gamma)'); ylabel('MSE');
    legend('MAP Estimator', 'ML Estimator');

end

function [xTrain,yTrain,xValidate,yValidate] = hw2q2(Ntrain,Nvalidate)
%Ntrain = 100;
data = generateData(Ntrain);
figure(1), plot3(data(1,:),data(2,:),data(3,:),'.'), axis equal,
xlabel('x1'),ylabel('x2'), zlabel('y'), title('Training Dataset'),
xTrain = data(1:2,:); yTrain = data(3,:);
%Nvalidate = 1000;
data = generateData(Nvalidate);
figure(2), plot3(data(1,:),data(2,:),data(3,:),'.'), axis equal,
```

```
xlabel('x1'),ylabel('x2'), zlabel('y'), title('Validation Dataset'),
xValidate = data(1:2,:); yValidate = data(3,:);
end
%%
function x = generateData(N)
gmmParameters.priors = [.3,.4,.3]; % priors should be a row vector
gmmParameters.meanVectors = [-10 0 10;0 0 0;10 0 -10];
gmmParameters.covMatrices(:,:,1) = [1 0 -3;0 1 0;-3 0 15];
gmmParameters.covMatrices(:,:,2) = [8 0 0;0 .5 0;0 0 .5];
gmmParameters.covMatrices(:,:,3) = [1 0 -3;0 1 0;-3 0 15];
[x,labels] = generateDataFromGMM(N,gmmParameters);
end
%%
function [x,labels] = 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 = size(gmmParameters.meanVectors,1); % Data dimensionality
C = length(priors); % Number of components
x = zeros(n,N); labels = zeros(1,N);
% Decide randomly which samples will come from each component
u = rand(1,N); thresholds = [cumsum(priors),1];
for l = 1:C
    indl = find(u <= thresholds(l)); Nl = length(indl);
    labels(1,indl) = l*ones(1,Nl);
    u(1,indl) = 1.1*ones(1,Nl); % these samples should not be used again
    x(:,indl) = mvnrnd(meanVectors(:,l),covMatrices(:,:,l),Nl)';
end
end
```

Question 3:

```python
#!/usr/bin/env python3

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm

SIGMA_X = 0.25
SIGMA_Y = 0.25
SIGMA_I = 0.3

CONTOUR_LEVELS = np.geomspace(0.0001, 250, 100)

# Returns a random xy pair within the unit circle centered at the origin.
def random_unit_circle_coords():
    # Polar coordinates with a square-rooted r-value produce a uniform distribution
    r = np.sqrt(np.random.uniform(0, 1))
    theta = np.random.uniform(0, 2) * np.pi
    x = r * np.cos(theta)
    y = r * np.sin(theta)
    return np.array([x, y])

# Generates noisy measurements for all K landmarks given the true position.
def get_range_measurements(K, xy_true):
    return [generate_measurement(landmark_pos(i, K), xy_true) for i in range(K)]

# Returns the xy coordinate pair of the i'th landmark out of K total landmarks.
def landmark_pos(i, K):
    angle = 2 * np.pi / K * i
    x = np.cos(angle)
    y = np.sin(angle)
    return np.array([x, y])

# Generates a range measurement between the given landmark and true positions,
# with random Gaussian noise.
def generate_measurement(xy_landmark, xy_true):
    dTi = np.linalg.norm(xy_true-xy_landmark)
    while True:
        noise = np.random.normal(0, SIGMA_I)
        measurement = dTi + noise
        if measurement >= 0:
            return measurement

# Creates an equilevel contour plot for the MAP estimation objective function
# given a set of range measurements
def plot_equilevels(range_measurements, xy_true):
    # First, create a mesh grid of values from the objective function
```

```
    gridpoints = np.meshgrid(np.linspace(-2, 2, 128), np.linspace(-2, 2, 128))
    contour_values = MAP_objective(gridpoints, range_measurements)

    # Then, set up the plot
    plt.style.use('seaborn-white')

    ax = plt.gca()

    unit_circle = plt.Circle((0, 0), 1, color='#888888', fill=False)
    ax.add_artist(unit_circle)

    plt.contour(gridpoints[0], gridpoints[1], contour_values, cmap='plasma_r',
levels=CONTOUR_LEVELS);

    for (i, r_i) in enumerate(range_measurements):
        (x, y) = landmark_pos(i, len(range_measurements))
        plt.plot((x), (y), 'o', color='g', markerfacecolor='none')
        # I added faint blue circles to demonstrate the range from each landmark
        range_circle = plt.Circle((x, y), r_i, color='#0000bb33', fill=False)
        ax.add_artist(range_circle)

    ax.set_xlabel("x coordinate")
    ax.set_ylabel("y coordinate")
    ax.set_title("MAP estimation objective contours, K = " +
str(len(range_measurements)))

    ax.set_xlim((-2, 2))
    ax.set_ylim((-2, 2))
    ax.plot([xy_true[0]], [xy_true[1]], '+', color='r')
    plt.colorbar();
    plt.show()

# Calculates values of the MAP estimation objective function on a given mesh
# grid of input x-y coordinate pairs.
def MAP_objective(xy, range_measurements):
    # The shape of xy is (2, n, m), but it needs to be (n, m, 1, 2).
    xy = np.expand_dims(np.transpose(xy, axes=(1, 2, 0)), axis=len(np.shape(xy))-1)

    prior = np.matmul(xy, np.linalg.inv(np.array([[SIGMA_X**2, 0],[0, SIGMA_Y**2]])))
    prior = np.matmul(prior, np.swapaxes(xy, 2, 3))
    prior = np.squeeze(prior)
    # prior is now of shape (n, m).

    range_sum = 0

    for (i, r_i) in enumerate(range_measurements):
        xy_i = landmark_pos(i, len(range_measurements))
        d_i = np.linalg.norm(xy - xy_i[None, None, None, :], axis=3)
        range_sum += np.squeeze((r_i - d_i)**2 / SIGMA_I**2)
```

```
    return prior + range_sum


xy_true = random_unit_circle_coords()

for K in [1, 2, 3, 4]:
    range_measurements = get_range_measurements(K, xy_true)
    plot_equilevels(range_measurements, xy_true)
```

Citations and References:
Notes and Codes by Prof. Deniz Erdogmus, GitHub.