

EECE 5644 Assignment 2

Ishaan Desai (desai.is@northeastern.edu)

October 27th, 2025

Professor Erdoganmus

GitHub Link:

<https://github.com/ishaandesai0/EECE-5644/tree/main/Assignment%202>

Question 1

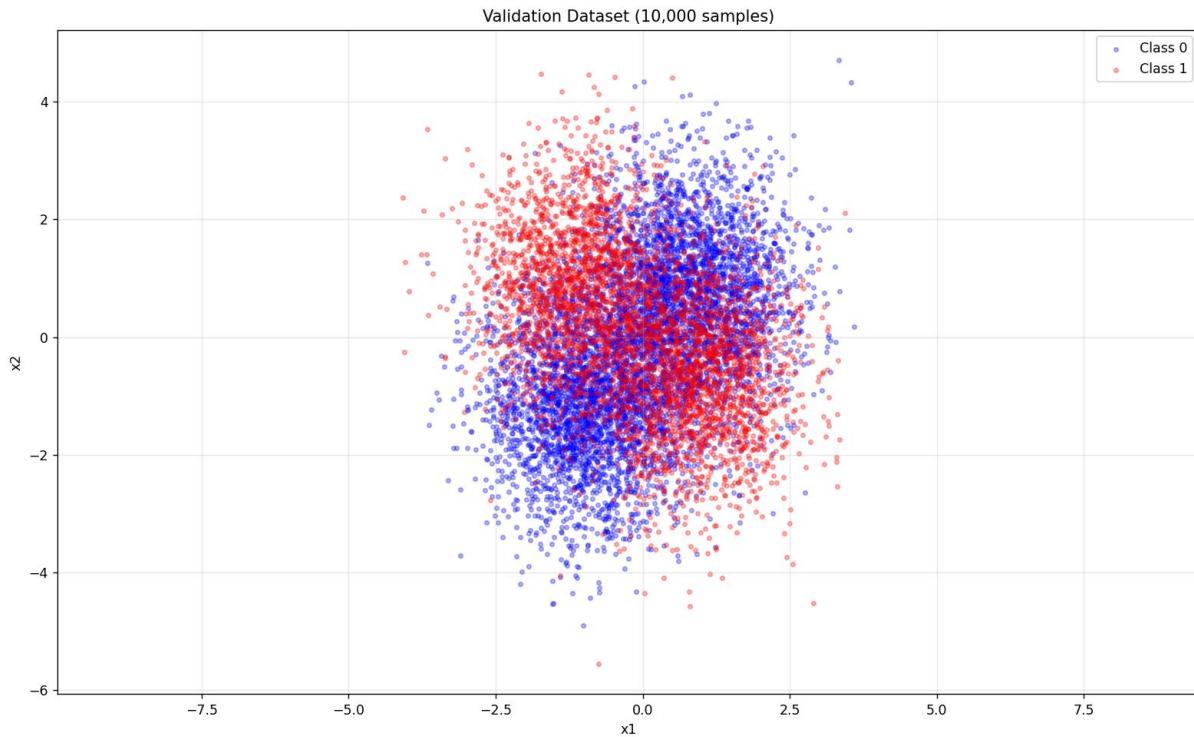


Figure 1: 10,000 Sample Values Validation Dataset

This problem had me tasked with classifying 2D data points into two classes based on a known probability distribution. Data was generated from a mixture of Gaussian distributions with priors $P(L=0) = 0.6$ and $P(L=1) = 0.4$. Each class-conditional distribution is a mixture of two Gaussian components with equal weights of 0.5. The mixture components for each class were centered at the following:

$$\mathbf{m}_{01} = \begin{bmatrix} -0.9 \\ -1.1 \end{bmatrix} \quad \mathbf{m}_{02} = \begin{bmatrix} 0.8 \\ 0.75 \end{bmatrix} \quad \mathbf{m}_{11} = \begin{bmatrix} -1.1 \\ 0.9 \end{bmatrix} \quad \mathbf{m}_{12} = \begin{bmatrix} 0.9 \\ -0.75 \end{bmatrix} \quad \mathbf{C}_{ij} = \begin{bmatrix} 0.75 & 0 \\ 0 & 1.25 \end{bmatrix} \text{ for all } \{ij\} \text{ pairs.}$$

All of the Gaussian components shared the same covariance matrix with diagonal elements [0.75, 1.25]. This made the distributions have different spreads along the x1 and x2 axes. This created a complex X shaped decision boundary challenging simple classifiers. This question had 4 independent datasets created. Training sets of 50, 500, 5000 samples and a validation set of 10,000 samples which can be seen above in Figure 1.

1.A Theoretical Optimal Classifier

The theoretical optimal classifier is derived from Bayes decision theory and achieves the minimum probability of error by maximizing the posterior probability. According to Bayes' rule,

we should choose class 1 if:

$$p(x|L=1)P(L=1) > p(x|L=0)P(L=0)$$

This can also be expressed as a likelihood ratio test, as in choosing class 1 if the ratio is greater than 1.5. Since we know the true probability distributions, we can compute the likelihoods for any input. The class-conditional densities are computed as weighted sums of the two Gaussian components for each class. The ROC curve can be seen below in Figure 2:

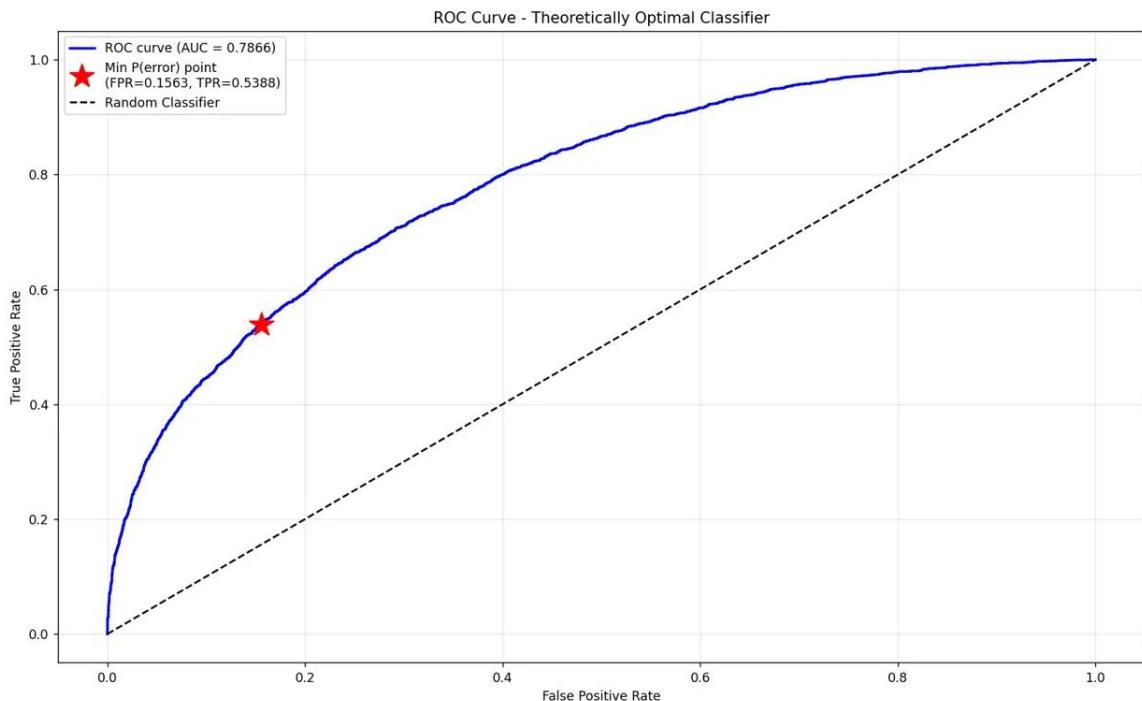


Figure 2: ROC Curve

I estimated the minimum achievable probability of error to be 0.2778. The confusion matrix shows 5075 true negatives, 940 false positives, 1838 false negatives, and 2147 true positives. The ROC curve, which was generated by varying the decision threshold on the likelihood ratio, gave an area under curve of 0.7866, showing a good discriminative performance. The operating point of minimum-error classifier is marked on the curve at a FPR of 0.1563 and a TPR of 0.5388, corresponding to the threshold of 1.5.

The high error rate of 27.78% is not due to classifier inadequacy but rather reflects the overlap between the two-class distributions. When the class-conditional densities overlap significantly in this X-shaped configuration, no classifier can achieve perfect separation. This error rate is the

Bayes error, the theoretical lower bound on classification error for this problem. Any practical classifier we make should be evaluated against this benchmark

1.B Logistic Linear Classification Models

The logistic linear model attempts to approximate the class posterior probability $P(L=1|x)$ using a linear function of the input features. Specifically, we transform each 2D input $x = [x_1, x_2]^T$ into a 3D feature vector $z(x) = [1, x_1, x_2]^T$ by adding a bias term. The model then predicts the class-1 posterior as $h(x,w) = 1 / (1 + \exp(-w^T z(x)))$, where w is a 3 parameter weight vector to be learned. This is the standard logistic regression model, producing a linear decision boundary in the original 2D feature space.

Three separate logistic-linear models were trained using maximum likelihood estimation. Training sets of 50, 500, and 5000 samples were used. This approach minimizes the negative log-likelihood of the training data, equivalent to minimizing the binary cross-entropy loss. I used the BFGS optimization algorithm, which efficiently finds the optimal weight vector. Each of the trained models were then evaluated on the 10,000 sample validation set to estimate generalization performance.

The results showed a fundamental limitation of the logistic-linear model. With 50 training samples, the model achieved a validation error of 40.40%. 500 samples had 38.64%, and 5000 surprisingly went up to 39.82%. The confusion matrices showed that all three models struggled to classify Class 1 samples correctly.

This performance indicated severe underfitting. The linear decision boundary imposed by the model is fundamentally incapable of capturing X-shaped optimal decision region. No matter how much training data was provided, the model couldn't overcome this limitation. The training set size of 5000 is more than enough to reliably estimate 3 parameters yet there was no improvement. The model is essentially learning to classify almost everything as Class 0. The high error rates are also significantly worse than the optimal rate found earlier. The linear model's inability to represent a curved boundary makes it a bad choice for this dataset.

1.C Logistic Quadratic Classification Models

The logistic quadratic model extends the linear model by adding quadratic terms to the feature space. We are now using $z(x) = [1, x_1, x_2, x_1^2, x_1x_2, x_2^2]^T$. This 6D feature vector includes all second order polynomial terms of the original features. The logistic function $h(x,w) = 1/(1 + \exp(-w^T z(x)))$ remains the same but operates on the expanded feature space. While the decision boundary is still linear in the 6D feature space, when projected back to the original 2D space, it becomes a conic section. This allows the model to represent curved and possibly X-shaped decision boundaries.

Training went almost identically to the linear case using MLE and BFGS optimization but now estimates 6 parameters instead of 3. This increased model capacity comes with a theoretical risk of overfitting, especially with smaller training sets. However, if the true decision boundary is curved, the quadratic model will be better suited to capture this structure.

The quadratic model significantly outperformed the linear model. With 50 training samples, it achieved a validation error of 28.47%. 500 samples brought us to 28.14%, and 5000 samples achieving 27.97%. The confusion matrices showed much more balanced performance with a reasonable number of both true positives and true negatives, meaning the model successfully used both features in decision making.

The consistent improvement with more samples indicated that the model was functioning properly. It had enough capacity to represent the decision boundary, and higher sample sizes allowed it to converge toward the optimal solution. The progression in validation error percentages showed the expected learning curve behavior where performance approaches the Bayes error asymptote as sample size increased.

The quadratic model with only 50 samples was substantially better than the linear model with 5000 samples, highlighting the importance of model capacity. The quadratic model's 6 parameters are sufficient to capture the essential curved structure of the decision boundary, while the 3 parameters of the linear model can't. This is a clear example of the bias-variance tradeoff favoring a more complex model: the quadratic model's higher variance is very much outweighed by its lower bias.

1.D Discussion

The contrast between logistic linear and logistic quadratic performance shows the fundamental principles of ML and model selection. The linear model suffered from high bias, as its functional form is too restrictive to capture the true decision boundary. This is underfitting, as no amount of data will make up for the model family simply not containing a good solution. The error rate plateaued around 40%, significantly higher than the achievable 28%. In contrast, the quadratic model had the appropriate capacity for this problem. Being able to represent curved boundaries matched the structure of the problem and allowed it to approach optimal performance with sufficient data.

The sample efficiency comparison also stands out. The quadratic model with 50 samples had 28.47% error while the linear model with 5000 samples achieved 39.82% error. The better model with 1% of the data outperformed the worse model with 100% of the data. This could be applied and provide strong cost savings. Collecting labeled data can be expensive, so choosing the right model architecture can reduce data requirements by orders of magnitude. This is especially applicable in fields like medical imaging and autonomous driving.

These results also show the relationship between model complexity and problem complexity. The true decision boundary had an X-shaped curved structure. The optimal boundary cannot be approximated well by a line but can be approximated by a conic section. The quadratic model learned to place an approximately X-shaped conic section to separate the classes. The matching of model capacity to problem complexity is important for model selection.

From a bias-variance standpoint, the linear model had low variance but high bias. The quadratic model had higher variance but lower bias. The total error is the sum of variance and squared bias, and in this problem, the bias reduction from using quadratic features far exceeds the variance increase. With 5000 samples, the quadratic model's variance became negligible, and we achieved the bias-limited performance.

Finally, the results highlight the importance of domain knowledge and exploratory analysis. If we tried linear models and saw the 40% error, we may have concluded the features not being predictive. Visualizing the data revealed the X-shaped pattern, suggesting that the curved boundaries are needed. The lesson here is that model selection should be informed by understanding the problem structure, not just blindly trying different kinds of models and selecting based on validation performance on its own.

Question 2

This problem explored parameter estimation for a regression model where we must learn the relationship between 2D input vectors x ($[x_1, x_2]^T$) and scalar output y . The assumed model is $y=c(x, w) + v$, where $c(x, w)$ is a cubic polynomial in the input variables and v is zero-mean Gaussian noise with variance (σ^2). The cubic polynomial expansion in two variables includes 10 terms: the constant, two linear terms, three quadratic terms, and four cubic terms. As a result, the weight vector w has 10 components to be estimated.

Before starting the problem, we were told that true data distribution does not actually follow this model. Data was generated from a 3D Gaussian mixture model with 3 components, where the first two dimensions are inputs and the third was output. This model mismatch was intentional and also realistic. In practice, our assumed model structure rarely matches the true data generation process. We have to figure out how to best estimate the parameters of our approximate model given the mismatch, as well as how regularization can help.

I generated a training set of 100 samples and a validation set of 1000 samples from the Gaussian mixture. The 3D scatter plots of both datasets reveal the underlying structure: there are three distinct clusters corresponding to the three mixture components, with clear relationships between inputs/outputs within each cluster. However, the overall relationship is more complex than a single cubic polynomial can capture perfectly, ensuring that even the best model will have some

irreducible error. The training data and validation data plots can be seen below in Figures 3 and 4:

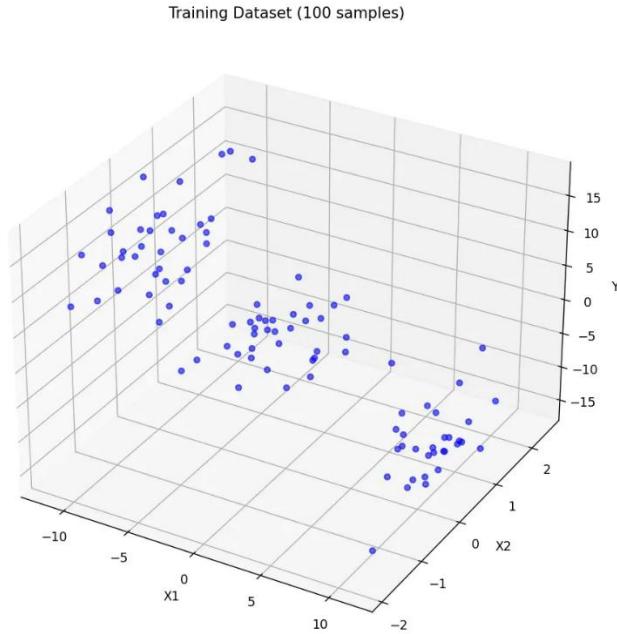


Figure 3: Training Dataset

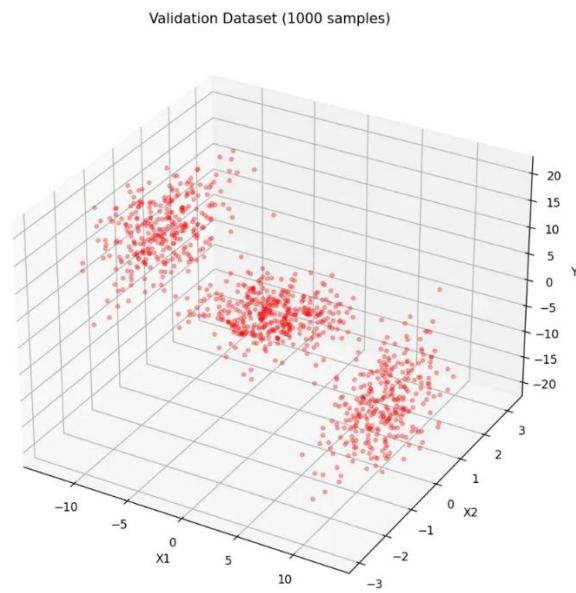


Figure 4: Validation Dataset

The ML estimator was derived by maximizing the likelihood of observing the training data under model assumptions. Given the model $y = c(x, w) + v$ where $v \sim N(0, \sigma^2)$, the likelihood of a single observation (x_i, y_i) is $p(y_i|x_i, w, \sigma^2) = N(y_i | c(x_i, w), \sigma^2)$, a Gaussian centered at the predicted value $c(x_i, w)$. For N independent samples, the total likelihood is the product of individual likelihoods.

Taking the log to obtain log-likelihood makes the product become a sum and remove exponentials, yielding the equation:

$$l(w) = -\frac{N}{2} * \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} * \sum_i [y_i - c(x_i, w)]^2$$

The first term is constant with respect to w , so maximizing is equivalent to the sum of squared errors.

To express this in matrix form, we construct the design matrix Φ where each row I contains the cubic feature expansion of input x_i . The predictions are then Φw , and the squared error is $\|y - \Phi w\|^2$. Taking the gradient with respect to w and setting it to zero allows us to obtain the closed form ML solution: $w_{ML} = (\Phi^T \Phi)^{-1} \Phi^T y$. This is the classic least squares estimator.

I implemented this directly in Python, constructing the design matrix from provided training data, computing $\Phi^T \Phi$ and $\Phi^T y$, and solving the linear system with NumPy's `linalg.solve` for numerical stability. The resulting vector defined the trained cubic polynomial model.

Evaluating this model on training set yielded a mean squared error of 3.2243. This showed how well the cubic polynomial fits the 100 training points. Applying the same model to the validation set gave an MSE of 4.8862, much higher than the training error. This indicates that the model is overfitting.

The overfitting is not too severe such that it causes very bad predictions but is substantial enough to warrant regularization. The sample to parameter ratio is not very generous. The model has enough flexibility to fit the training data well, but some of that is likely capturing noise over signal.

MAP Estimator

The MAP estimator incorporates prior knowledge about the parameters through Bayesian inference. We assume a prior distribution over the weights: $w \sim N(0, \gamma I)$, a zero-mean Gaussian with covariance γI where γ is the prior variance and I is the identity matrix. This prior gives a belief that parameters should be small in magnitude, with a typical scale determined by γ .

By Bayes' rule, the posterior distribution is proportional to the likelihood times the prior: $p(w|D, \sigma^2, \gamma) \propto p(D|w, \sigma^2) \cdot p(w|\gamma)$. MAP estimate maximizes this posterior, or equally, minimizes the negative log-posterior.

Dropping constants and the common factor of $\frac{1}{2}$, the MAP objective can be written as: minimize $\sum_i [y_i - c(x_i, w)]^2 + (\sigma^2/\gamma) \cdot w^T w$. This is the sum of squared errors plus a regularization term penalizing large weights. The coefficient $\lambda = \sigma^2/\gamma$ controls the strength of regularization: large λ means strong regularization, small λ means weak regularization.

The key challenge in MAP estimation is selecting the prior variance γ . We assume measurement noise variance = 1 and vary γ over a wide range from 10^{-4} to 10^4 , testing 50 logarithmically spaced values. For each γ , we train a model using the MAP formula and evaluate it on both training and validation sets. Figure 5 below shows the MSE vs Gamma Plots:

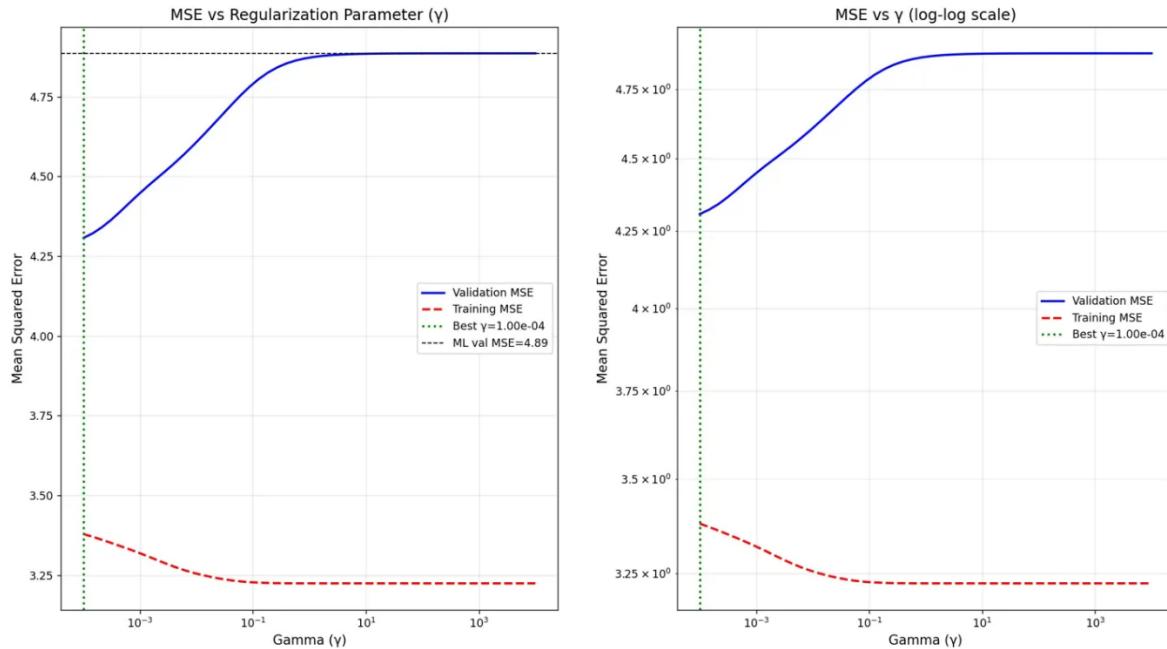


Figure 5: MSE vs Gamma Plots (Semi-log and Log-log)

These plots reveal a clear trend. At very small γ , the validation MSE is high because the model is constrained too heavily. The weights are forced to be so small that the model underfits. As γ increases, validation error initially decreases as the model gains enough flexibility to capture true patterns. The minimum validation MSE of 4.3075 occurs at $\gamma = 10^{-4}$. Further increasing γ causes validation error to rise again once the model begins to overfit, eventually approaching the ML solution's validation error of 4.8862 as γ grows to infinity.

The training error curve shows the opposite trend: it's higher when regularization is strong and decreases as regularization weakens, reaching the ML training error of 3.2243 as γ approaches

infinity. This was expected as regularization trades off training fit for better generalization. The key insight is that the optimal γ balances the two effects.

At optimal $\gamma = 10^{-4}$, the MAP model achieved a validation MSE of 4.3075 compared to ML's 4.8862, an improvement of 11.84%. This is a big improvement, as the regularized model makes predictions on average 12% closer to true values. The training MSE at optimal γ is 3.3787, a little worse than ML's 3.2243. This shows that we sacrificed a small amount of training fit to gain a large improvement in validation performance.

Optimal γ is at the left edge of our search range, suggesting that even stronger regularization might help slightly. This makes sense given the sample size of 100 and parameter count of 10. The model has enough capacity to memorize noise, so aggressive regularization could be beneficial. The optimal γ of 10^{-4} is a very large regularization coefficient, showing that the prior belief of “weights being small” is well justified.

Discussion

The ML and MAP comparison revealed fundamental machine learning principles. ML achieved optimal training fit but overfits to noise, resulting in poor generalization. With only 100 samples and 10 parameters, the model captures spurious training patterns that don't reflect the true underlying function.

MAP estimation addressed this through regularization. The Gaussian prior $p(w|\gamma) = N(0, \gamma I)$ penalizes large weights, encouraging simpler models. This regularization trades 3.4% worse training error for 11.8% better validation error. The optimal γ of 10^{-4} showed strong regularization is beneficial for this sample-to-parameter ratio.

This exemplifies the bias-variance tradeoff: ML has low bias but high variance, while MAP increases bias slightly but substantially reduces variance. For finite samples, the tradeoff favors MAP.

The model mismatch – true data from a Gaussian mixture, not cubic polynomial – adds complexity. Even if we had infinite data, our model can't recover the true function. However, regularization helps by preventing the polynomial from contorting to fit noise, yielding smoother approximations that generalize better.

These results emphasize regularization's necessity in real world ML. Model mismatch is common, and samples are finite. Cross-validation or validation performance provides the standard approach for tuning regularization strength, balancing training fit with generalization.

Question 3

This question explores MAP estimation for vehicle localization using noisy range measurements to known landmarks. The vehicle position $[x, y]^T$ is unknown, with K landmarks placed evenly on a unit circle. Range measurements $r_i = d_i + n_i$ are corrupted by Gaussian noise ($\sigma = 0.3$), where d_i is the true Euclidean distance. A Gaussian prior centered at the origin ($\sigma_x = \sigma_y = 0.25$) gives prior belief about the vehicle's likely location. The true position is $[0.3, 0.4]^T$. The task here was to derive the map function objective, implement for K=1, 2, 3, and 4, and analyze how localization accuracy improves with more measurements.

The likelihood of measurement r_i given position $[x, y]^T$ is $r_i \sim N(d_i, \sigma_i^2)$, where $d_i(x,y) = \sqrt{[(x-x_i)^2 + (y-y_i)^2]}$ is the true distance. The log-likelihood for K measurements is $\sum_i \log p(r_i|x,y) = -1/2 \cdot \sum_i [r_i - d_i(x,y)]^2/\sigma_i^2 + \text{const}$. The minimized MAP objective:

$$J(x, y) = \sum_i [r_i - \sqrt{[(x - x_i)^2 + (y - y_i)^2]}]^2/\sigma_i^2 + x^2/\sigma_x^2 + y^2/\sigma_y^2$$

The first term penalizes positions inconsistent with measurements; the second term penalizes positions far from the prior mean [0,0]. The MAP estimate balances measurement fit with prior belief.

I implemented the MAP objective function in Python, computing $d_i(x,y)$ for each landmark and summing the terms. To find the MAP estimate for a given set of measurements, I evaluate $J(x,y)$ on a dense grid of candidate positions and identify the grid point with minimum objective value. This grid search is computationally simple and produces high quality visualizations, although more sophisticated optimization methods could find the minimum more efficiently.

For each value of K, we place K landmarks evenly spaced on the unit circle. For K=1, the single landmark is at [1,0]; for K=2, they're at [1,0] and [-1,0]; for K=3, they're at 120 degree intervals; for K=4, they're at the four cardinal directions [1,0], [0,1], [-1,0], and [0,-1]. This placement ensures fair coverage and avoids biasing results toward any particular direction.

I generate noisy range measurements from the true vehicle position by computing true distances and adding Gaussian noise. If a measurement turns out negative, we reject it and resample it until it gets a non-negative value. This ensures all measurements are realistic.

For each K, we create a contour plot showing level curves of the MAP objective function $J(x,y)$. These contours reveal the shape of the objective landscape – how certain/uncertain the estimate is and in which directions. We superimpose the true vehicle location (red +), MAP estimate (green star), and landmark locations (white circles) on these plots for easy interpretation. We also compute localization error to quantify performance. Figure 6 below shows the contour plot for K = 1:

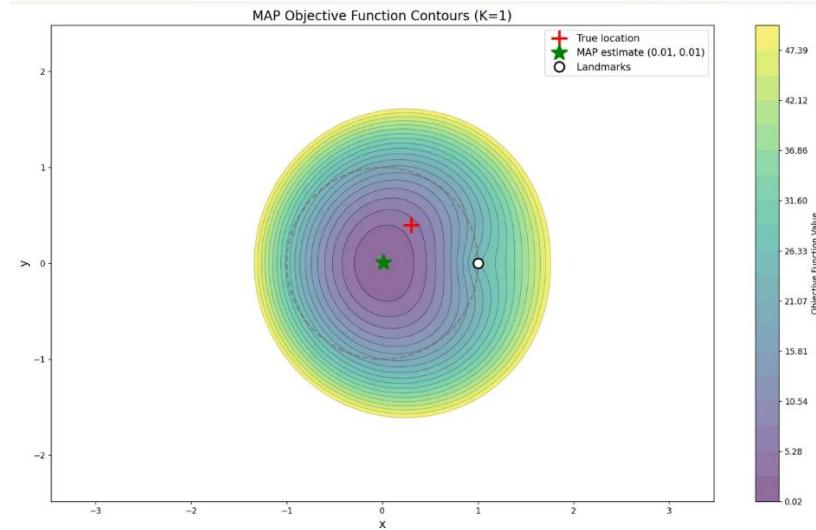


Figure 6: K = 1 Contour Plot

With only one landmark, the system has limited information. The MAP estimate of [0.010, 0.010], very close to the origin, giving a localization error of 0.486. The contour plot shows nearly circular contours centered close to the origin. This symmetry indicates that the single range measurement provides only radial information – we know the vehicle is approximately a certain distance from [1,0], but this defines a circle of possible positions. The prior breaks this symmetry somewhat by pulling the estimate toward the origin, but without multiple landmarks to triangulate, we can't localize accurately. The MAP estimate essentially reflects a compromise between the weak directional information from one measurement and the strong prior centered at the origin. Figure 7 below shows K = 2:

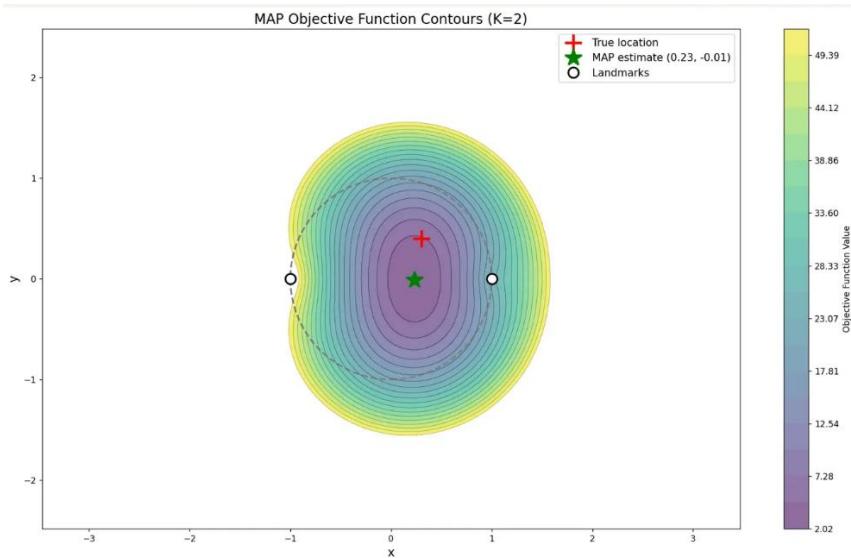


Figure 7: K = 2 Contour Plot

With landmarks at opposite sides of the circle, performance improves substantially. The MAP estimate moves to [0.231, -0.010], closer to the true location, though the y-coordinate is wrong in sign. The localization error improves to 0.416, a 14% reduction from K = 1. The contours become elliptical, elongated vertically. This makes sense since the two horizontal axis landmarks provide strong x coordinate information, but weak y coordinates. The K = 3 contour can be seen below in Figure 8:

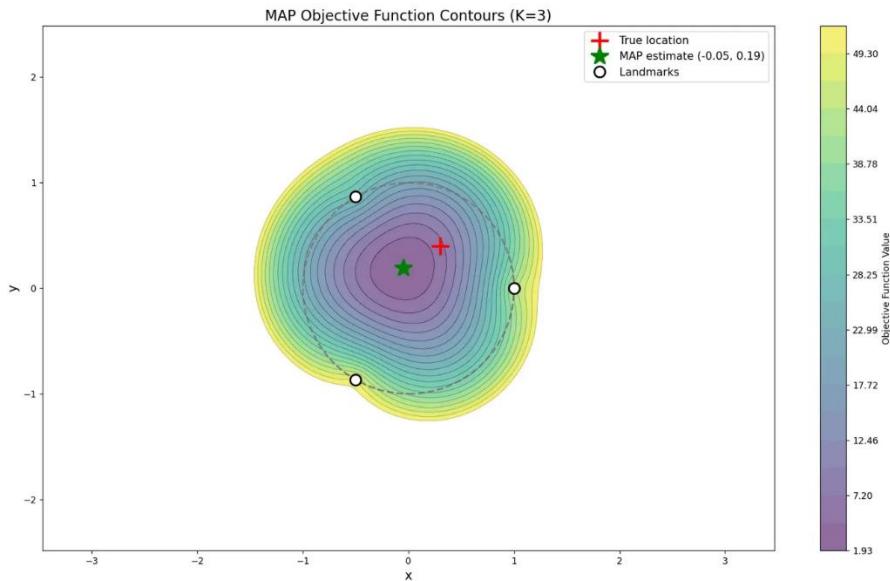


Figure 8: K = 3 Contour

Adding a third landmark with all 3 spaced out 120 degrees further improves the estimate. The MAP estimate is [-0.050, 0.191] with error 0.408. While there is only a slight improvement, the contours tighten a lot, and the elliptical shape becomes more circular. Three landmarks provide true triangulation: the intersection of three circles determines a unique point. The remaining error is due to measurement noise and the influence of the prior. The estimate is pulled slightly toward the origin. The K = 4 contour can be seen below in Figure 9:

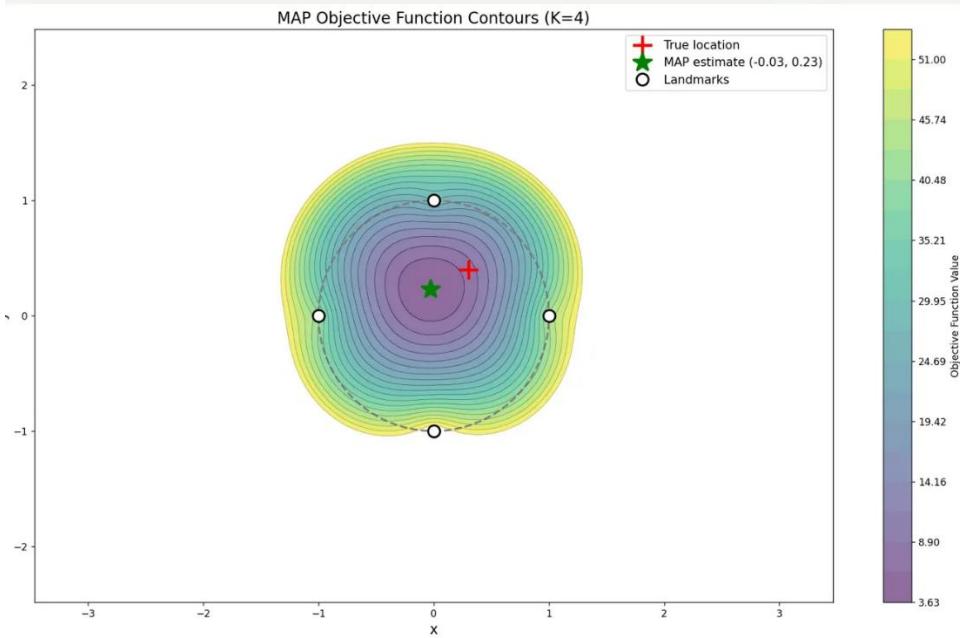


Figure 9: K = 4 Contour

With four landmarks symmetrically placed at the cardinal directions, we achieve the best performance: MAP estimate $[-0.030, 0.231]$ with error 0.371, a 24% improvement over $K = 1$. The contours are the tightest here, showing high certainty. Four measurements provide some redundancy. Two measurements suffice for 2D positions, so four give us an overdetermined system that averages out measurement noise. The symmetric placement at cardinal directions provide balanced info in all directions, which eliminates the directional bias with the previous two configurations. Figure 10 shows the Error vs K plot:

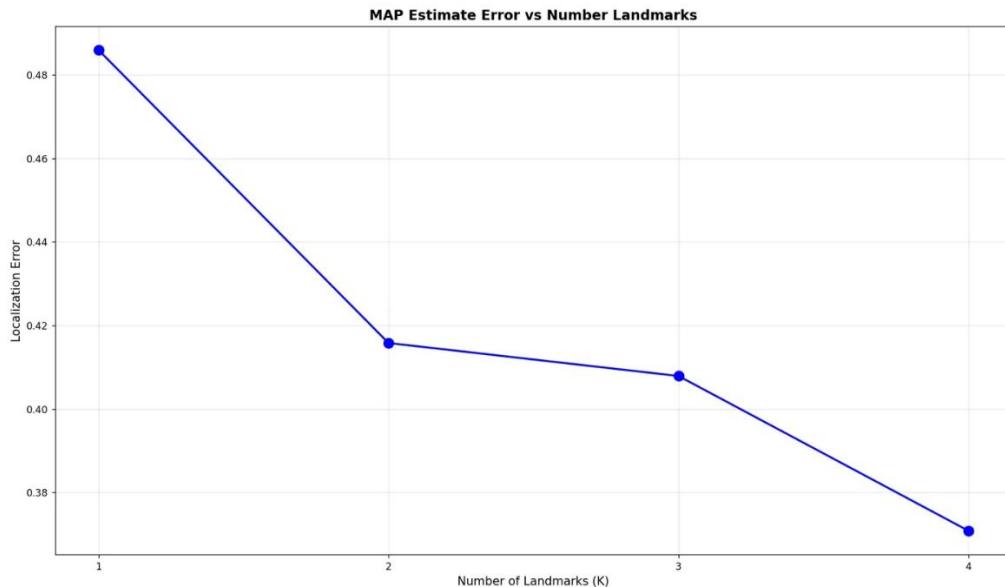


Figure 10: Estimate Error vs Landmarks

The plot of localization error vs landmark numbers show a clear downward trend. The largest improvement came from adding a second landmark, allowing basic triangulation. Subsequent landmarks provide diminishing returns, which is a typical pattern in sensor fusion.

Discussion

With one landmark, the system is poorly observable – a single range defines a circle of possible positions. The prior helps but can't overcome measurement noise. Adding landmarks improve observability: two ranges typically define two intersection points, and further landmarks add redundancy that averages out noise. By K=4, error drops to 0.371, a large improvement.

The interplay between measurements and prior is evident in the results. With K = 1, the MAP estimate stays near the prior mean, still far from the true location. As K increases, measurement increasingly dominate: with K = 4, the estimate of [-0.03, 0.23] approaches the true location as sensor info outweighs prior belief.

The contour plots reveal uncertainty structure. Tight circular contours indicate high certainty in all directions. Elongated elliptical contours show directional uncertainty – for K = 2 with horizontal landmarks, vertical elongation indicates less certainty about the y-coordinate. For K=4 with symmetric placement, more circular contours show balanced uncertainty.

Landmark geometry matters for localization performance. The symmetric placement provided balanced observability in all directions. Clustering landmarks would give redundant information from similar perspectives, worsening performance despite multiple measurements.

These principles extend to real applications. GPS localization, SLAM, and autonomous vehicle sensor fusion all use similar Bayesian frameworks combining measurements with prior beliefs.

Question 4:

In many classification applications, forcing a decision on every input can be costly when the classifier is uncertain. Fields such as medical diagnosis, autonomous driving, and quality control systems benefit from the ability to defer uncertain cases to human experts rather than risk misclassification. This problem formalizes classification with a rejection option through a loss function framework.

We consider a c-class problem where the classifier can choose one of c classes or reject. The loss structure assigns 0 cost to correct classification, λ_s to substitution errors, and λ_r to rejection. The goal is to derive the minimum-risk decision rule that determines when to classify versus when to reject based on the posterior probabilities and cost ratio. The math can be seen below:

Conditional Risk Classification

For deciding class i

$$\begin{aligned}
 R(a_i | x) &= \sum_j \lambda_j a_{ij} P(w_j | x) \\
 &= 0 \cdot P(w_i | x) + \sum_{j \neq i} \lambda_j P(w_j | x) \\
 &= \lambda_i [1 - P(w_i | x)]
 \end{aligned}$$

For Rejection

$$R(a_{\text{ref}} | x) = \sum_j \lambda_j P(w_j | x) = \lambda_T$$

Minimum Risk Decision

Choose class i if

- $R(a_i | x) < R(a_j | x)$ for all $j \neq i$
- $R(a_i | x) < R(a_{\text{ref}} | x)$

Condition 1: From $R(a_i | x) < R(a_j | x)$

$$\lambda_i [1 - P(w_i | x)] < \lambda_j [1 - P(w_j | x)]$$

$$\Rightarrow P(w_i | x) > P(w_j | x) \text{ for all } j \neq i$$

Figure 11: Question 4 Page 1

Condition 2: From $R(a_i|x) \leq R(a_{i+1}|x)$

$$\lambda_s [1 - P(w_i|x)] \leq \lambda_r$$

$$\lambda_s - \lambda_s P(w_i|x) \leq \lambda_r$$

$$\lambda_s P(w_i|x) \geq \lambda_s - \lambda_r$$

$$\Rightarrow P(w_i|x) \geq 1 - \lambda_r / \lambda_s$$

Decision Rule

Decide Class i if

1. $P(w_i|x) \geq P(w_j|x)$ for all j (i has max posterior)
2. $P(w_i|x) \geq 1 - \lambda_r / \lambda_s$ (posterior exceeds threshold)
Reject otherwise (when max posterior < threshold)

Special Cases

Case 1: $\lambda_r = 0$ (Rejection is free)	Case 2: $\lambda_r > \lambda_s$ (Rejection costs more than error)
Threshold: $P(w_i x) \geq 1 - 0 / \lambda_s = 1$	Threshold: $P(w_i x) \geq 1 - \lambda_r / \lambda_s < 0$
Only classify if $P(w_i x) = 1$, else reject	Since probabilities are always non-negative, this condition always satisfied
	Result: Never reject

Figure 12: Question 4 Page 2

The derived decision rule provides a principled framework for handling uncertainty: classify when the maximum posterior probability exceeds the threshold $1 - \lambda_r / \lambda_s$, otherwise reject. This threshold adapts to the cost structure – when rejection is cheap relative to errors (small cost ratio), the threshold is high and the system rejects conservatively. When rejection is expensive, the threshold is low and the system rarely abstains.

The special cases confirm intuition: free rejection leads to rejecting everything except certainties, while expensive rejection eliminates that option entirely. This framework allows systems to provide honest uncertainty quantification rather than forcing incorrect decisions.

Question 5:

The categorical distribution models discrete random variables with K possible outcomes, generalizing the Bernoulli distribution beyond binary cases. It's fundamental to applications like

language modeling, Naïve Bayes classification, and multinomial sampling. This problem derives ML and MAP estimators for the categorical distribution parameters.

I use a 1-of-K encoding where $\mathbf{z} = [z_1, \dots, z_k]^T$ with only one component equal to 1. The parameters $\Theta = [\theta_1, \dots, \theta_k]^T$ represent probabilities that must be non-negative and sum to one. The ML estimator was derived using direct likelihood maximization, then incorporate a Dirichlet prior to obtain the MAP estimator, providing Bayesian smoothing through pseudo-counts. The math can be seen below:

<h3>Maximum Likelihood Estimator</h3> <p>Likelihood Functions</p> <p>Single: $P(\mathbf{z} \Theta) = \prod_{k=1}^K \theta_k^{z_k}$ Dataset: $P(\mathbf{D} \Theta) = \prod_{n=1}^N \prod_{k=1}^K \theta_k^{z_{nk}}$</p> <p>Log-Likelihood</p> $\log P(\mathbf{D} \Theta) = \sum_{n=1}^N \sum_{k=1}^K z_{nk} \log \theta_k = \sum_{k=1}^K N_k \log \theta_k$ <p>$N_k = \sum_{n=1}^N z_{nk}$ count samples in state k</p> <p>Optimization with Constraint</p> <p>Constraint: $\sum_{k=1}^K \theta_k = 1$</p> $\mathcal{L} = \sum_{k=1}^K N_k \log \theta_k + \lambda \left(1 - \sum_{k=1}^K \theta_k \right)$ <p>Dual objective</p> $\frac{N_k}{\theta_k} - \lambda = 0 \quad \theta_k = \frac{N_k}{\lambda}$ <p>Use constraint: $\sum_k \theta_k = 1$</p> $\sum_{k=1}^K \frac{N_k}{\lambda} = 1 \Rightarrow \lambda = \sum_{k=1}^K N_k = N$ <p>ML Estimator: $\hat{\theta}_k^{ML} = \frac{N_k}{N}$</p>
--

Figure 13: Question 5 Page 1

Maximum A Posteriori Estimator

Posterior Distribution

$$p(\theta | D, \alpha) \propto p(D|\theta) \cdot p(\theta | \alpha)$$

$$\propto \left[\prod_{n=1}^N \theta_n^{N_n} \right] \cdot \left[\prod_{n=1}^N \theta_n^{d_{n-1}} \right]$$

$$= \prod_{n=1}^N \theta_n^{N_n + d_{n-1}}$$

Log Posterior

$$\log p(\theta | D, \alpha) = \sum_{n=1}^N (N_n + d_{n-1}) \log \theta_n + \text{const}$$

Optimization with constraint

Lagrangian

$$\mathcal{L} = \sum_{n=1}^N (N_n + d_{n-1}) \log \theta_n + \lambda \left(1 - \sum_{n=1}^N \theta_n \right)$$

Derivative:

$$\frac{N_n + d_{n-1}}{\theta_n} - \lambda = 0$$

$$\theta_n = \frac{N_n + d_{n-1}}{\lambda}$$

Using constraint

$$\lambda = \sum_{n=1}^N (N_n + d_{n-1}) = N + \sum_{n=1}^N d_n - K$$

Result

$$\theta_n^{\text{MAP}} = \frac{N_n + d_{n-1}}{N + d_n - K}$$

Figure 14: Question 5 Page 2

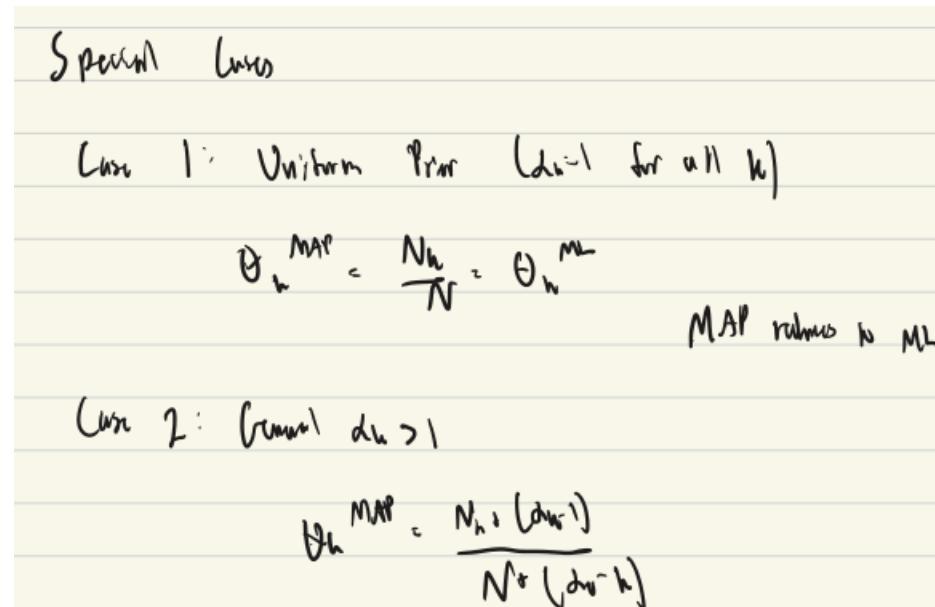


Figure 15: Question 5 Page 3

The ML estimator $\hat{\theta}_k^{(ML)} = N_k/N$ is the empirical frequency, while intuitive and unbiased, also assigns 0 probability to unobserved states. This problem is severe in high dimensional settings where many states may be absent from finite training data.

The MAP estimator $\hat{\theta}_k^{(MAP)} = (N_k + \alpha_k - 1)/(N + \alpha_0 - K)$ addresses this by adding pseudo counts from the Dirichlet prior. With uniform prior, MAP reduces to ML. With $\alpha_k > 1$, every state receives positive probability even when unobserved, providing essential smoothing for generalization.

As sample size grows, MAP converges to ML since data overwhelms the prior. For finite samples, the prior provides crucial regularization. This framework is computationally efficient and foundational to Bayesian models like topic modeling and language modeling where handling unseen events is critical.

Citations

- Lecture Notes
- Code Folder
- Repo Link: <https://github.com/ishaandesai0/EECE-5644/tree/main/Assignment%202>
- Claude AI – Mathematical Concepts
- Duda, R.O., Hart, P.E., and Stork, D.G. Pattern Classification, 2nd Edition