

EECE 5644 Assignment 4

Ishaan Desai (desai.is@northeastern.edu)

December 1st, 2025

Professor Erdogmus

GitHub Link:

<https://github.com/ishaandesai0/EECE-5644/tree/main/Assignment%204>

Question 1

This problem had me train and test a Support Vector Machine and Multi-Layer Perceptron classifiers on a synthetic dataset consisting of two overlapping concentric circles with Gaussian noise. The goal here was to minimize the probability of classification error using appropriate hyperparameter selection.

The dataset was generated using the following model:

$$x = r[\cos(\theta) \sin(\theta)] + n$$

Where $\theta \sim \text{Uniform}[-\pi, \pi]$ and $\mathbf{n} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$.

The parameters were as follows:

- $r-1 = 2$ (inner circle, class -1)
- $r+1 = 4$ (outer circle, class +1)
- $\sigma = 1$ (noise standard deviation)

The dataset sizes were as follows:

- Training Set: 1000 samples (500 per class)
- Test Set: 10,000 samples (5,000 per class)

Figures 1 and 2 show the generated training and testing data. The visualizations show two concentric circles with a lot of overlap due to noise. This also shows that the optimal decision boundary should be circular around radius $r = 3$. Figures 1 and 2 can be seen below:

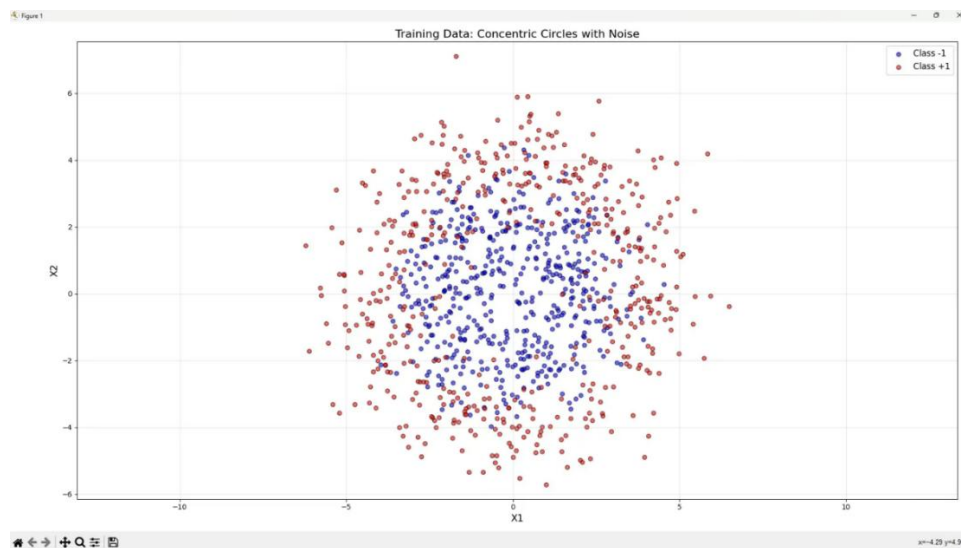


Figure 1: Training Data

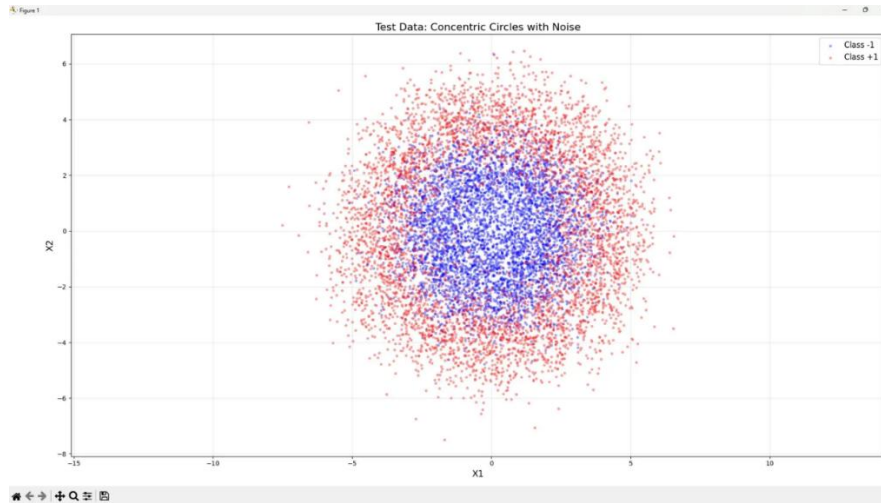


Figure 2: Test Data

The graphs show the inner blue circle (class -1) centered at radius ~ 2 and the outer red circle (class +1) centered at radius ~ 4 , with significant overlap in the region between the circles as a result of noise.

Support Vector Machine

An SVM with a Gaussian kernel was trained to classify the data. The RBF kernel is defined as

$$K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}$$

The RBF kernel operates through the kernel trick, implicitly computing inner products in an infinite-dimensional feature space without explicitly constructing the feature map. For radially symmetric data like concentric circles, the RBF kernel is theoretically optimal because it treats all points equidistant from a center equally, naturally capturing circular decision boundaries. The decision boundary has the form

$$\sum_i \alpha_i y_i e^{-\gamma \|x_i - x_j\|^2} + b = 0$$

This depends on distance only and exhibits rotational invariance. The parameter γ controls kernel width: small γ creates smoother boundaries while a larger value creates more localized and complex boundaries.

The SVM training solves:

$$\min \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \xi_i$$

The first term regularizes the boundary, while the second term penalizes misclassifications. The box constraint C balances these objectives.

The SVM required tuning two important hyperparameters: gamma, which controls the kernel width, and C, the box constraint controlling regularization strength. For gamma, I tested eight values spanning multiple orders of magnitude: 0.01, 0.05, 0.1, 0.5, 1, 2, 5, and 10. Smaller gamma values make smoother decision boundaries by considering distant points as similar, while larger values make more complex and localized boundaries. For C, I tested five values, 0.1, 1, 10, 100, and 1000. Larger C values penalize margin violations more heavily, potentially leading to overfitting, while smaller values allow more misclassifications during training for better generalization. I also performed 5-fold cross-validation across all 40 combinations (8 gammas * 5 C values), training SVM on 4 folds and evaluating on the remaining fold for each combination, then averaging validation accuracy across all five folds to identify optimal hyperparameters. Figure 3 below shows the cross-validation heatmap:

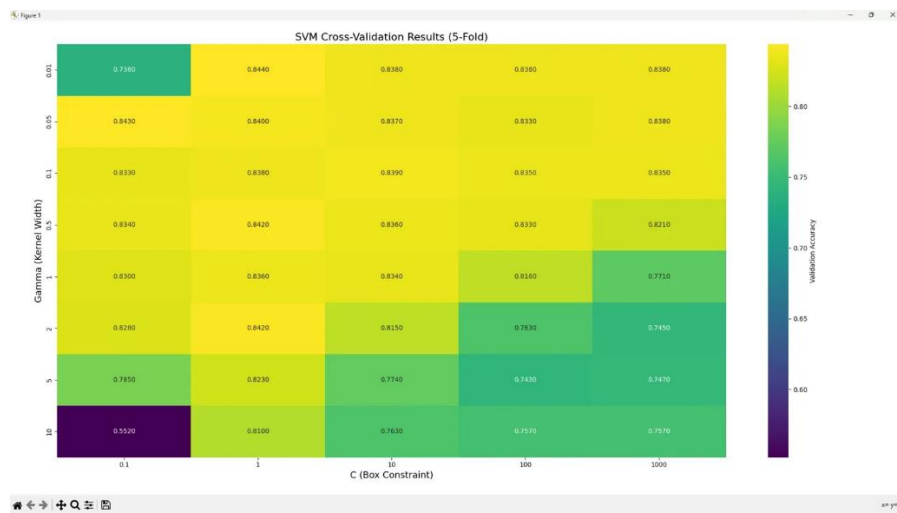


Figure 3: Cross Validation Heatmap

The heatmap revealed clear patterns in hyperparameter performance. The optimal region appeared in the top-left portion of the heatmap, corresponding to small gamma values paired with moderate C values. The best combination was gamma = 0.01 and C = 1, with validation accuracy of 84.4% and error rate of 15.6%. Larger gamma values caused overfitting particularly when paired with small C values. This occurs because large gamma creates highly localized kernels that memorize training data instead of learning generalizable patterns. Very large C values showed minimal improvement over the lower ones, showing diminishing returns.

Using optimal hyperparameters, the final SVM model was trained on the full training set and evaluated on the 10k sample test set. The model achieved a test accuracy of 83.12% with an error of 16.88%, correctly classifying 8312/10,000 samples. The small decrease from CV to test accuracy was expected and showed good generalization without significant overfitting. This small gap validated the effectiveness of cross-validation in selecting hyperparameters generalizing well to unseen data. Figure 4 below shows the decision boundary visualization:

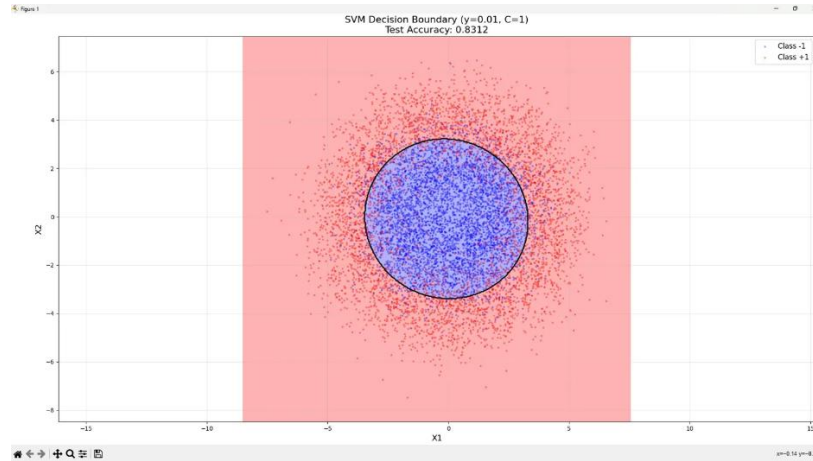


Figure 4: SVM Decision Boundary

The SVM successfully learned a smooth, nearly circular decision boundary that separates the inner blue region (Class -1) from the outer red (Class +1). The black contour line represents the decision boundary where the classifier is maximally uncertain. The smooth circular shape demonstrates the RBF kernel's natural ability to capture radially symmetric pattern.

Multi-Layer Perceptron Classification

An MLP with one hidden layer computes the following function

$$h = \sigma_1(W^{(1)}x + b^{(1)})$$

$$y^{\wedge} = \text{softmax}(W^{(2)}h + b^{(2)})$$

Where $W^{(1)} \in \mathbb{R}^{P \times 2}$ are input-to-hidden weights, $b^{(1)} \in \mathbb{R}^P$ are hidden biases, σ_1 is the hidden layer activation function, $W^{(2)} \in \mathbb{R}^{2 \times P}$ are hidden-to-output weights, $b^{(2)} \in \mathbb{R}^2$ are output biases, and P is the number of hidden neurons.

The softmax function for class k is:

$$P(y = k|x) = \frac{e^{w_k^T h + b_k}}{\sum_{j=1}^2 e^{w_j^T h + b_j}}$$

The MLP is trained by minimizing the cross-entropy loss, which is equal to maximum likelihood estimation:

$$L = - \sum_{i=1}^N \sum_{k=1}^2 y_{ik} \log(y_{ik})$$

Where y_{ik} is 1 if sample i belongs to class k and 0 otherwise, and y^{\wedge}_{ik} is the predicted probability for class k .

The MLP architecture consisted of three layers: an input layer with 2 neurons corresponding to the 2D feature space, a hidden layer with P neurons using a specified activation function, and an output layer with 2 neurons using softmax activation for binary classification. Two hyperparameters were tuned through cross validation: the hidden layer size, testing values of 3, 5, 10, 15, 20, 30, and 50 neurons, and the activation function, testing tanh, ReLU, and logistic functions. The training configuration used the Adam optimizer with a default learning rate of 0.001, a max of 1000 iterations, and early stopping enabled with a 10% validation split to prevent overfitting. 5-fold cross-validation was performed across all 21 combinations, creating a comprehensive search over the hyperparameter space. Figure 5 below shows the cross-validation results:



Figure 5: MLP Cross-Validation Results

Figure 5 showed clear performance patterns across different architectures. ReLU activation consistently outperformed tanh and logistic across most sizes, achieving the highest validation accuracies. Performance improved with network size up to 30 neurons, reaching peak CV accuracy of 76.5%, but 50 neurons showed similar performance to 30, suggesting diminishing returns from additional model capacity. Logistic activation performed poorly across all network sizes with accuracy below 60%, indicating that it's unsuitable for this problem. Small networks with 3-5 neurons underfitted regardless of activation function, unable to capture the complexity of the circular decision boundary. The optimal configuration was 30 hidden neurons with ReLU activation, achieving a CV accuracy of 76.5% and CV error rate of 23.5%.

The final MLP model trained with optimal hyperparameters (30 neurons, ReLU) achieved a test accuracy of 79.25% with error rate of 20.75%, correctly classifying 7,925 out of 10,000 samples. The test accuracy exceeded the CV accuracy by 2.75%, suggesting that the model generalized well and the CV estimate was a bit conservative. This can occur due to random variation in fold

splits or the model benefiting from training on the full dataset after CV selection. Figure 6 below shows the decision boundary:

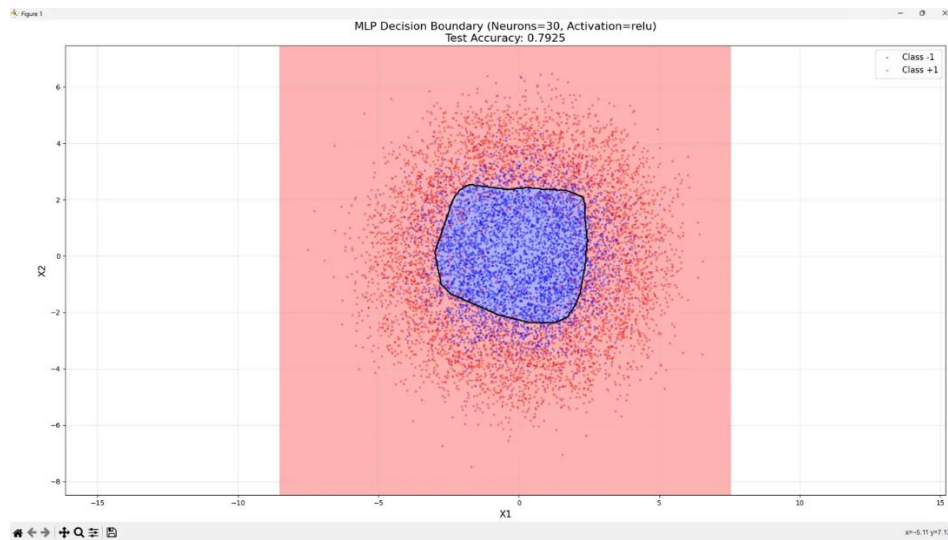


Figure 6: MLP decision boundary

The MLP learned a piecewise-linear decision boundary that approximates the circular shape. The boundary exhibits a polygonal appearance characteristic of ReLU networks, which construct decision regions through combinations of hyperplanes. While the boundary was able to successfully separate most points, it lacks the smoothness of the SVM's circular boundary.

Figure 7 below shows both decision boundaries side by side for comparison:

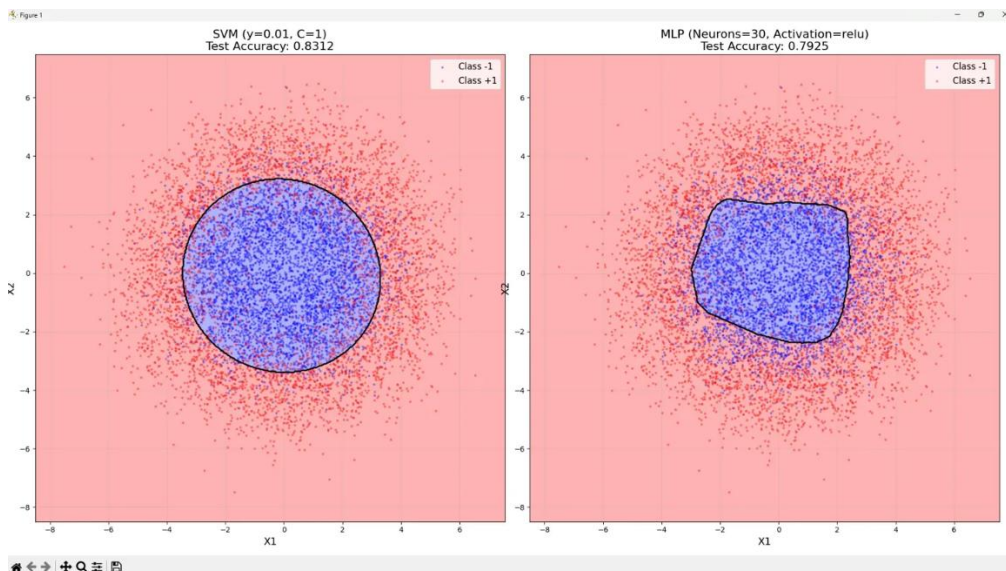


Figure 7: Comparison of Decision Boundaries

Table 1 also provides numerical comparisons:

Metric	SVM	MLP
CV Accuracy	84.40%	76.50%
CV Error	15.60%	23.50%
Test Accuracy	83.12%	79.25%
Test Error	16.88%	20.75%
Correct Predictions	8,312/10,000	7,925/10,000
Hyperparameters	Gamma=0.01, C=1	Neurons=30, ReLU

Table 1: SVM vs MLP Performance Comparison

Figure 8 below also visualizes this comparison:

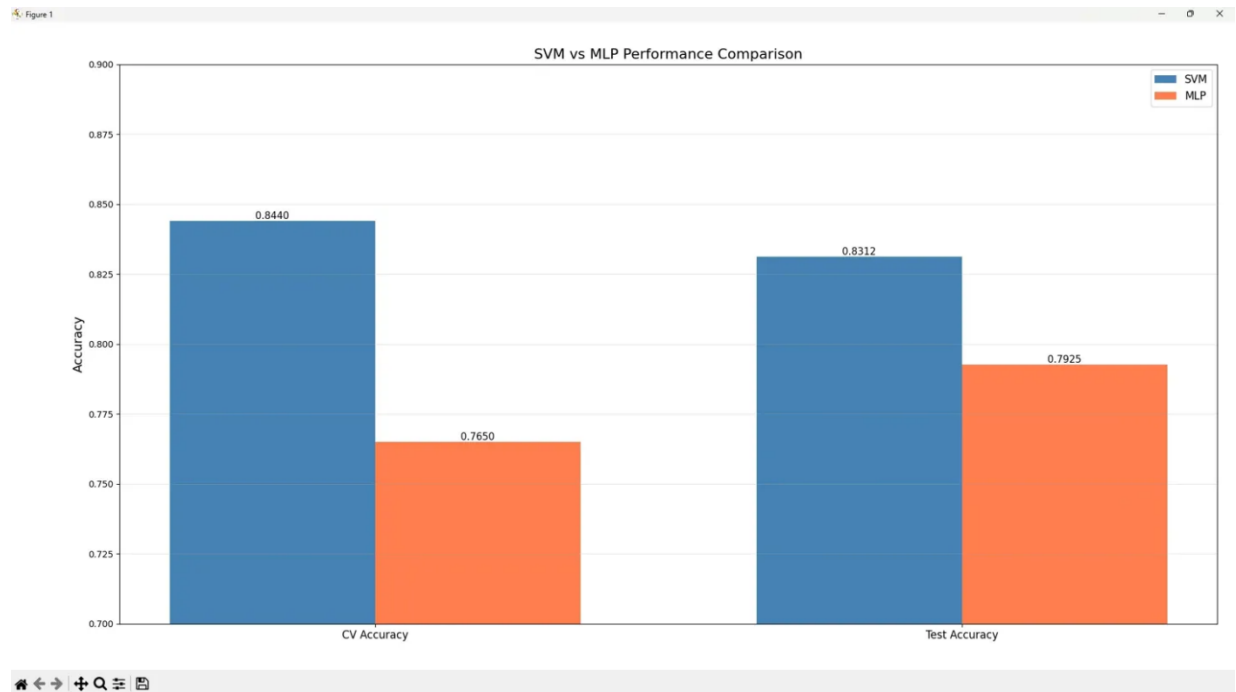


Figure 8: Performance Comparison Bar Chart

SVM outperformed MLP by 3.87% on the test set. Both models showed consistent performance between CV and test sets, showing good generalization. The SVM's smooth circular boundary more closely matches the theoretical optimal boundary, while the MLP's piecewise-linear approximation is less geometrically efficient.

The performance gaps come from the problem structure. With the optimal decision boundary being a circle, we have a quadratic curve ($x_1^2 + x_2^2 = r^2$). The RBF kernel implicitly maps data to an infinite dimensional space where this circular boundary becomes linear, while MLP must approximate the circle using piecewise-linear segments from ReLU activations. Even though 30 neurons were used, the MLP underperformed since standard activation functions aren't quadratic, forcing the network to approximate the curve with linear pieces. From a computational perspective, the SVM only needs 2 hyperparameters compared to the MLP's architecture and

activation function combinations, though SVM kernel computations can be expensive for large datasets while MLP inference is faster after training usually.

To contextualize these results, we can estimate the theoretical Bayes error rate given the noise level and class overlap. With $\sigma = 1$ and circles at radii 2 and 4, the optimal decision boundary at radius $r = 3$ still experiences significant overlap from both classes. The Bayes error rate is approximately 15-17%. Both the SVM and MLP perform reasonably close to this theoretical limit, with the SVM nearly achieving optimal performance.

Question 2

This question involved using Gaussian Mixture Models with the Expectation-Maximization algorithm to segment a colored image. The task required creating 5-dimensional feature vectors for each pixel, performing K-fold cross-validation to select the optimal number of GMM components and visualizing the final segmentation.

A GMM represents the probability density as a weighted sum of Gaussian components:

$$p(x) = \sum_{k=1}^K \alpha_k N(x|\mu_k, \Sigma_k)$$

Where K is the number of components, α_k are mixture weights satisfying $\sum_{k=1}^K \alpha_k = 1$, μ_k are component means, and Σ_k are covariance matrices.

The EM algorithm estimates these parameters by iterating between two steps. The E-step computes responsibilities for each data point to each component:

$$\gamma_{ik} = \frac{\alpha_k N(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^K \alpha_j N(x_i|\mu_j, \Sigma_j)}$$

The M-step updates parameters and these steps repeat until convergence, maximizing the log-likelihood:

$$\log p(X|\theta) = \sum_{i=1}^N \log \left[\sum_{k=1}^K \alpha_k N(x_i|\mu_k, \Sigma_k) \right]$$

I selected image #167083 from the Segmentation Dataset, which shows a mountain lake scene with distinct visual regions including sky, snow-covered mountains, rocky surfaces, water, and a cliff face. The image was 481 X 321, giving 154,401 pixels to process. Figure 9 shows the original image:

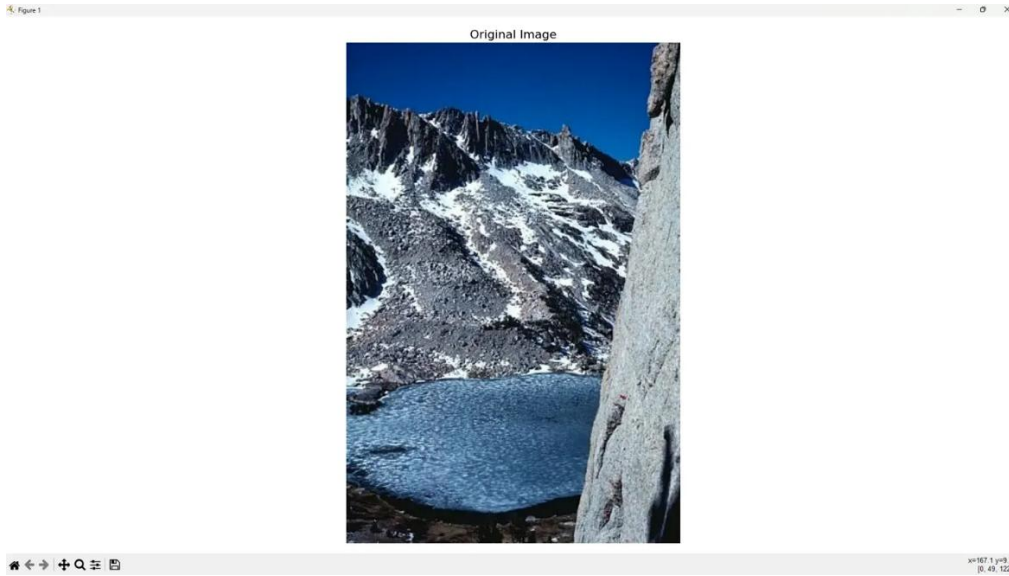


Figure 9: Original Image

Each pixel at position (i, j) with RGB values (R, G, B) was transformed into a 5D normalized feature vector:

$$[i/(H - 1)]$$

$$[j/(W - 1)]$$

$$f=[R/255]$$

$$[G/255]$$

$$[B/255]$$

Where $H = 481$ and $W = 321$ (image height and width). This normalization ensures all features lie in $[0, 1]$, placing feature vectors in the 5D unit hypercube. The spatial features encourage nearby pixels to cluster together, while color features group pixels with similar appearance. The joint 5D space allows GMM to find segments that are both spatially contiguous and visually similar.

The resulting feature matrix had shape $154,401 \times 5$ with values ranging from 0 to 1. The first five feature vectors showed pixels from the top-left corner with similar dark blue colors and incrementing column positions, validating the feature extraction process.

The key challenge in GMM-based segmentation is selecting the optimal number of components K . Too few components under-segment the image, merging distinct regions, while too many components over-segment, fragmenting uniform regions. I used 5-fold cross-validation with validation log-likelihood as the selection criterion.

I tested $K \in \{2, 3, 4, 5, 6, 7, 8, 9, 10\}$ components. For each value, the features were randomly portioned into 5 folds. The GMM was trained on 4 folds (123,521 samples) using the EM algorithm, then the log-likelihood was computed on the validation fold (30,880 samples). This process repeated for all 5 folds, and the average validation log-likelihood was recorded. The GMM training used full covariance matrices to allow arbitrary ellipsoidal components, a maximum of 100 EM iterations, and random seed 43 for reproducibility.

Table 2 below shows the validation log-likelihood for each model order:

Components (K)	Avg Validation Log-Likelihood
2	4.4533
3	4.9962
4	5.1889
5	5.5293
6	5.6978
7	5.7843
8	5.9352
9	6.0002
10	6.0418

Table 2: GMM Model Order Selection Results

Figure 10 below also visualizes this:

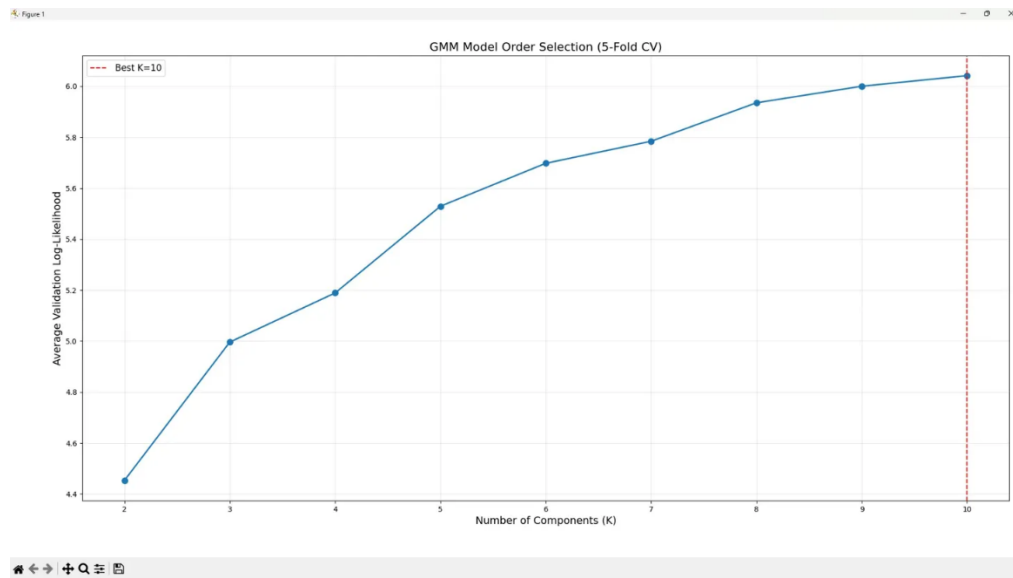


Figure 10: GMM Model Order Selection

The validation log-likelihood increased with K , with the rate of improvement starting to decrease as K grows. Moving from $K = 2$ to $K = 3$ improved log-likelihood by 0.543, while moving from $K = 9$ to $K = 10$ improved by 0.042. These diminishing returns suggest that the optimal complexity was being approached. I chose $K = 10$ as optimal.

Once picking $K = 10$, the final GMM was trained on all 154,401 pixel features using the EM algorithm. Each pixel was assigned to its most likely component by computing posterior probabilities:

$$label_i = \operatorname{argmax} P(f|f_i) = \operatorname{argmax} \frac{\alpha_k N(f_i|\mu_k, \Sigma k)}{\sum_{j=1}^{10} \alpha_k N(f_i|\mu_j, \Sigma j)}$$

All 10 components were used in the final segmentation, with each component assigned to at least some pixels, indicating no redundant components. The labels were reshaped back to the image dimensions and normalized to $[0, 255]$ for visualization with good contrast. Figures 11 and 12 below show the segmentation results:

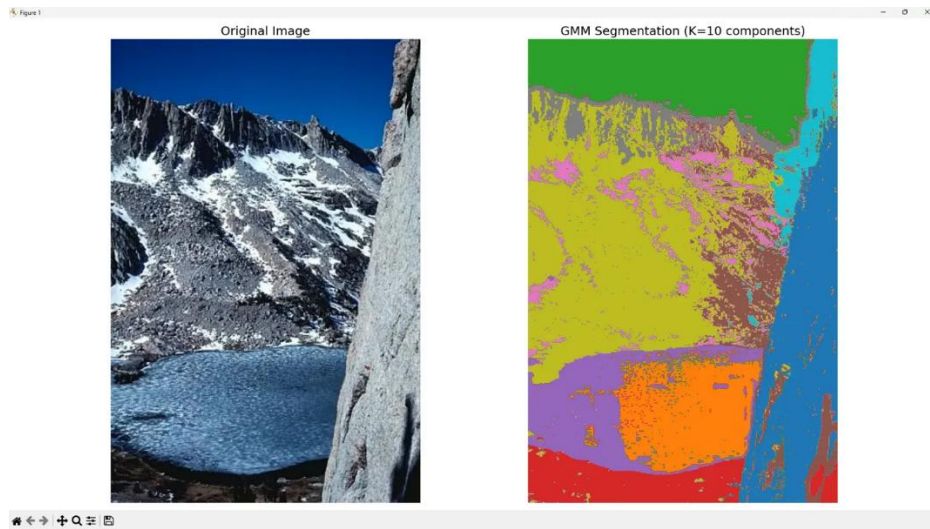


Figure 11: Side by Side Comparison

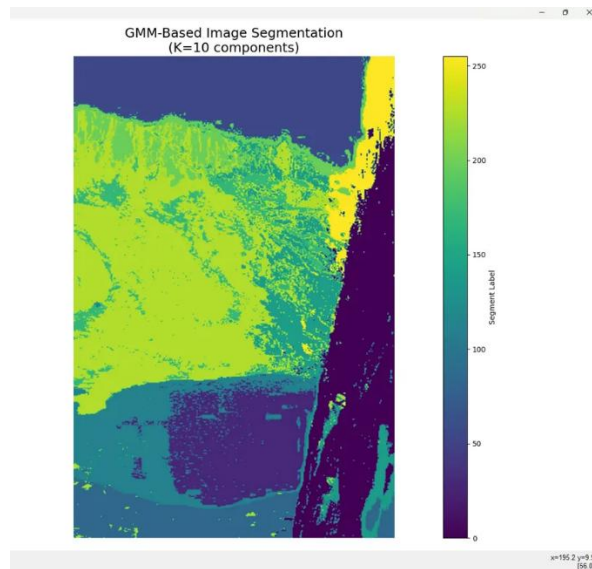


Figure 12: GMM Segmentation with Viridis Colormap

The 10 components successfully identified semantically meaningful regions. The sky region at the top was captured by components representing blue hues. Snow and ice patches on the mountains were assigned to components for white and bright regions in yellow and pink. The rocky mountain surfaces with gray tones were split across multiple components, capturing texture variations. Water at the bottom received components for blue-purple lake tones. The cliff face on the right was assigned to components capturing the lighter gray rock wall. Additional components handled transition zones where colors and textures mix at region boundaries.

The segmentation demonstrated both strengths and limitations. Major regions like the sky, water, snow, and rock were clearly distinguished. Spatial coherence was maintained through the row and column features, preventing scattered fragments. Color similarity was also well captured through RGB features. Some over-segmentation occurred within visually uniform regions, where the sky or rock surfaces were split into multiple components. Boundaries weren't perfectly sharp due to the probabilistic nature of GMM assignments. Some small, isolated segments appeared where local noise or texture created variations.

The GMM based segmentation successfully partitioned the image into semantically meaningful regions without any labeled training data. The inclusion of spatial features alongside color features proved crucial. Using only RGB features would have produced scattered and spatially incoherent segments where all of the blue pixels would cluster together regardless of location. The spatial features encouraged nearby pixels to share labels, maintaining region connectivity.

The selection of $K = 10$ components balanced segmentation quality with computational cost. Fewer components would under-segment, merging distinct regions such as sky and water. More components beyond 10 might over-segment uniform regions excessively, though the diminishing returns in log-likelihood suggest minimal benefit. The cross-validation approach provided a data-driven method for this selection and in turn avoided arbitrary choices.

Citations

- Lecture Notes
- Code Folder
- Repo Link: <https://github.com/ishaandesai0/EECE-5644/tree/main/Assignment%204>
- Claude AI – Mathematical Concepts
- Duda, R.O., Hart, P.E., and Stork, D.G. Pattern Classification, 2nd Edition
- Berkeley Segmentation Dataset:
<https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/>