EECE5644 Assignment 4

Mariona Jaramillo Civill

December 3, 2024

NUID: 002413201 - jaramillocivill.m@northeastern.edu

Problem 1

In this project, we explored the training and evaluation of Support Vector Machine (SVM) and Multi-layer Perceptron (MLP) classifiers, with the primary objective of minimizing the probability of classification error using 0-1 loss, where all classification errors are treated equally. The data for classification were generated as samples for classes $l \in \{-1, +1\}$ using the following equation:

$$x = r_l \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} + n,$$

where θ follows a uniform distribution over $[-\pi, \pi]$, and n is Gaussian noise with $\sigma^2 I$ where $\sigma = 1$. The radii for the classes are $r_{-1} = 2$ and $r_{+1} = 4$, resulting in two overlapping concentric disks, as shown in Figure 1. This configuration presents a challenging classification task due to its angular symmetry.

The SVM model employs a Gaussian (radial-basis function) kernel to map the input data to a higher-dimensional space. The key hyperparameters include the regularization parameter C, which balances the trade-off between maximizing the margin and minimizing classification errors, and the kernel width γ, which controls the smoothness of the decision boundary. These hyperparameters were selected using 10-fold cross-validation to ensure optimal generalization.

The MLP model was configured with a single hidden layer. The primary hyperparameter examined was the number of perceptrons in the hidden layer, as this impacts the complexity and capacity of the model; and also the activation function (relu, tanh and logistic). As with the SVM, hyperparameters for the MLP were selected through 10-fold cross-validation.

The 10-fold cross-validation method divides the training dataset into ten subsets, using nine for training and one for validation, iterating until each subset has served as the validation set. The average validation performance across folds was used to determine the best hyperparameters.

The best hyperparameters found for SVM were: $\{'C' : 1, 'gamma' : 0.01\}$ and the best for MLP were $\{'activation' :' relu', 'hidden\_layer\_sizes' : (100,)\}$

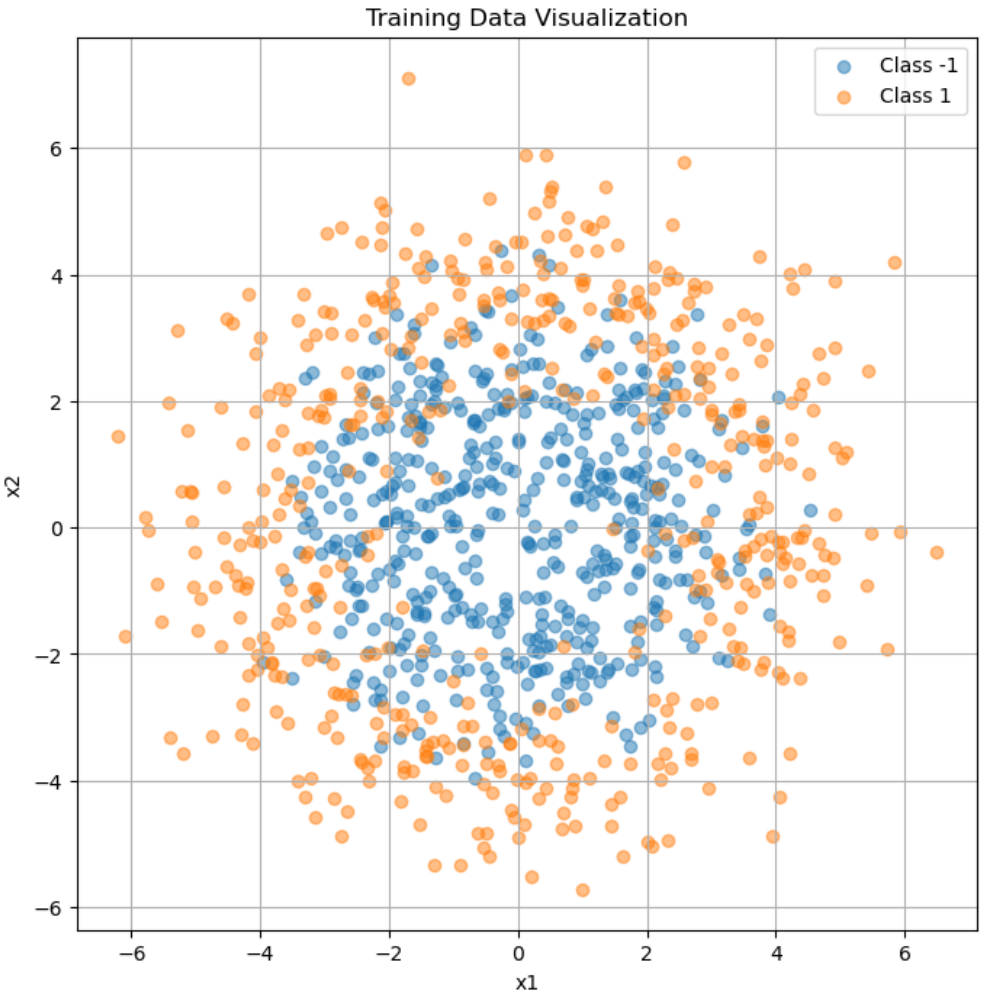

Figure 1: Training data visualization

After identifying the best hyperparameters, the final SVM and MLP models were trained on the entire training dataset (1,000 samples) and applied to the test dataset (10,000 samples) to estimate the probability of classification error. A test accuracy of 82.36% was achieved on SVM and 82.81% on MLP. The decision boundaries indicating the classification error for both models have been plotted for SVM and MLP in Figure 2 and Figure 3 respectively.

Both models successfully captured the classification boundary between the two data classes, demonstrating effective training and generalization strategies. The 10-fold cross-validation process was crucial in selecting hyperparameters that minimized classification errors, ensuring robust performance on the test dataset.

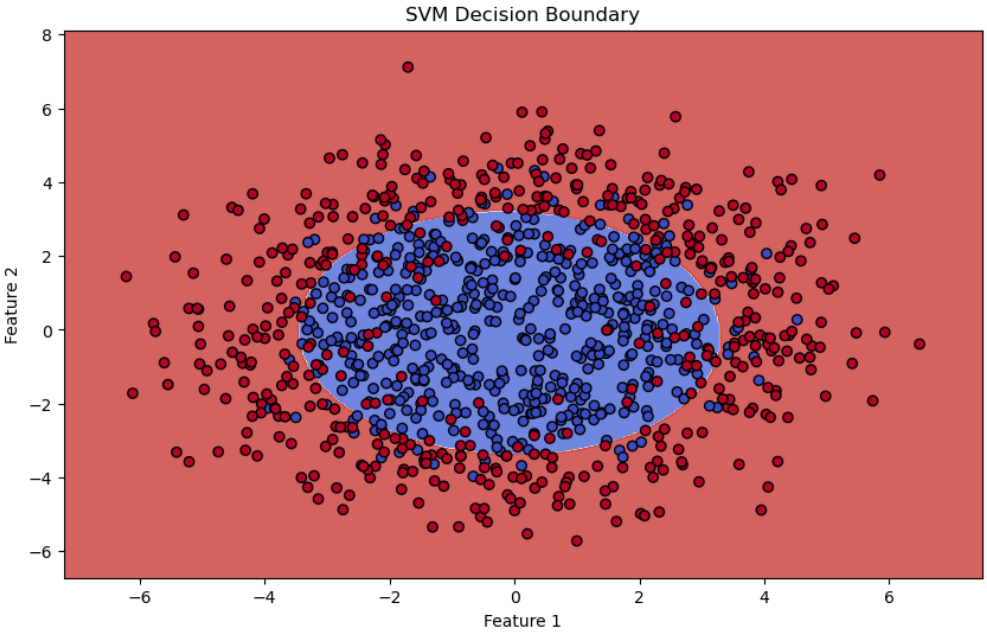

Figure 2: SVM decision boundary

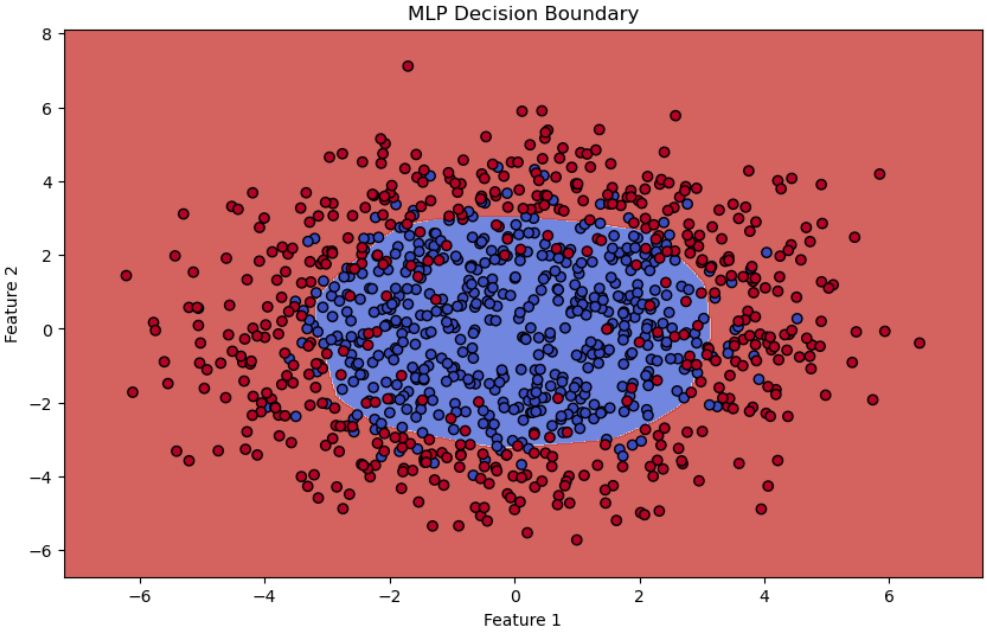

Figure 3: MLP decision boundary

Problem 2

In this assignment, we focused on segmenting a color image using Gaussian Mixture Model (GMM)-based clustering. The image was selected from a publicly available dataset, and each pixel was processed to form a 5-dimensional feature vector. This vector consisted of the row index, column index, and the red, green, and blue (RGB) values of the pixel. Each entry of the feature vector was normalized individually to fit within the interval [0,1], ensuring uniform scaling across all dimensions and allowing the data to occupy the 5-dimensional unit hypercube.

To determine the optimal GMM, we employed maximum likelihood parameter estimation combined with 10-fold cross-validation. This process involved dividing the dataset into 10 folds, training the model on nine folds, and validating on the remaining one, with each fold serving as the validation set exactly once. The average validation log-likelihood was calculated for different model configurations, and the GMM with the highest average validation score was selected as the best model. The best model was the one using 7 Gaussian components.

After fitting the final GMM using the entire training dataset, we assigned each pixel a component label based on the maximum a posteriori (MAP) probability. This label assignment was done by evaluating the posterior probabilities for each pixel's feature vector and selecting the component with the highest probability.

For visual assessment, we mapped the assigned component labels to grayscale values, ensuring a uniform distribution to maintain contrast across the segmented image Figure 4. The results highlighted the ability of the GMM to capture the underlying distribution of pixel features and segment the image into distinct regions, demonstrating the effectiveness of clustering approaches in image segmentation tasks.

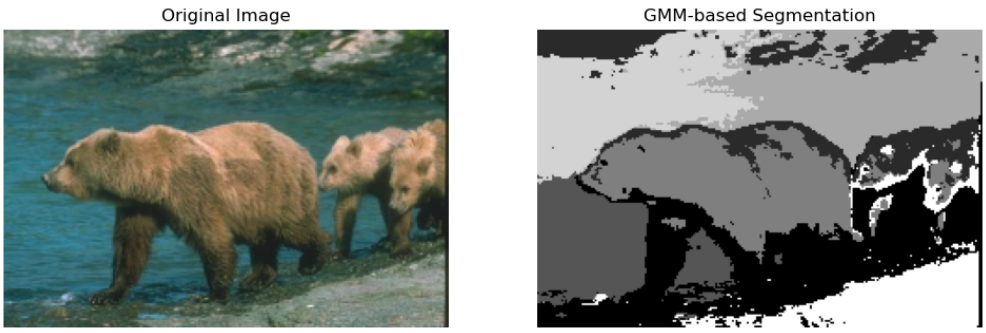

Figure 4: GMM based segmentation

Appendix: Code

Listing 1: Question 1

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import cross_val_score, GridSearchCV, train_test_split
from sklearn.metrics import accuracy_score

# Set seed for reproducibility
np.random.seed(42)

# Generate data function
def generate_data(num_samples, r, label):
    theta = np.random.uniform(-np.pi, np.pi, num_samples)
    noise = np.random.normal(0, 1, (num_samples, 2))
    x = r * np.column_stack((np.cos(theta), np.sin(theta))) + noise
    y = np.full(num_samples, label)
    return x, y

# Function to plot decision boundaries
def plot_decision_boundary(model, X, y, title):
    # Create a mesh grid
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 500), np.linspace(y_min, y_max, 500))

    # Predict on the mesh grid
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Plot the contour and training examples
    plt.figure(figsize=(10, 6))
    plt.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.coolwarm)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k', cmap=plt.cm.coolwarm)
    plt.title(title)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.show()

# Generate training data (1000 samples)
x_train_neg, y_train_neg = generate_data(500, 2, -1)
x_train_pos, y_train_pos = generate_data(500, 4, 1)
x_train = np.vstack((x_train_neg, x_train_pos))
y_train = np.hstack((y_train_neg, y_train_pos))


# Generate test data (10000 samples)
x_test_neg, y_test_neg = generate_data(5000, 2, -1)
x_test_pos, y_test_pos = generate_data(5000, 4, 1)
x_test = np.vstack((x_test_neg, x_test_pos))
y_test = np.hstack((y_test_neg, y_test_pos))

# Visualize the generated data
plt.figure(figsize=(8, 8))
plt.scatter(x_train[y_train == -1][:, 0], x_train[y_train == -1][:, 1], label='Class -1',
    alpha=0.5)
plt.scatter(x_train[y_train == 1][:, 0], x_train[y_train == 1][:, 1], label='Class 1',
    alpha=0.5)
plt.title('Training Data Visualization')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()
plt.grid(True)
plt.show()

# SVM hyperparameter tuning using GridSearchCV
param_grid_svm = {
    'C': [0.1, 1, 10, 100],
    'gamma': ['scale', 0.01, 0.1, 1, 10]
}
svm_model = SVC(kernel='rbf')
grid_svm = GridSearchCV(svm_model, param_grid_svm, cv=10)
```

```
grid_svm.fit(x_train, y_train)

# Print the best SVM parameters
print("Best SVM Parameters:", grid_svm.best_params_)

# Train the final SVM model
best_svm = grid_svm.best_estimator_
y_pred_svm = best_svm.predict(x_test)
svm_accuracy = accuracy_score(y_test, y_pred_svm)
print("SVM Test Accuracy:", svm_accuracy)

# MLP hyperparameter tuning using GridSearchCV
param_grid_mlp = {
    'hidden_layer_sizes': [(10,), (50,), (100,)],
    'activation': ['relu', 'tanh', 'logistic']
}
mlp_model = MLPClassifier(max_iter=1000)
grid_mlp = GridSearchCV(mlp_model, param_grid_mlp, cv=10)
grid_mlp.fit(x_train, y_train)

# Print the best MLP parameters
print("Best MLP Parameters:", grid_mlp.best_params_)

# Train the final MLP model
best_mlp = grid_mlp.best_estimator_
y_pred_mlp = best_mlp.predict(x_test)
mlp_accuracy = accuracy_score(y_test, y_pred_mlp)
print("MLP Test Accuracy:", mlp_accuracy)

# Train the final SVM model and visualize
plot_decision_boundary(best_svm, x_train, y_train, title='SVM Decision Boundary')

# Train the final MLP model and visualize
plot_decision_boundary(best_mlp, x_train, y_train, title='MLP Decision Boundary')
```

Listing 2: Question 2

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
from sklearn.model_selection import KFold
import cv2
import urllib.request

# Load the image
image_path = 'image_q2.jpg'
image = cv2.imread(image_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Optional: downsample for computational feasibility
scale_percent = 50  # e.g., downsample to 50% of the original size
width = int(image.shape[1] * scale_percent / 100)
height = int(image.shape[0] * scale_percent / 100)
dim = (width, height)
image_resized = cv2.resize(image, dim, interpolation=cv2.INTER_AREA)

# Extract features: row index, column index, and RGB values
rows, cols, _ = image_resized.shape
features = np.zeros((rows * cols, 5))
for r in range(rows):
    for c in range(cols):
        features[r * cols + c, :] = [r / rows, c / cols,
                                     image_resized[r, c, 0] / 255.0,
                                     image_resized[r, c, 1] / 255.0,
                                     image_resized[r, c, 2] / 255.0]

# Fit GMM with 10-fold cross-validation to select the best number of components
kf = KFold(n_splits=10)
best_gmm = None
best_score = -np.inf
best_n_components = 0
possible_n_components = range(2, 21)  # Range of possible components

for n_components in possible_n_components:
    gmm = GaussianMixture(n_components=n_components, covariance_type='full')
    scores = []
```

```
    for train_idx, val_idx in kf.split(features):
        gmm.fit(features[train_idx])
        score = gmm.score(features[val_idx])
        scores.append(score)

    avg_score = np.mean(scores)
    if avg_score > best_score:
        best_score = avg_score
        best_gmm = gmm
        best_n_components = n_components

print(f"Best number of components: {best_n_components}")

# Assign labels to each pixel
labels = best_gmm.predict(features)
label_image = labels.reshape(rows, cols)

# Normalize labels to grayscale values for visualization
label_image_normalized = (255 * (label_image - np.min(label_image)) /
                          (np.max(label_image) - np.min(label_image))).astype(np.uint8)

# Plot the original and segmented images
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.imshow(image_resized)
plt.title("Original Image")
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(label_image_normalized, cmap='gray')
plt.title("GMM-based Segmentation")
plt.axis('off')

plt.show()
```