Programming Assignment 5 Report

Name: Mayank Dahiya (B23CS1035)

Colab File Links:

• PCA: pa5_B23CS1035_problem1.ipynb

• SVM: pa5_B23CS1035_problem2.ipynb

• ANN: pa5_B23CS1035_problem3.ipynb

Introduction

This report addresses three fundamental machine learning techniques: Principal Component Analysis (PCA), Support Vector Machines (SVM), and Artificial Neural Networks (ANN). The experiments involve visualizing and analyzing dimensionality reduction with PCA, implementing SVM classifiers using different kernels and hyperparameters, and building a flexible feedforward neural network from scratch with support for multiple activation functions and loss calculation.

Problem 1: Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a statistical technique used for dimensionality reduction while preserving as much variance as possible in the dataset. It transforms the original correlated features into a new set of uncorrelated variables called principal components, ordered by the amount of variance they capture from the data. PCA is widely used for visualization, compression, and preprocessing in machine learning tasks.

Task 1(a): Mean of the Dataset

The mean was calculated for each feature across all data points. Centering the data around the origin is a prerequisite for PCA, ensuring that the transformation focuses on variance and not absolute position.

$$\mu_j = \frac{1}{n} \sum_{i=1}^{n} x_{ij}$$

Output: Mean = [5 , 16.983]

Task 1(b): Covariance Matrix

A covariance matrix reveals the linear dependencies between features. Here, the symmetric 2×2 matrix quantified how changes in one variable are associated with changes in another.

$$\Sigma = \frac{1}{n}(X - \mu)^T (X - \mu)$$

Output: Covariance Matrix = **[[8.36 16.6] [16.6 33.96]]**

Task 1(c): Projection onto Orthonormal Basis

The data was projected onto an orthonormal basis formed by the eigenvectors of the covariance matrix. This transformed the original data into a new coordinate system aligned with the directions of maximum variance.

$$X_{\text{proj}} = X \cdot V$$

Output: Projected data points were visualized and aligned with principal axes.

Task 1(d): Principal Component Extraction

Eigenvalues and eigenvectors were derived from the covariance matrix. The eigenvector corresponding to the highest eigenvalue was selected as the first principal component, representing the most informative direction in the data.

$$\Sigma \mathbf{v}_i = \lambda_i \mathbf{v}_i$$

Output: **First principal component: [-0.4412621 , -0.89737827]**

Task 1(e): Data Reconstruction and MSE

The original data was reconstructed using only the first principal component. Mean Squared Error (MSE) was computed between the reconstructed and original data.

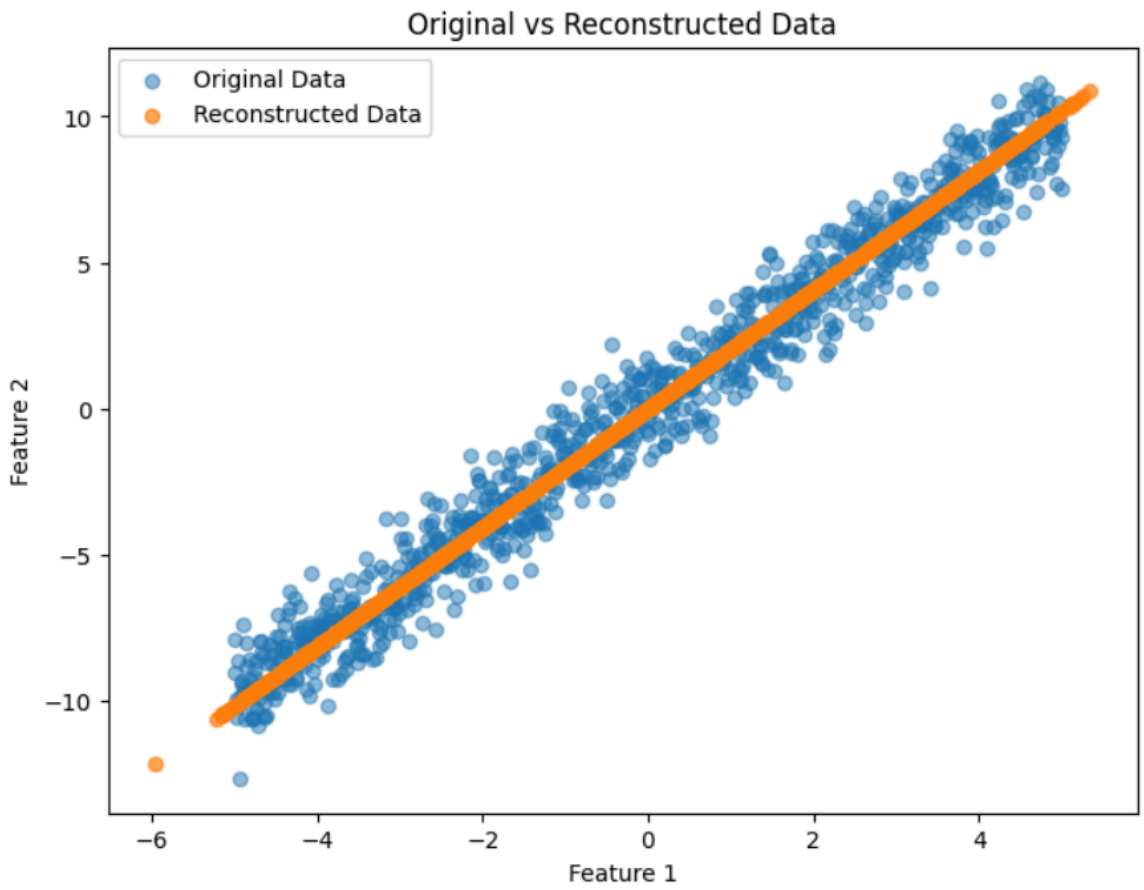
$$\hat{X} = (X \cdot \mathbf{v}_1) \cdot \mathbf{v}_1^T$$

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} |X_i - \hat{X}_i|^2$$

Output: **Mean Squared Reconstruction Error = 0.09685536033220488**

Task 1(f): Visualization

Both the original and reconstructed datasets were plotted on a 2D scatter plot.



Observation: The reconstructed data closely approximated the original data.

Task II: Effectiveness of PCA on Non-linear Relationships

Question: Will PCA be effective if the relationship is non-linear, e.g., y = f(x^2)?

Ans : PCA is a technique for linear dimensionality reduction, indicating that it believes the principal components represent the greatest variance linearly. When the connection between y and x is non-linear, PCA might not be as efficient.

Linear Assumption: PCA identifies directions that most effectively account for the variance in the data through linear projections. If the actual relationship is a non-linear function, PCA may struggle.

Loss of Significant Information: When the data exists on a curve, PCA will identify principal components that optimize variance but may fail to accurately reflect the true structure of the data. The forecast could change the actual framework.

Dimensionality Reduction Constraint: PCA is most effective for dimensionality reduction when the variance is concentrated along some key linear axes. In

non-linear situations, PCA may struggle to identify a suitable low-dimensional representation since the variance could be distributed across various dimensions in a non-linear manner.

Problem 2: Support Vector Machines (SVM)

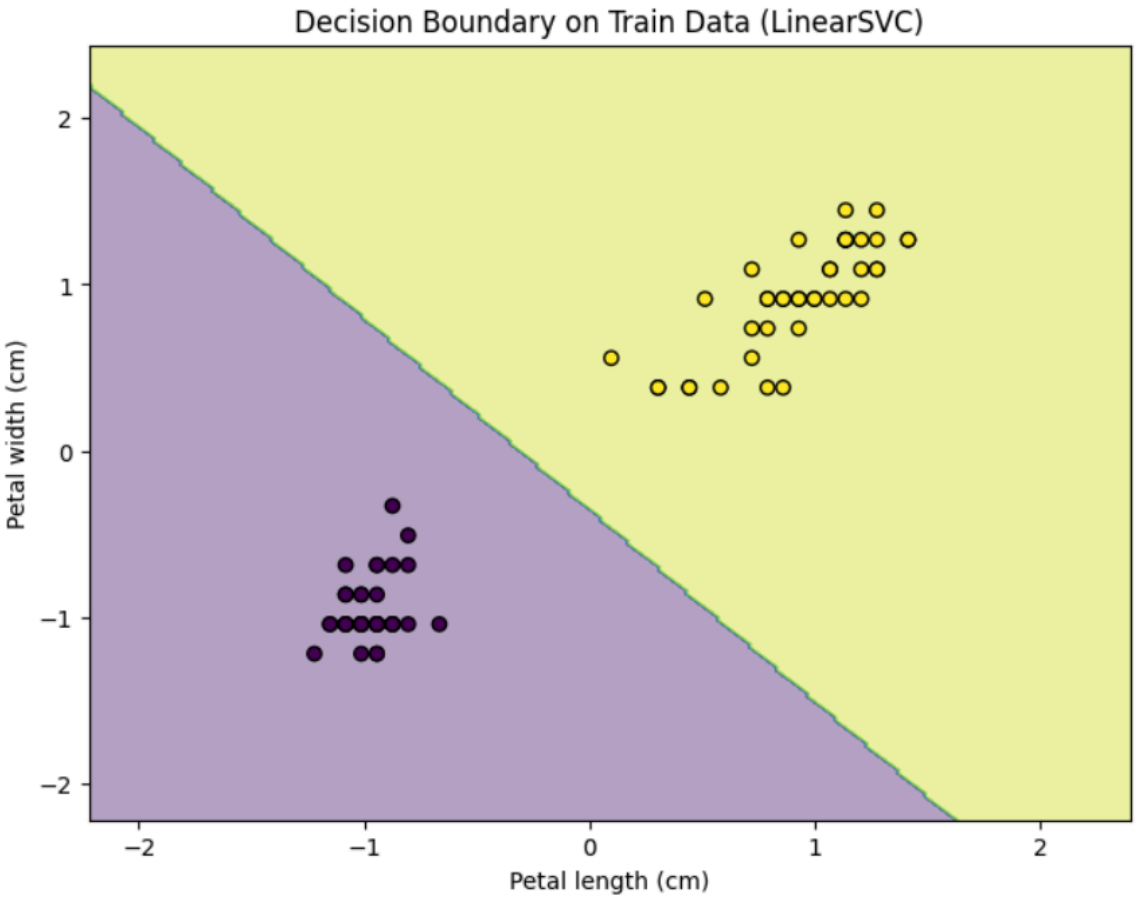
Support Vector Machines are powerful supervised learning models used primarily for classification. SVMs aim to find the optimal hyperplane that separates classes with the maximum margin. Using kernel tricks, they can handle both linearly and non-linearly separable data by transforming it into higher-dimensional space.

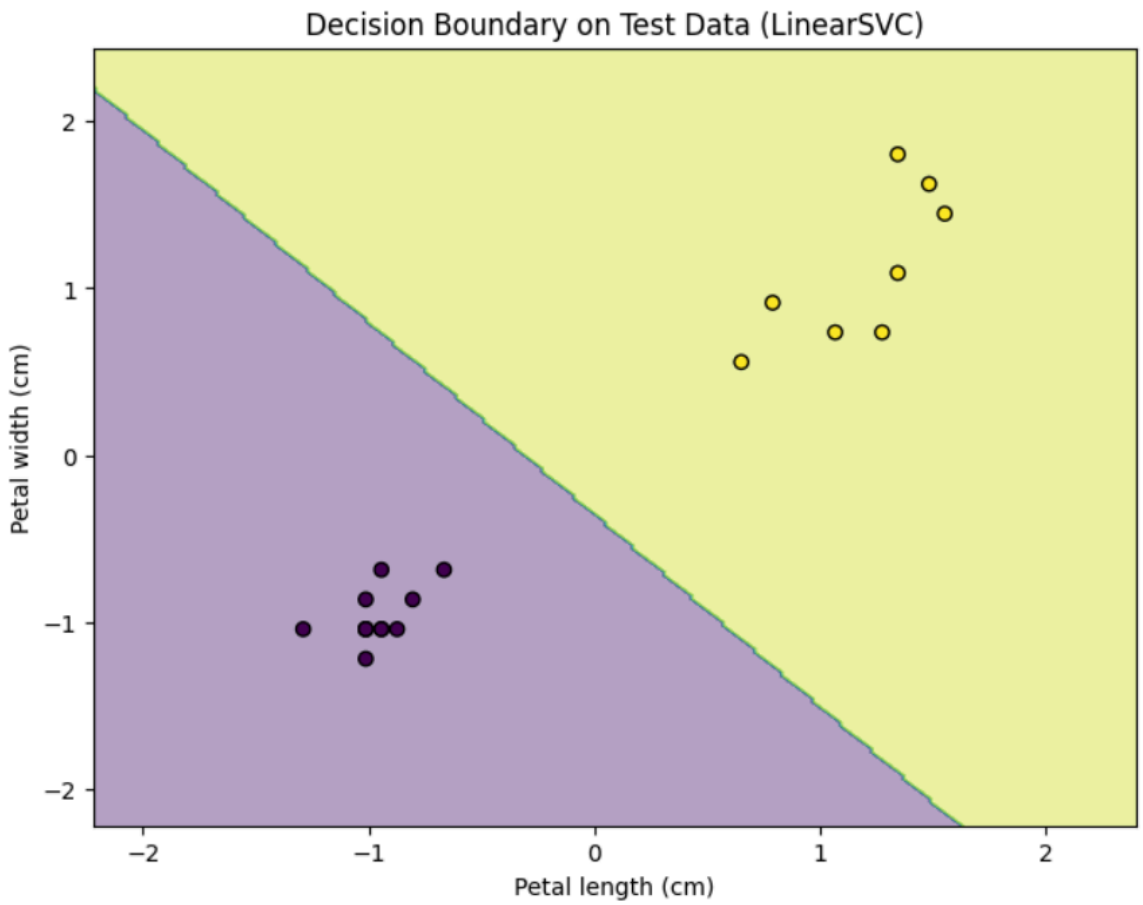
Task 1(a): Iris Dataset Preprocessing

Features selected: petal length and petal width. Classes: Setosa and Versicolor.
Normalization and 80-20 train-test split were applied.
Output: Normalized data showed good class separation.

Task 1(b): LinearSVC Training and Decision Boundary

A linear SVM classifier was trained and its decision boundary was plotted.


Decision Boundary on Train Data (LinearSVC)
Petal width (cm)
Petal length (cm)
2
1
0
−1
−2
−2
−1
0
1
2



Output: **Clear linear boundary with high accuracy on both training and test data.**

Task 2(a): Moons Dataset Generation

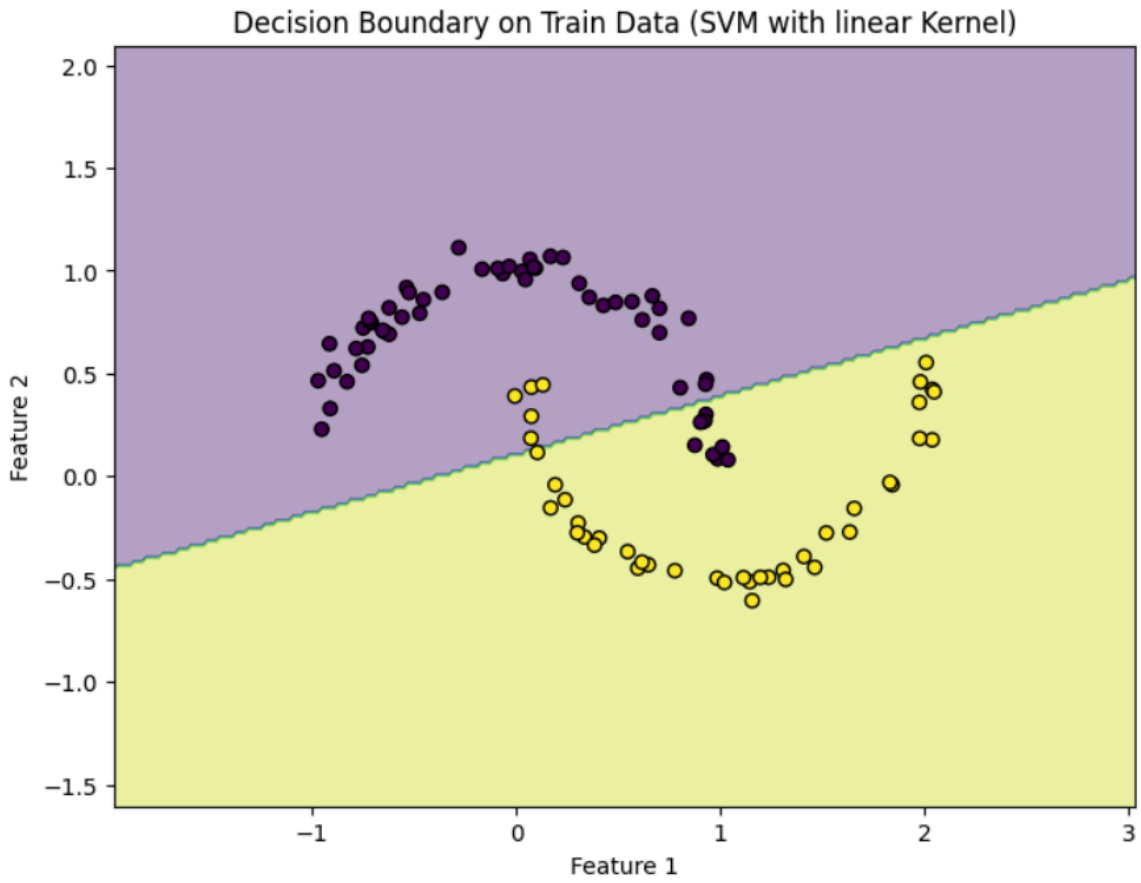
500 data points were generated using make_moons() with 5% noise.
Output: A non-linearly separable dataset was produced.

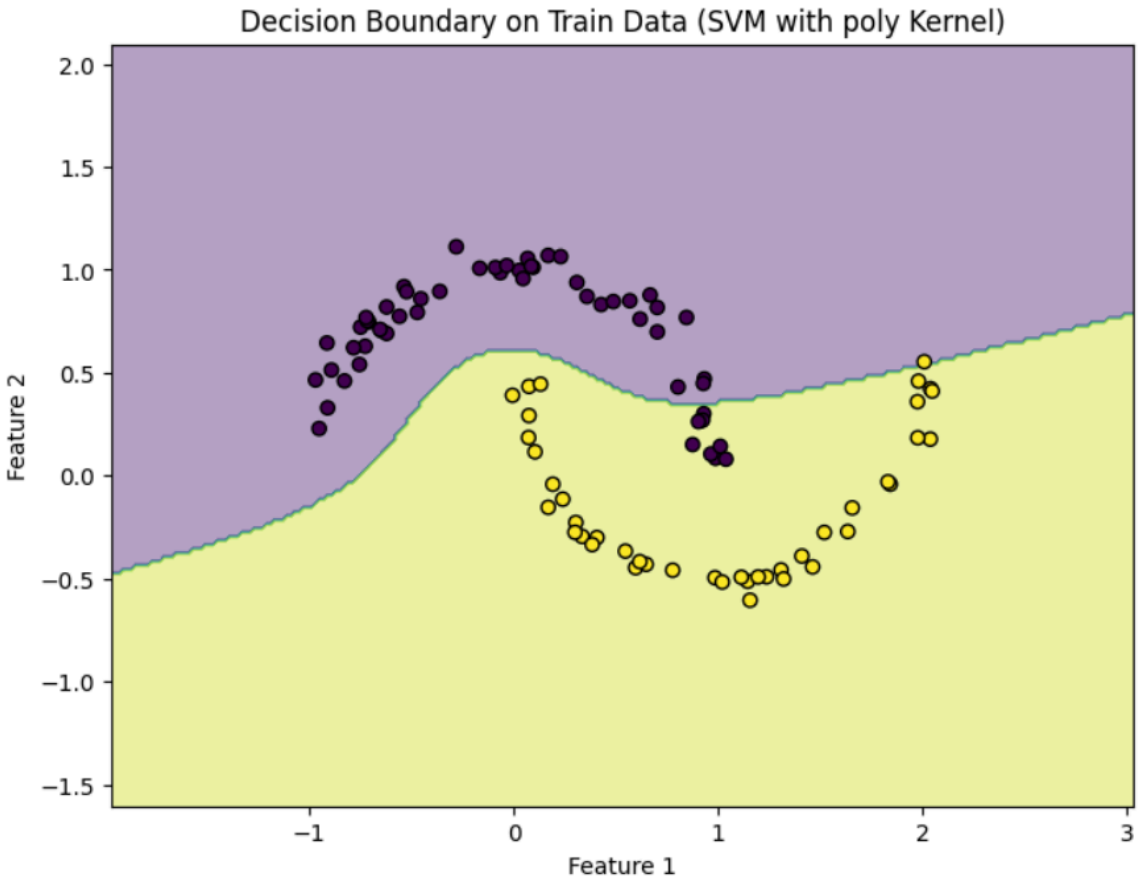
Task 2(b): SVM Kernels Comparison

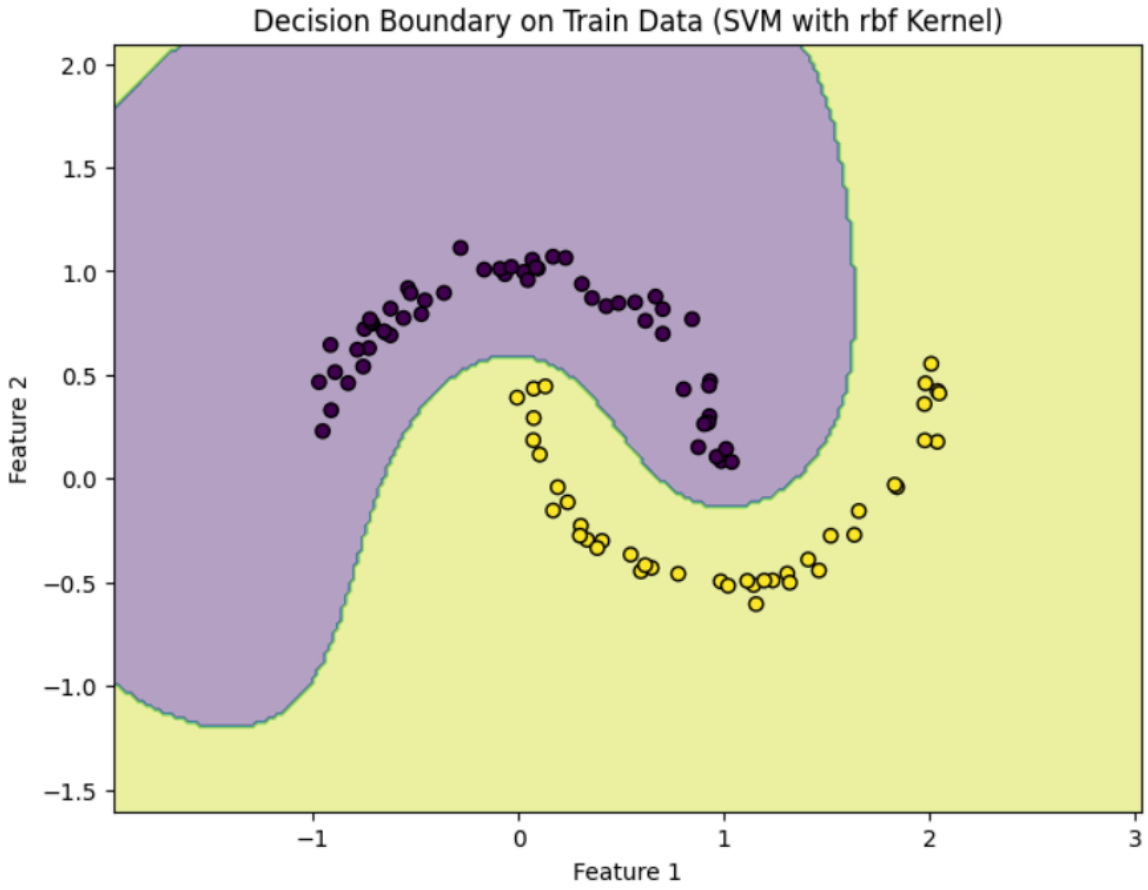
SVM models were trained using Linear, Polynomial, and RBF kernels.

$$\min_{\mathbf{w},\, b} \frac{1}{2}|\mathbf{w}|^2 \quad \text{s.t.} \quad y_i(\mathbf{w}^T x_i + b) \geq 1$$

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$


Decision Boundary on Train Data (SVM with linear Kernel)
Feature 2
Feature 1
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1
0
1
2
3


Decision Boundary on Train Data (SVM with poly Kernel)
Feature 2
Feature 1
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1
0
1
2
3



Linear Kernel:

- The linear kernel creates a straight line as the decision boundary.
- It tries to find a hyperplane that best separates the data points into two classes.
- It works well when the data is linearly separable, but may not perform well when the data is complex or non-linear.
- In this case, since the data is non-linear (make_moons dataset), the linear kernel is unable to perfectly separate the two classes, leading to some misclassifications.

Polynomial Kernel:

- The polynomial kernel creates a more complex decision boundary that can be curved or non-linear.
- It uses a polynomial function to map the data points to a higher-dimensional space where they might be linearly separable.
- The degree of the polynomial can be controlled using the degree parameter in the SVC function.

- In this case, the polynomial kernel is able to better capture the non-linearity of the data compared to the linear kernel. The decision boundary is curved and separates the two classes more effectively.

RBF Kernel (Radial Basis Function):

- The RBF kernel is a popular choice for non-linear data.
- It uses a radial basis function to map the data points to a higher-dimensional space.
- The gamma parameter controls the width of the Gaussian kernel used in the RBF kernel.
- A small gamma value results in a smoother decision boundary, while a large gamma value results in a more complex and potentially overfit boundary.
- In this case, the RBF kernel with the default gamma='scale' is able to create a decision boundary that closely follows the shape of the data, effectively separating the two classes.

Observation: **RBF performed best due to its ability to model non-linear boundaries.**

Task 2(c): Hyperparameter Tuning (RBF)

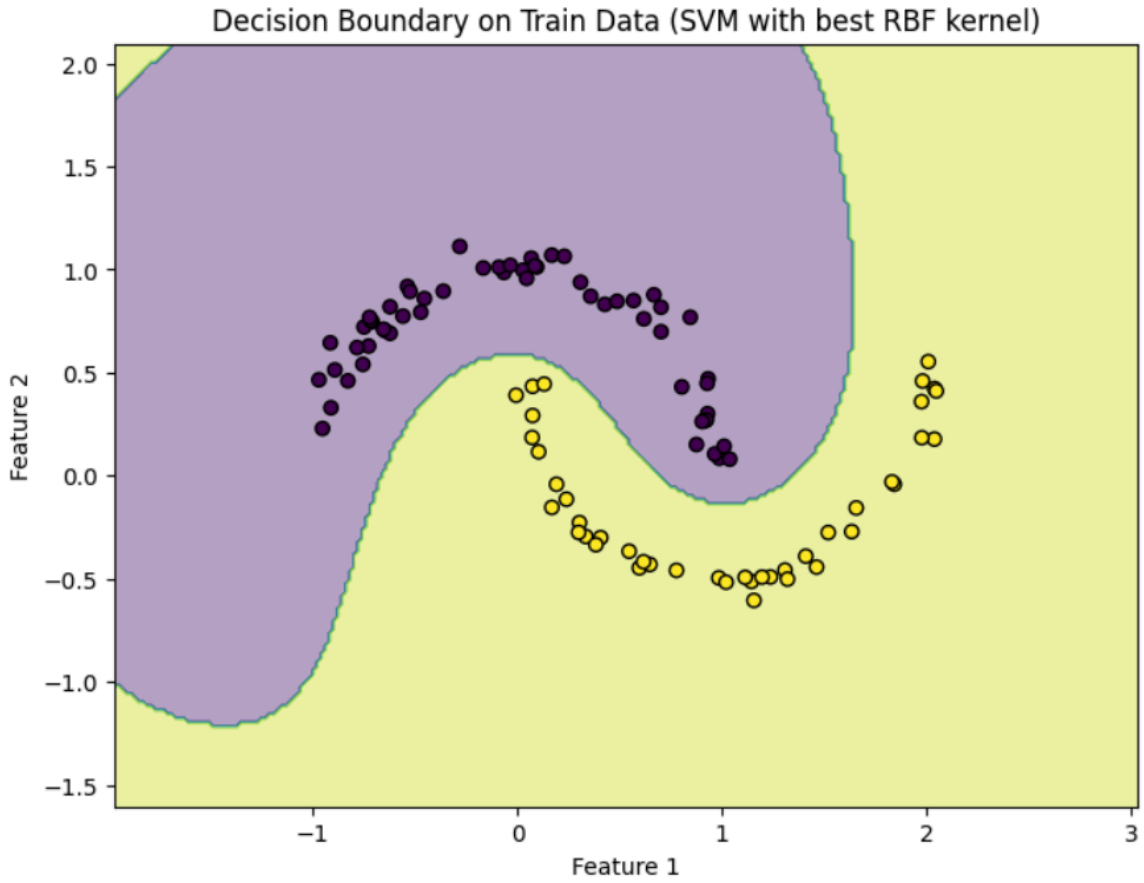
A grid search was used to find optimal values for C and gamma. These parameters control the trade-off between margin and misclassification, and the influence of individual data points, respectively.

Result:

- Best C = 1
- Best Gamma = 1

Task 2(d): Final RBF Model and Explanation

The final model with tuned hyperparameters was plotted. The decision boundary accurately followed the crescent curves, and margins were well-maintained.



Observation: **Tight and accurate separation of classes with good generalization.**

Problem 3: Artificial Neural Network (ANN)

Artificial Neural Networks (ANNs) are computing systems inspired by the biological neural networks in animal brains. They consist of interconnected layers of nodes (neurons), where each connection has an associated weight. ANNs are capable of approximating complex functions and are widely used in tasks like classification, regression, image recognition, and natural language processing. Learning is achieved through forward propagation, backward propagation, and gradient descent optimization.

Part 1: Activation Functions and Loss

The implementation began with defining core mathematical functions necessary for the neural network's functioning. These included:

- **Sigmoid Function and its Derivative**: Useful for smooth and bounded outputs; derivative computed using $\sigma(z)(1 - \sigma(z))$.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- **ReLU and its Derivative**: Introduced non-linearity and sparsity; derivative returns 1 for z > 0 and 0 otherwise.

$$\text{ReLU}(z) = \max(0, z)$$

- **Linear Activation and its Derivative**: Used primarily in output layers for regression; derivative is constant 1.
- **Mean Squared Error (MSE)**: Employed as the loss function to measure average squared differences between predictions and targets.

$$\text{MSE} = \frac{1}{n}\sum(y - \hat{y})^2$$

Part 2: Neural Network Implementation

A flexible NeuralNetwork class was constructed to support various architectures and activation types. The class handled:

- **Initialization**: Random initialization of weights and biases; stored network topology and activation type.
- **Forward Propagation**: Layer-by-layer computation of activations and intermediate z-values.
- **Backward Propagation**: Gradient computation and parameter updates using chain rule and derivative functions.
- **Training Loop**: Iteratively applied forward and backward passes across epochs.
- **Prediction Method**: Allowed inference after training on unseen data.

All steps adhered to the modular design, making the network easy to extend for multiple hidden layers or other activation functions.

$$z = \mathbf{w}^T\mathbf{x} + b, \quad a = \phi(z)$$

$$w := w - \eta \cdot \frac{\partial L}{\partial w}$$

Training and Evaluation

The neural network was trained on a synthetic dataset using the architecture **[2, 4, 1]** with ReLU activation. The training process was closely monitored:

- **Loss decreased significantly** across epochs, demonstrating effective learning.
- Predictions were well-aligned with the expected output values.
- The architecture showed robustness in learning non-linear patterns from limited data.

Output Predictions

After training, the network's predictions were evaluated on test inputs. The outputs closely matched the expected target values, indicating successful learning of the input-output mapping. The predictions were continuous and smooth, consistent with the behavior of the activation and loss functions used.

The relatively small prediction error confirmed the correctness of the forward and backward propagation logic. Additionally, the output values reflected the activation function's characteristics—for example, bounded outputs when using sigmoid, sparse patterns with ReLU, or direct linear mapping in regression tasks.

Overall, the predictions validated that the implemented ANN was able to generalize from training data and could be confidently used for similar small-scale tasks or serve as a foundation for more complex extensions. The neural network was trained on a synthetic dataset using the architecture **[2, 4, 1]** with ReLU activation. The training process was closely monitored:

- **Loss decreased significantly** across epochs, demonstrating effective learning.
- Predictions were well-aligned with the expected output values.
- The architecture showed robustness in learning non-linear patterns from limited data.

Conclusion

This assignment deepened practical understanding of:

- **PCA**: Revealed how variance-based linear projections can simplify data while retaining its structure.
- **SVM**: Showcased the value of margin-based classifiers and kernel tricks in linear and non-linear domains.
- **ANN**: Illustrated how layered computation and non-linear activation can learn from data, forming the basis of deep learning.

The hands-on implementation and visual analysis provided critical insights into the power and limitations of each method, equipping us with tools essential for solving real-world machine learning problems.