Prediction of EyeState using Machine Learning Techniques

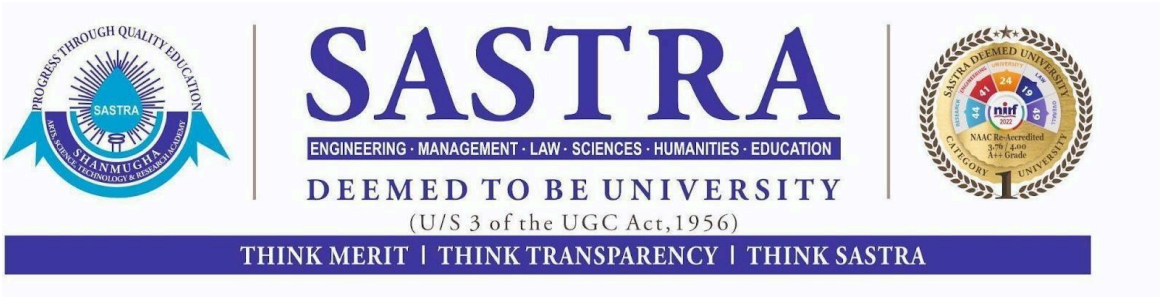
Report submitted to SASTRA Deemed to be University As per the requirement for the course

CSE425 : MACHINE LEARNING ESSENTIALS

Submitted by

Parepalli Rohith Narasimha Swamy Naidu
(Reg No: 125018055, B. Tech Computer Science and Business Systems)

OCTOBER- 2024



SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

Table of Contents

ABSTRACT

This project classifies eye states (open/closed) using EEG signals and machine learning techniques. Various algorithms, including Random Forest, SVM, Logistic Regression, and K-Nearest Neighbors (KNN), are applied to the dataset. The data is preprocessed using outlier removal and scaling techniques, and the models are evaluated with metrics like accuracy, precision, recall, and F1 score. KNN demonstrated the highest accuracy, outperforming other models.

INTRODUCTION

Importance of Dataset: EEG signals provide valuable insights into brain activity, including eye movements. Accurate classification of eye states from EEG data can be applied in fields like sleep monitoring and neurofeedback training.

Project Objective: The primary objective of the Project is to classify the eye state (open/closed) based on EEG signals using machine learning models, optimized through hyperparameter tuning.

Problem Statement:

The task of accurately predicting the eye state (open/closed) using EEG signals is a complex classification problem due to the non-linear nature of the data and the presence of noise. The **EEG Eye State** dataset consists of 14 EEG signal features that reflect brain activity at each timestamp. The challenge is to develop a machine learning model that can reliably classify the eye state based on these features.

Approach:

Use classification models with a focus on handling outliers and appropriate model selection.

Related Work

References:

Dataset: EEG Eye State - UCI Machine Learning Repository

OpenAI. (2024). *ChatGPT* (October 2024 version) [Large language model]. https://chat.openai.com/

Background

- **Models :**
    1. *Random Forest Classifier* : Random Forest is an ensemble learning method that combines multiple decision trees to improve accuracy and control overfitting. Each tree is trained on a random subset of the data, and the final output is based on the majority vote across all trees.

    2. *Support Vector Machines (SVM)*: SVM is a powerful classification algorithm that seeks to find the optimal hyperplane that best separates data points of different classes in a high-dimensional space. It is particularly effective for both linear and non-linear data.
    3. *Logistic Regression:* Logistic Regression is a simple and interpretable model used for binary classification tasks. It models the probability of an event occurring as a function of a linear combination of features.

4. *K-Nearest Neighbors (KNN):* KNN is a non-parametric, instance-based learning algorithm where the class of a data point is determined by the majority vote of its k nearest neighbors. It is effective for low-dimensional datasets but can struggle with high-dimensional spaces.

Mathematics Behind The Models

1. Logistic Regression for Classification:

Logistic Regression is a supervised learning algorithm used for binary classification problems. It models the probability of a binary outcome as a linear combination of input features, but instead of a linear relationship, it uses a logistic (sigmoid) function to map the output between 0 and 1, representing the probability of class membership.

Step 1: Hypothesis

In logistic regression, the hypothesis is based on a **linear combination** of the input features X, but the output is mapped through the **sigmoid function** to constrain it between 0 and 1. The hypothesis function is defined as:

$$h_\theta(X) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 X_1 + \theta_2 X_2 + \cdots + \theta_n X_n)}}$$

\

Here:

- $h_\theta(X)$ is the predicted probability that the output Y belongs to class 1 (e.g., $P(Y = 1|X)$),
- $\theta_0, \theta_1, \ldots, \theta_n$ are the model parameters (weights),
- $X_1, X_2, \ldots, X_n$ are the input features.

Step 2: Sigmoid (Logistic) Function

The sigmoid function ensures the predicted output is always between 0 and 1:

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

where $z = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \cdots + \theta_n X_n$. This function outputs values close to 0 for large negative z, and values close to 1 for large positive z.

Step 3: Decision Boundary

The predicted probability hθ(X) is then compared to a **threshold** (typically 0.5) to classify the outcome:

- If $h_\theta(X) \geq 0.5$, predict $Y = 1$,
- If $h_\theta(X) < 0.5$, predict $Y = 0$.

This threshold creates a decision boundary in the feature space where the classification changes from 0 to 1.

Step 4: Cost Function (Log-Loss)

Instead of minimizing the sum of squared errors (as in linear regression), logistic regression uses a **logistic loss function** (also known as log-loss or binary cross-entropy) to measure the error:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right]$$

Here:

- m is the number of training examples,
- $y^{(i)}$ is the actual label for the i-th training example,
- $h_\theta(x^{(i)})$ is the predicted probability for the i-th example.

Step 5: Optimization (Gradient Descent)

To minimize the cost function and find the best parameters θ, **Gradient Descent** is used. The update rule for gradient descent is:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

for each parameter θ_j, where:

- α is the learning rate (a small positive value that controls the step size),
- $\frac{\partial}{\partial \theta_j} J(\theta)$ is the derivative of the cost function with respect to θ_j.

The partial derivative of the cost function for logistic regression is:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Step 6: Regularization

To prevent overfitting, especially when dealing with high-dimensional data, **regularization** can be applied. A regularization term is added to the cost function to penalize large weights. The regularized cost function (for L2 regularization) becomes:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

where λ is the regularization parameter that controls the strength of the penalty.

2. Random Forest Classification:

Random Forest is an ensemble method that combines multiple decision trees. It improves accuracy by reducing overfitting through

random sampling of both data and features.

Step 1: Bootstrap Sampling

- Multiple subsets of the training data are generated by randomly sampling with replacement (bootstrapping).

Step 2: Tree Construction

- For each subset, a decision tree is built using a random subset of features. The tree is split based on a criterion such as Gini impurity or Entropy.
- **Gini Impurity:**

$$Gini(D) = 1 - \sum_{i=1}^{C} p_i^2$$

where pi is the proportion of samples in class i, and C is the number of classes.

Entropy (used in information gain):

$$H(D) = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

The goal is to maximize the information gain or reduce impurity at each split.

Support Vector Classification:

Support Vector Machines (SVM) aim to find the hyperplane that best separates the data into classes while maximizing the margin between the classes.

Step 1: Objective Function

- SVC tries to solve the optimization problem:

$$\min \frac{1}{2}||w||^2$$

subject to:

$$y_i(w \cdot x_i + b) \geq 1 \quad \forall i$$

Here, w is the weight vector, b is the bias term, and yi are the class labels. The goal is to find w and b such that the margin between the two classes is maximized.

Step 2: Support Vectors

- The points that lie on the edge of the margin are called **support vectors**. These points define the decision boundary.

Step 3: Kernel Trick (Non-linear data)

- For non-linearly separable data, SVM uses a **kernel trick** to map the data into a higher-dimensional space. A common kernel is the **Radial Basis Function (RBF)** kernel:

$$K(x, x') = \exp\left(-\gamma||x - x'||^2\right)$$

K-Nearest-Neighbours(KNN):

- KNN is a non-parametric algorithm that classifies a data point based on the majority vote of its k-nearest neighbors. The distance between data points is calculated using a distance metric like Euclidean distance:

$$d(x_i, x_j) = \sqrt{\sum_{l=1}^{n} (x_{il} - x_{jl})^2}$$

Dataset Description:

The Dataset used in this project consists of 14980 instances with 14 features.

The **EEG Eye State** dataset includes the following columns:

1. **AF3**: EEG signal from the left frontal lobe.
2. **F7**: EEG signal from the left frontal cortex.
3. **F3**: EEG signal from the frontal cortex.
4. **FC5**: EEG signal from the frontocentral area.

5. **T7**: EEG signal from the left temporal lobe.
6. **P7**: EEG signal from the parietal lobe.
7. **O1**: EEG signal from the left occipital lobe.
8. **O2**: EEG signal from the right occipital lobe.
9. **P8**: EEG signal from the parietal lobe.
10. **T8**: EEG signal from the right temporal lobe.
11. **FC6**: EEG signal from the frontocentral area.
12. **F4**: EEG signal from the frontal cortex.
13. **F8**: EEG signal from the right frontal cortex.
14. **AF4**: EEG signal from the right frontal lobe.

Data Sample:

	AF3	F7	F3	FC5	T7	P7	O1	O2	P8	T8	FC6	F4	F8	AF4	eyeDetecti
0	4329.23	4009.23	4289.23	4148.21	4350.26	4586.15	4096.92	4641.03	4222.05	4238.46	4211.28	4280.51	4635.90	4393.85	
1	4324.62	4004.62	4293.85	4148.72	4342.05	4586.67	4097.44	4638.97	4210.77	4226.67	4207.69	4279.49	4632.82	4384.10	
2	4327.69	4006.67	4295.38	4156.41	4336.92	4583.59	4096.92	4630.26	4207.69	4222.05	4206.67	4282.05	4628.72	4389.23	
3	4328.72	4011.79	4296.41	4155.90	4343.59	4582.56	4097.44	4630.77	4217.44	4235.38	4210.77	4287.69	4632.31	4396.41	
4	4326.15	4011.79	4292.31	4151.28	4347.69	4586.67	4095.90	4627.69	4210.77	4244.10	4212.82	4288.21	4632.82	4398.46	

Data Preprocessing:

- **Outlier Detection**: Z-score thresholding was applied to remove outliers with a Z-score greater than 3.
- **Feature Scaling**: Features were standardized using `StandardScaler`, transforming each feature to have a mean of 0 and a standard deviation of 1.
- **Data Splitting**: The dataset was split into 80% training and 20% testing sets.

Model Comparison Before and After Optimization:

Metric	Random Forest (Before)	Random Forest (After)	SVM (Before)	SVM (After)	Logistic Regression (Before)	Logistic Regression (After)	KNN (Before)	KNN (After)
Accuracy	91.30%	93.72%	89.50%	94.23%	60.50%	62.84%	93.50%	96.68%
Precision	0.91	0.93	0.90	0.94	0.61	0.62	0.92	0.97
Recall	0.91	0.95	0.89	0.95	0.75	0.77	0.94	0.97
F1 Score	0.91	0.94	0.89	0.94	0.68	0.69	0.93	0.96
ROC-AUC	0.91	0.94	0.90	0.94	-	-	-	-

- **Before Optimization**: Default parameters were used, resulting in lower accuracy and F1 scores across models, especially for Logistic Regression.

- **After Optimization**: Hyperparameter tuning (e.g., adjusting `n_estimators`, `max_depth`, `C` values) improved accuracy and recall, especially for KNN and SVM.

Model Tuning

Random Forest:

- Tuned `n_estimators` (number of trees) and `max_depth`.
- Best Parameters: `n_estimators=200`, `max_depth=30`.

SVM:

- Tuned the penalty term `C` and kernel (`linear`, `rbf`, `poly`).
- Best Parameters: `C=10`, `kernel='rbf'`.

Logistic Regression:

- Tuned the regularization parameter `C`.

- Best Parameters: `C=1`.

KNN:

- Tuned the number of neighbors (`n_neighbors`) and weighting scheme (`uniform` vs `distance`).
- Best Parameters: `n_neighbors=3`, `weights='distance'`.

Results

Random Forest:

- Achieved an accuracy of **93.72%** after tuning, with balanced precision (0.93) and recall (0.95).

SVM:

- After optimization, SVM reached an accuracy of **94.23%**, with high precision (0.94) and recall (0.95).

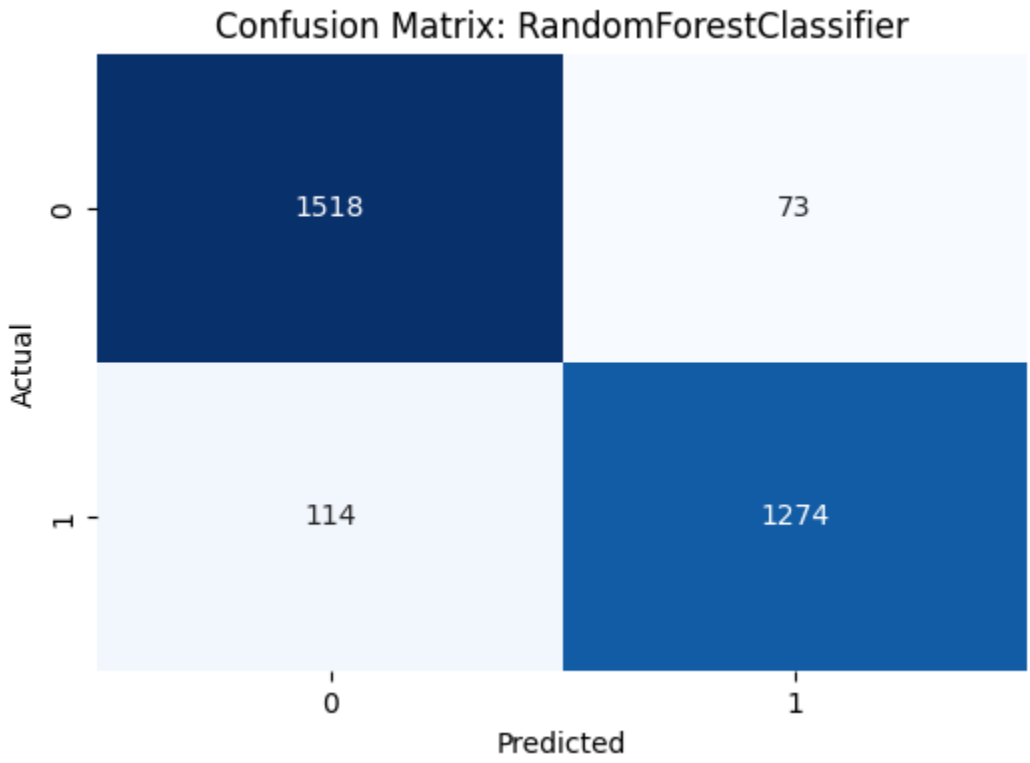
Logistic Regression:

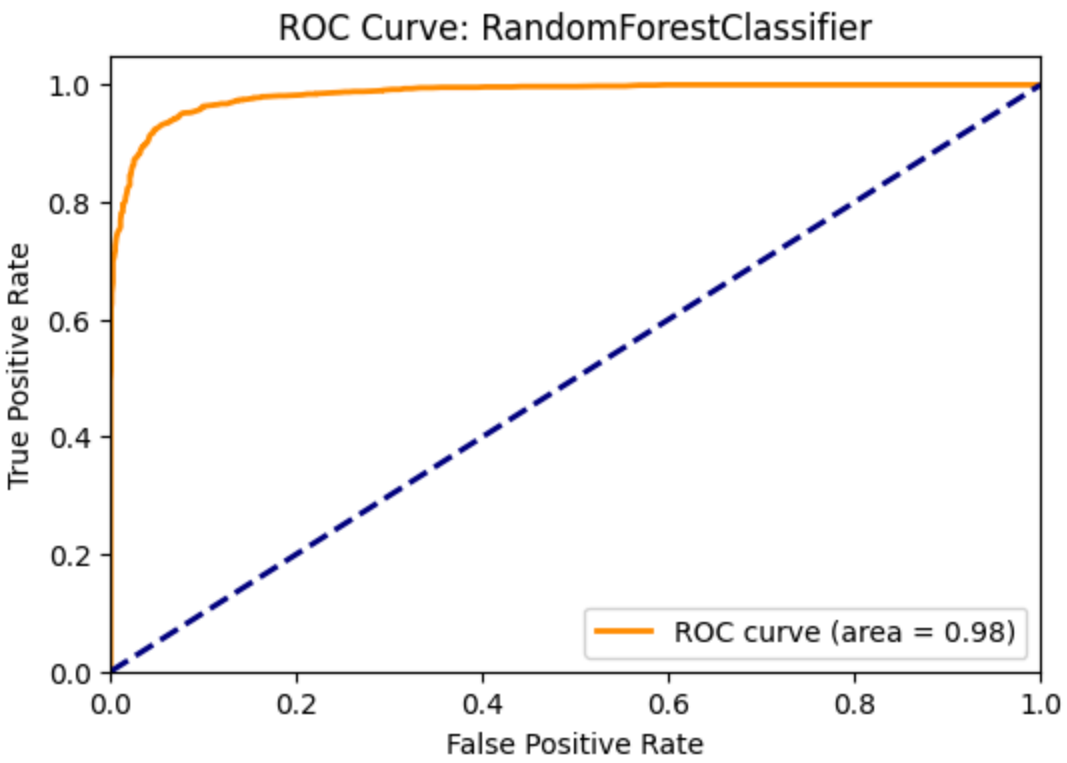
- With limited non-linear modeling capabilities, Logistic Regression achieved **62.84%** accuracy after tuning.

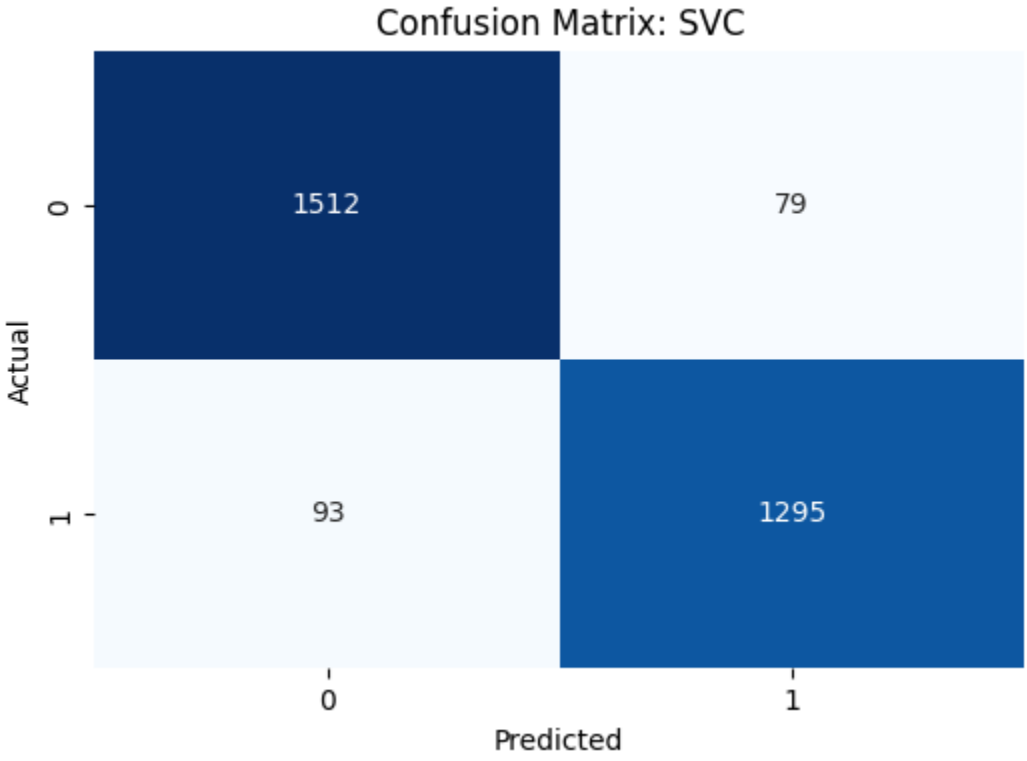
KNN:

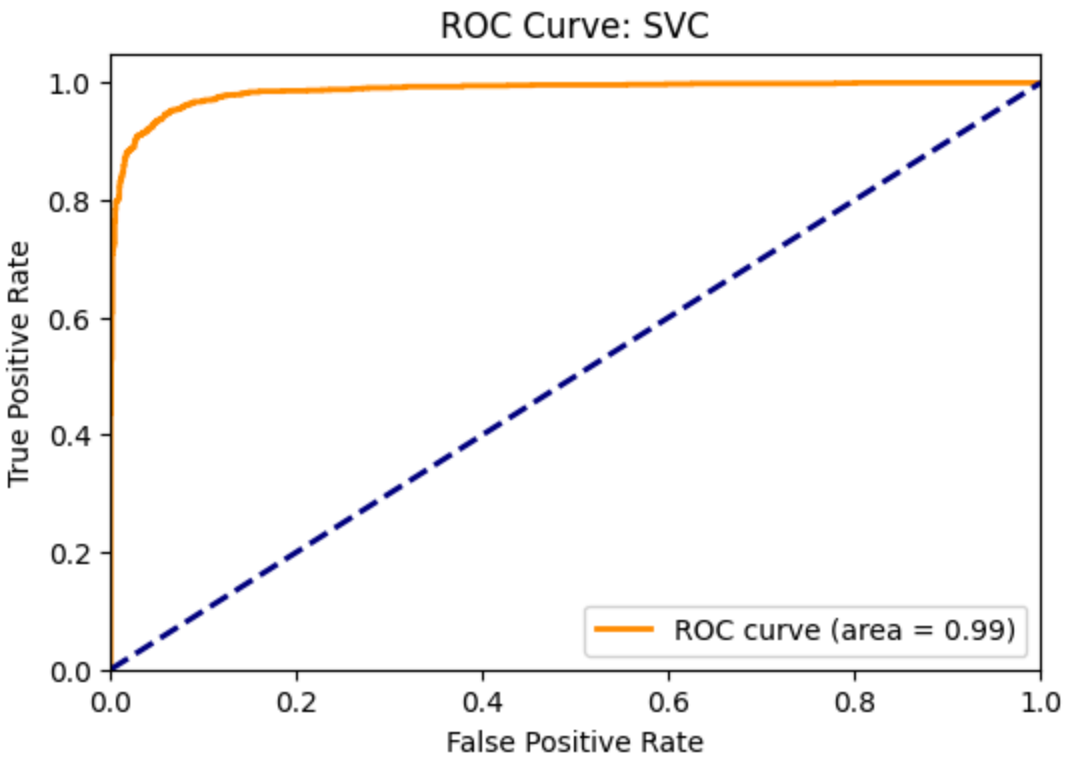
- Best performer with **96.68%** accuracy after tuning, showing high precision (0.97) and recall (0.97).

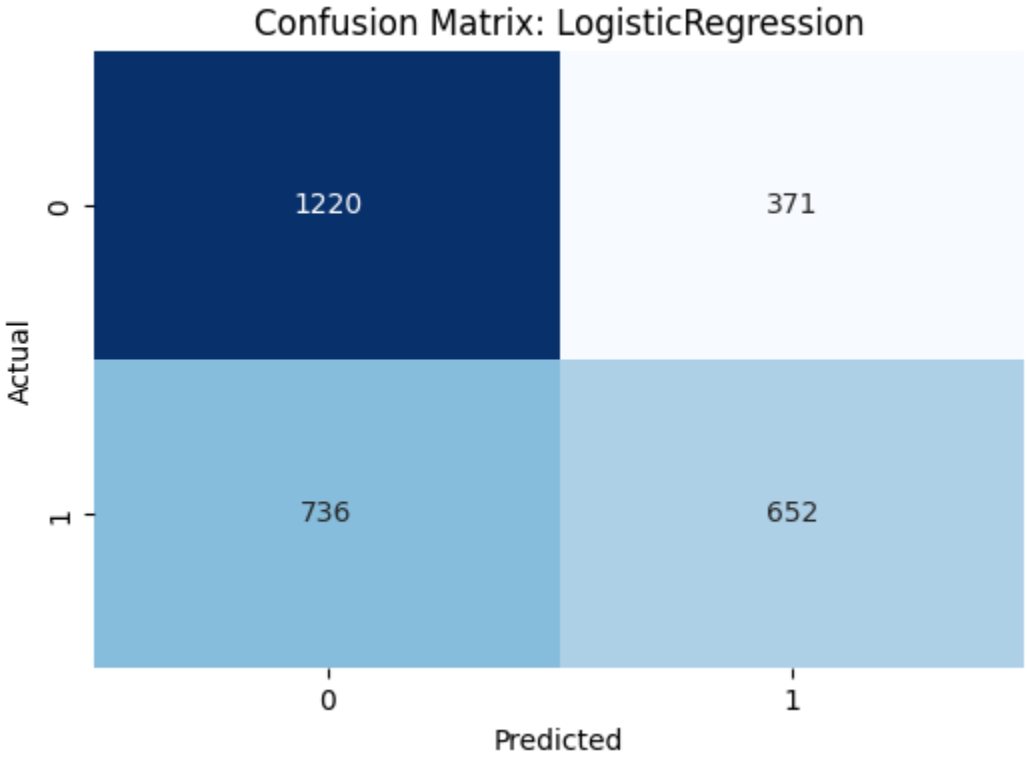
Result Visualization:

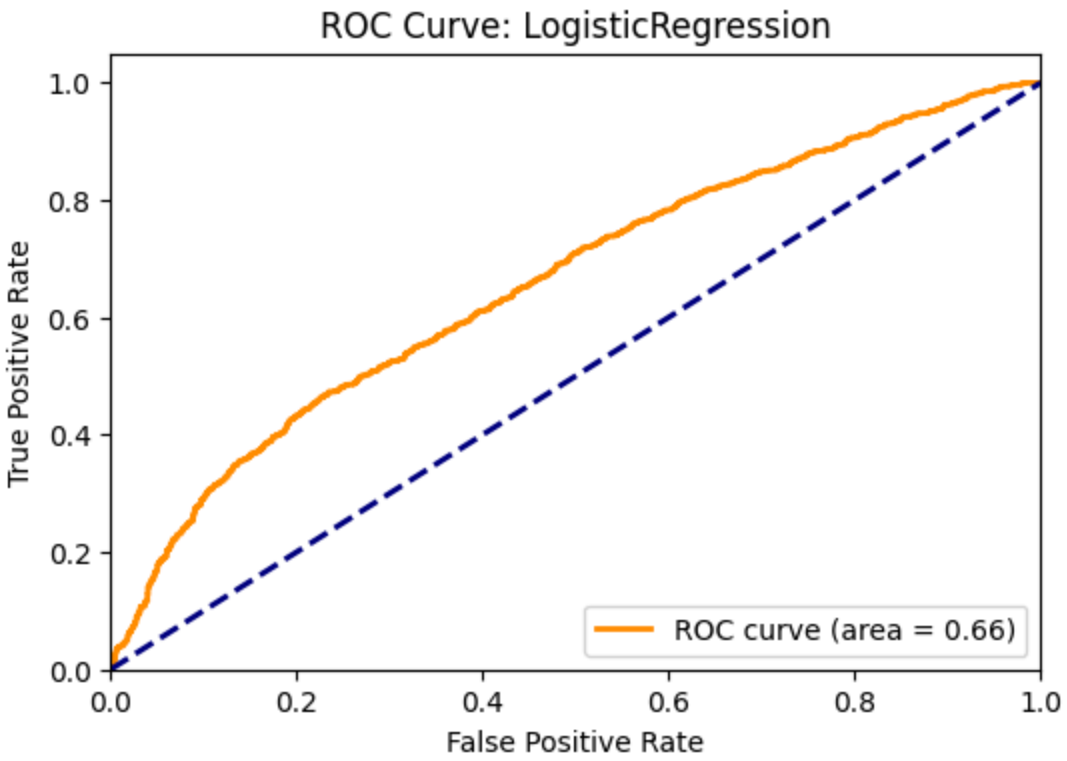


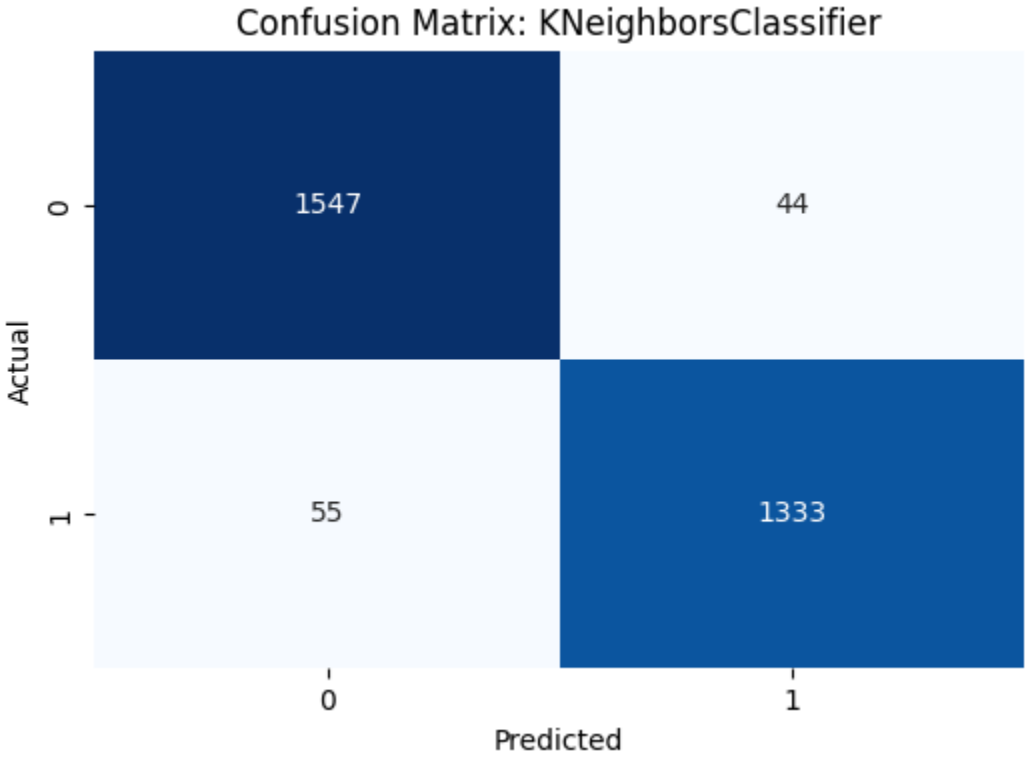

ROC Curve: RandomForestClassifier
True Positive Rate
False Positive Rate
ROC curve (area = 0.98)
0.0
0.2
0.4
0.6
0.8
1.0

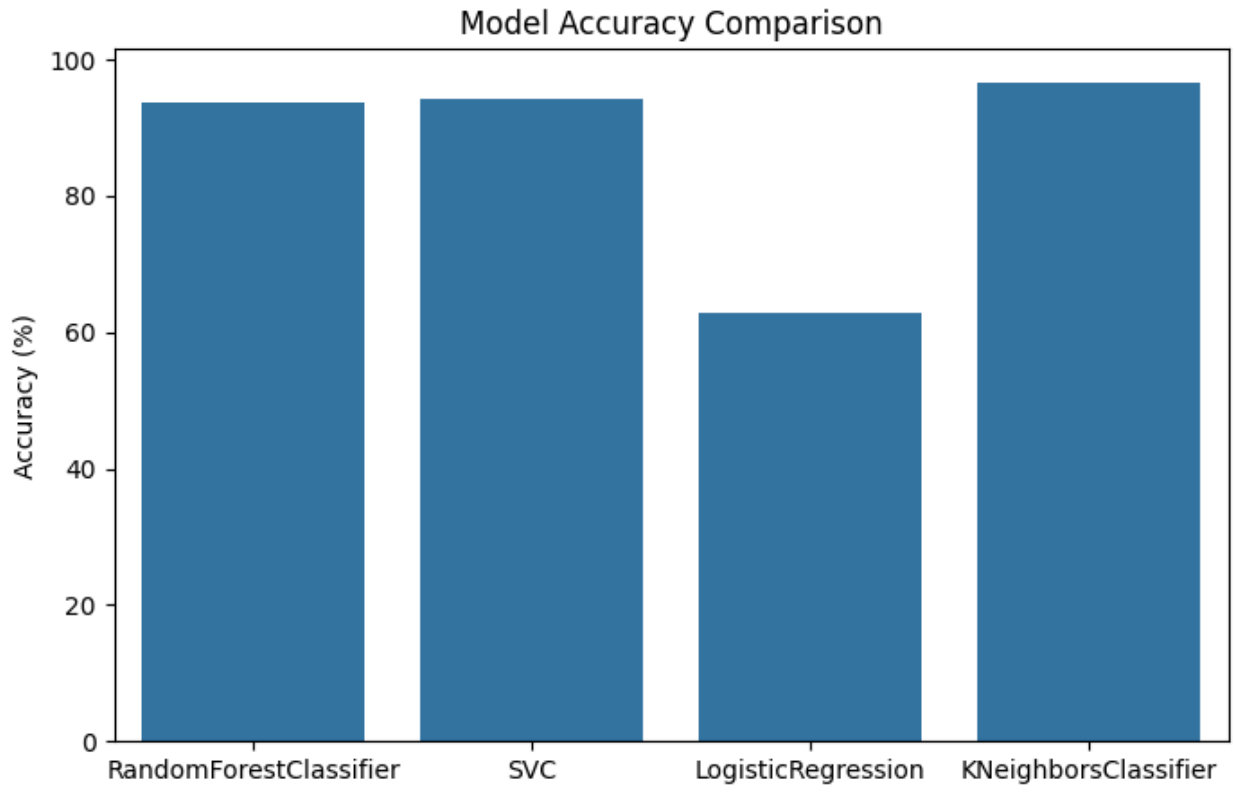

Confusion Matrix: SVC
Actual
Predicted
1512
79
93
1295
0
1


ROC Curve: SVC
True Positive Rate
False Positive Rate
ROC curve (area = 0.99)
0.0
0.2
0.4
0.6
0.8
1.0


Confusion Matrix: LogisticRegression
Actual
Predicted
1220
371
736
652
0
1


ROC Curve: LogisticRegression
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.66)


Confusion Matrix: KNeighborsClassifier
Actual
Predicted
0
1
1547
44
55
1333



Discussion:

The results indicate that ensemble and distance-based models (KNN, Random Forest) perform better for this EEG dataset. SVM also showed excellent performance with optimized hyperparameters. Logistic Regression, being a linear model, underperformed due to the complex nature of the data.

Learning Outcomes:

1. **Understanding of EEG-Based Classification:**
    - Gained experience in applying machine learning models to EEG data for classification tasks.
    - Learned about the unique challenges posed by EEG signal processing, such as handling noise and ensuring proper feature scaling.
2. **Model Evaluation and Comparison:**
    - Acquired the ability to evaluate and compare machine learning models using metrics like accuracy, precision, recall, F1 score, and ROC-AUC.
    - Learned the strengths and weaknesses of models such as Random Forest, SVM, Logistic Regression, and KNN, and when to apply them in different problem contexts.
3. **Hyperparameter Tuning:**
    - Developed expertise in tuning hyperparameters using GridSearchCV to optimize model

performance, improving the accuracy and generalization of models on unseen data.

- Understood the importance of choosing the right hyperparameters and their impact on the final model performance.

4. Model Selection for Classification:

- Gained experience in selecting appropriate classification algorithms for EEG-based tasks, discovering that KNN and SVM were particularly effective for non-linear EEG classification tasks.

5. Hands-On Experience with Python Libraries:

- Enhanced proficiency in machine learning libraries such as Scikit-learn for model building, tuning, and evaluation, and Matplotlib/Seaborn for visualizing model performance (e.g., confusion matrix, ROC curves).

Conclusion:

The project successfully demonstrated the ability of machine learning models to classify eye states using EEG signal data. Through the application of various classification models, it became evident that distance-based models, like KNN, and kernel-based models, like SVM, were the most effective for this task.

After hyperparameter tuning, KNN achieved the highest accuracy of 96.68%, followed by SVM at 94.23%, both outperforming simpler linear models like Logistic Regression, which only achieved 62.84% accuracy. Random Forest also performed well, with an accuracy of 93.72%, leveraging the ensemble nature of decision trees to capture complex relationships in the data.

The model performance was evaluated on key metrics, showing that KNN and SVM not only achieved high accuracy but also maintained balanced precision and recall, making them suitable for real-world applications where both positive and negative cases (eyes open/closed) need to be correctly classified.

This project highlights the importance of data preprocessing, particularly outlier removal and feature scaling, in achieving optimal results. The use of GridSearchCV for hyperparameter tuning was critical in improving model performance, demonstrating the significance of optimizing machine learning models for better generalization.