Classify Eye State Analysis for Eye Detection

Shaheer Ahsan, Mustafa Ssaid, Shahd Abu-Obeid, Anand Sabnis

2024-12-02

Table of contents

Introduction

The purpose of this project is to classify the eye state (open or closed) using EEG data collected from 14 electrodes placed on the scalp. The dataset consists of 10,486 observations with 14 numerical predictors and one binary response variable (eyeDetection). The focus is to evaluate the performance of a Support Vector Machine (SVM) model with a Radial Basis Function (RBF) kernel. This model is chosen for its ability to handle non-linear patterns effectively, which are common in EEG data. Additionally, we will also evaluate the performance of a K-Nearest Neighbors (KNN) model. KNN classifies data based on the majority class of nearby points, with the number of neighbors (k) being a key parameter. Features were scaled to ensure fair distance calculations, and cross-validation was used to select the optimal k. The model's performance was evaluated using error rates, confusion matrices, and visualizations to compare its effectiveness with SVM.

Methodology/Models

The models that will be implemented for this research project include **Support Vector Machine (SVM)** and **K-Nearest Neighbor (KNN)**

1. Radial SVM uses the RBF kernel, represented as:

K(x, x') = exp(-y || x - x '||^2)

where:

- y controls the influence of individual points.
- cost balances classification accuracy and margin width.

The dataset was divided into 80% for training and validation, and 20% for testing. Features were scaled to ensure equal importance. The best hyperparameters were identified through 5-fold cross-validation, optimizing both cost and . Performance was assessed using error rates, confusion matrices, and visualizations.

2. K-Nearest Neighbors (KNN) is a non-parametric, instance-based learning algorithm that classifies data points based on the majority class of their nearest neighbors. The key parameter in KNN is the number of neighbors k, which controls the size of the neighborhood used for classification.

The dataset was divided into 80% for training and validation, and 20% for testing. Features were scaled to ensure equal importance, as KNN relies on distance metrics like Euclidean distance, which are sensitive to the scale of features. The optimal value of k was identified through 5-fold cross-validation by minimizing the validation error rate. Performance was assessed using error rates, confusion matrices, and visualizations.

Data Analysis

1. Support Vector Machine (Shaheer Ahsan & Mustafa Ssaid)

Load Data

```
# Load training and test data
train_data <- read.csv("train.csv")
test_data <- read.csv("test.csv")  # No eyeDetection column

# Convert eyeDetection to a factor
train_data$eyeDetection <- as.factor(train_data$eyeDetection)

# Check data structure
str(train_data)
```

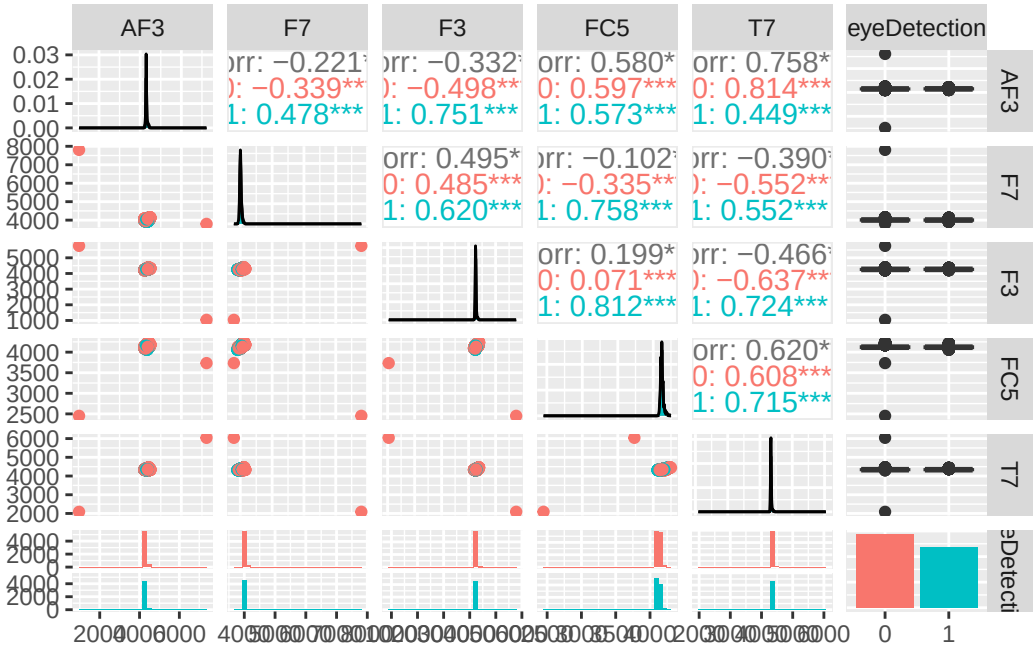
```
'data.frame':   10486 obs. of  15 variables:
 $ AF3          : num  4279 4294 4291 4289 4376 ...
 $ F7           : num  4004 4037 4000 4001 4006 ...
 $ F3           : num  4260 4264 4261 4256 4283 ...
```

```
$ FC5          : num  4113 4123 4106 4115 4142 ...
$ T7           : num  4332 4342 4339 4327 4322 ...
$ P7           : num  4607 4617 4622 4614 4598 ...
$ O1           : num  4038 4071 4061 4070 4065 ...
$ O2           : num  4585 4618 4629 4587 4606 ...
$ P8           : num  4163 4206 4197 4185 4193 ...
$ T8           : num  4201 4230 4222 4214 4247 ...
$ FC6          : num  4176 4208 4201 4180 4209 ...
$ F4           : num  4257 4276 4283 4271 4311 ...
$ F8           : num  4590 4612 4612 4597 4684 ...
$ AF4          : num  4325 4361 4364 4356 4454 ...
$ eyeDetection: Factor w/ 2 levels "0","1": 2 1 1 1 2 1 1 1 1 1 ...
```

Exploratory Data Analysis

```
# Pairwise plot of first few features
ggpairs(train_data[, c(1:5, 15)], aes(color = eyeDetection))
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```r
# Correlation between first two electrodes
cor(train_data$AF3, train_data$F7)
```

```
[1] -0.2207714
```

The correlation between the first two features (AF3 and F7) was found to be weak ($= -0.22$), indicating low collinearity. Similar patterns were observed across other features, suggesting that all predictors could contribute to the model without significant redundancy.

Scaling and Data Splitting

```r
# Scale features
set.seed(123)
train_scaled <- scale(train_data[, -15])
test_scaled <- scale(test_data, center = attr(train_scaled, "scaled:center"),
                     scale = attr(train_scaled, "scaled:scale"))

# Add eyeDetection back to the scaled training data
train_scaled <- data.frame(train_scaled)
train_scaled$eyeDetection <- train_data$eyeDetection

# Split training data into train (80%) and validation (20%)
n <- nrow(train_scaled)
train_idx <- sample(1:n, 0.8 * n)
train_set <- train_scaled[train_idx, ]
val_set <- train_scaled[-train_idx, ]
```

Model Tuning

```r
# Tune SVM with radial kernel
set.seed(123)
svm_tune <- tune(svm,
                 eyeDetection ~ .,
                 data = train_set,
                 kernel = "radial",
                 ranges = list(cost = c(1, 10), gamma = c(0.1, 1)),
                 tunecontrol = tune.control(cross = 5))

# Best model summary
summary(svm_tune)
```

```
Parameter tuning of 'svm':

- sampling method: 5-fold cross validation

- best parameters:
 cost gamma
   10     1

- best performance: 0.05483954

- Detailed performance results:
  cost gamma      error  dispersion
1    1   0.1 0.20457348 0.033314898
2   10   0.1 0.14508214 0.039521957
3    1   1.0 0.07951547 0.019468043
4   10   1.0 0.05483954 0.008510931
```

The Radial SVM model was tuned over the following ranges:

- cost: {1, 10}
- y: {0.1, 1}

Results:

- Best parameters: cost = 10, y = 1
- Cross-validation error: 5.48%

  This low error rate demonstrates the model's ability to generalize across folds.

Model Evaluation

```r
# Predict on validation set
val_predictions <- predict(svm_tune$best.model, val_set[, -15])

# Confusion matrix
val_conf_matrix <- table(True = val_set$eyeDetection, Predicted = val_predictions)
val_conf_matrix
```

```
    Predicted
True    0    1
   0 1090   45
   1   55  908
```

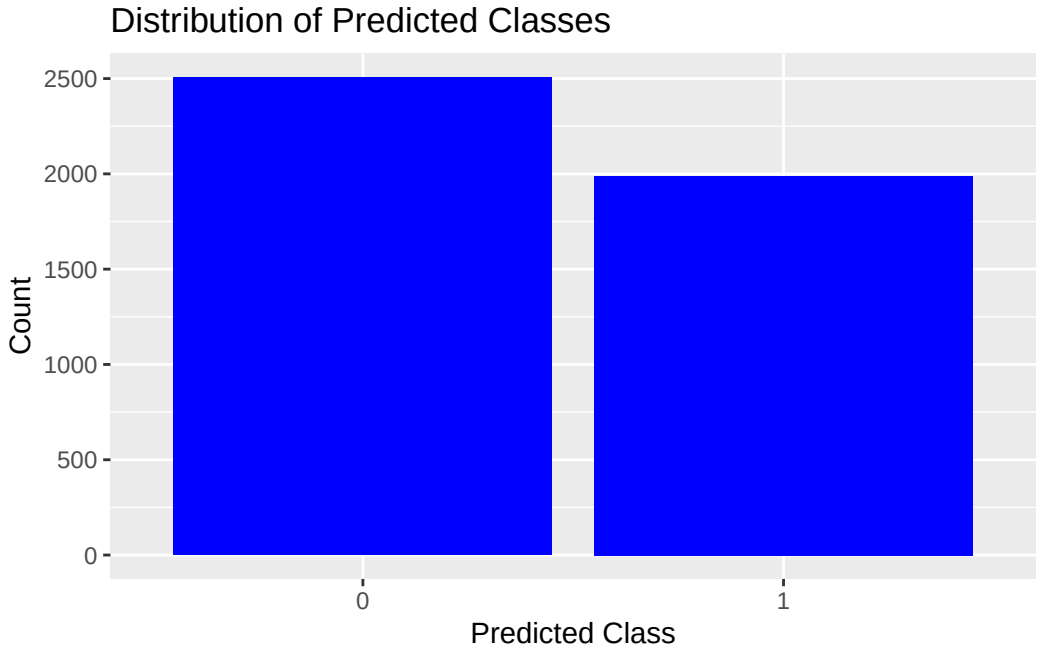
```r
# Error rate
val_error_rate <- mean(val_predictions != val_set$eyeDetection)
val_error_rate
```

```
[1] 0.04766444
```

The validation error rate of **4.77%** highlights the model's strong performance. Most misclassifications occur in borderline cases, which is expected with noisy EEG data.

Test Set Prediction

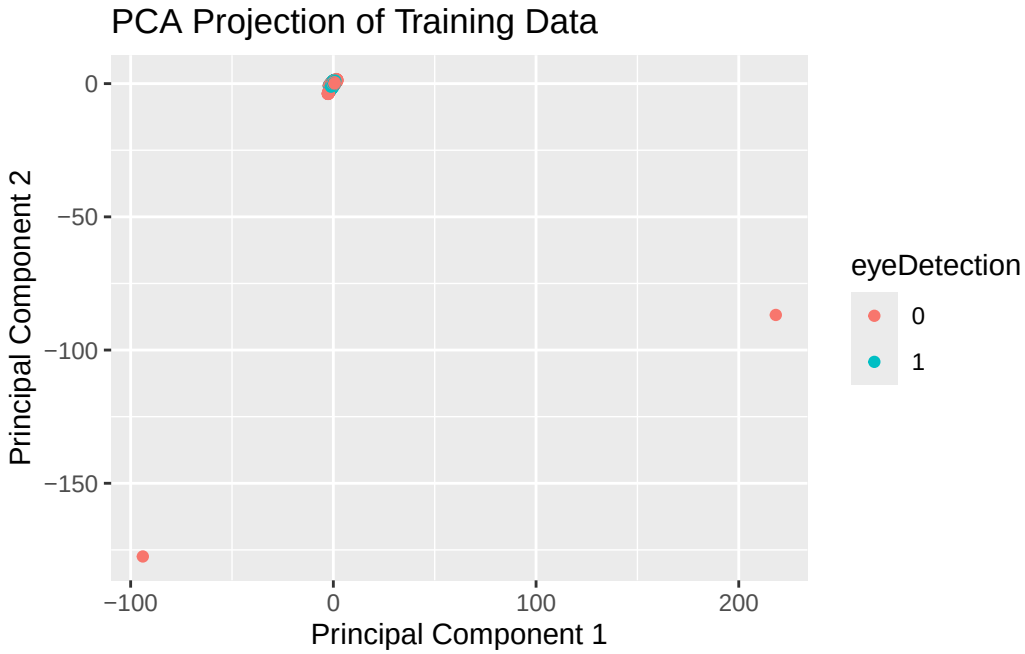
```r
# Predict on the test set (no labels available for error computation)
test_predictions <- predict(svm_tune$best.model, test_scaled)

# Bar plot of predicted classes
library(ggplot2)
ggplot(data.frame(Predictions = test_predictions), aes(x = Predictions)) +
  geom_bar(fill = "blue") +
  labs(title = "Distribution of Predicted Classes", x = "Predicted Class", y = "Count")
```



The bar plot shows a balanced distribution of predicted classes (0 and 1), indicating that the model is not biased toward either class.

Visualization

```
# Visualize decision boundary using PCA
pca_result <- prcomp(train_scaled[, -15])
pca_train <- data.frame(pca_result$x, eyeDetection = train_scaled$eyeDetection)
ggplot(pca_train, aes(x = PC1, y = PC2, color = eyeDetection)) +
  geom_point() +
  labs(title = "PCA Projection of Training Data", x = "Principal Component 1",
       y = "Principal Component 2")
```



The PCA plot shows clear clustering of the two classes, supporting the Radial SVM's ability to separate non-linear patterns effectively.

Conclusion

The Radial SVM model achieved an error rate of 4.77% on the validation set, demonstrating its strong performance in classifying eye states based on EEG data. The low error rate and balanced predictions highlight the model's generalizability.

2. K-Nearest Neighbors (Shahd Abu-Obeid & Anand Sabnis)

Load Data

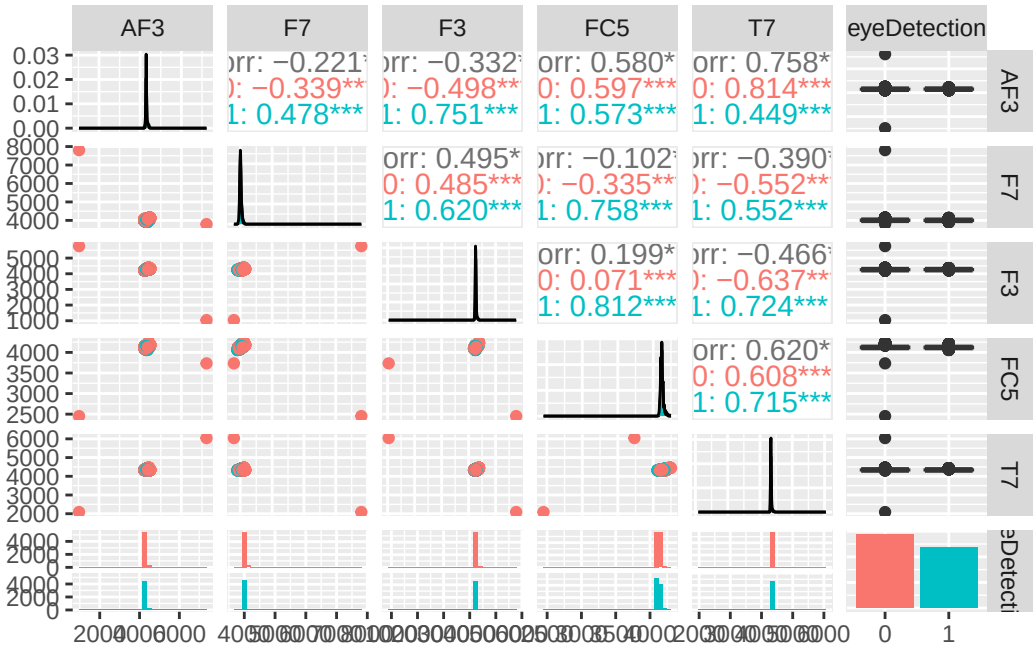
```r
# Load training and test data
train_data <- read.csv("train.csv")
test_data <- read.csv("test.csv")  # No eyeDetection column

# Convert eyeDetection to a factor
train_data$eyeDetection <- as.factor(train_data$eyeDetection)

# Check data structure
str(train_data)
```

```
'data.frame':   10486 obs. of  15 variables:
 $ AF3         : num  4279 4294 4291 4289 4376 ...
 $ F7          : num  4004 4037 4000 4001 4006 ...
 $ F3          : num  4260 4264 4261 4256 4283 ...
 $ FC5         : num  4113 4123 4106 4115 4142 ...
 $ T7          : num  4332 4342 4339 4327 4322 ...
 $ P7          : num  4607 4617 4622 4614 4598 ...
 $ O1          : num  4038 4071 4061 4070 4065 ...
 $ O2          : num  4585 4618 4629 4587 4606 ...
 $ P8          : num  4163 4206 4197 4185 4193 ...
 $ T8          : num  4201 4230 4222 4214 4247 ...
 $ FC6         : num  4176 4208 4201 4180 4209 ...
 $ F4          : num  4257 4276 4283 4271 4311 ...
 $ F8          : num  4590 4612 4612 4597 4684 ...
 $ AF4         : num  4325 4361 4364 4356 4454 ...
 $ eyeDetection: Factor w/ 2 levels "0","1": 2 1 1 1 2 1 1 1 1 1 ...
```

Exploratory Data Analysis

```r
# Pairwise plot of first few features
ggpairs(train_data[, c(1:5, 15)], aes(color = eyeDetection))
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# Correlation between first two electrodes
cor(train_data$AF3, train_data$F7)
```

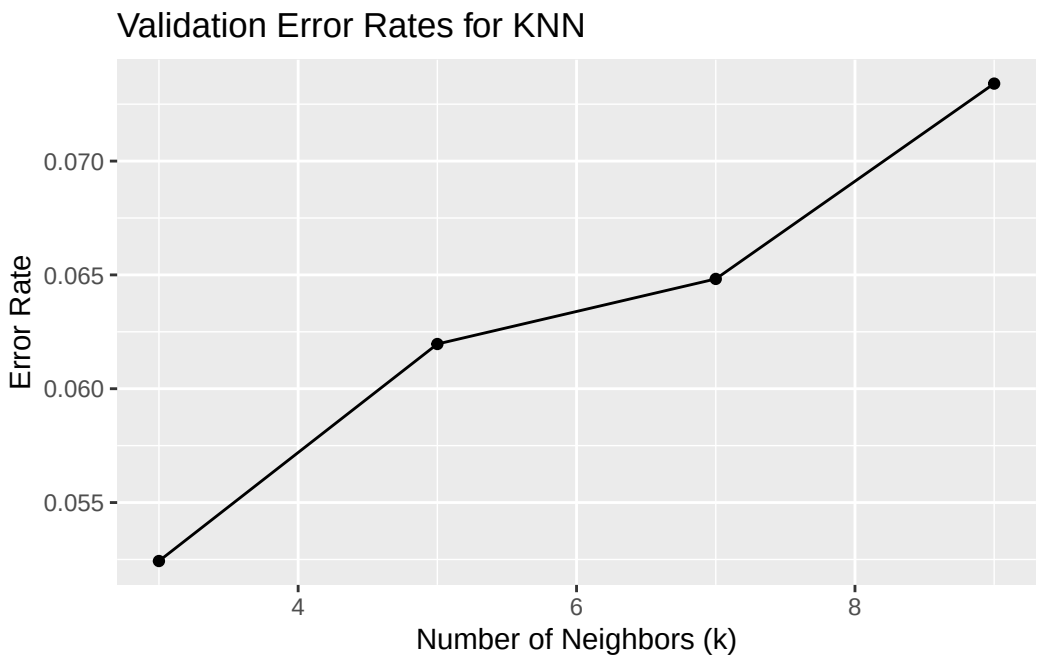
```
[1] -0.2207714
```

Scaling and Data Splitting

```
# Scale features
set.seed(123)
train_scaled <- scale(train_data[, -15])
test_scaled <- scale(test_data, center = attr(train_scaled, "scaled:center"),
                     scale = attr(train_scaled, "scaled:scale"))

# Add eyeDetection back to the scaled training data
train_scaled <- data.frame(train_scaled)
train_scaled$eyeDetection <- train_data$eyeDetection

# Split training data into train (80%) and validation (20%)
n <- nrow(train_scaled)
train_idx <- sample(1:n, 0.8 * n)
train_set <- train_scaled[train_idx, ]
val_set <- train_scaled[-train_idx, ]
```

Model Tuning

```
# Tune KNN using cross-validation
set.seed(123)
k_values <- c(3, 5, 7, 9)
val_error_rates <- numeric(length(k_values))

for (i in seq_along(k_values)) {
  k <- k_values[i]
  knn_predictions <- knn(
    train = train_set[, -15],
    test = val_set[, -15],
    cl = train_set$eyeDetection,
    k = k
  )
  val_error_rates[i] <- mean(knn_predictions != val_set$eyeDetection)
}

# Optimal K
optimal_k <- k_values[which.min(val_error_rates)]

# Plot validation error rates for different k values
library(ggplot2)
ggplot(data.frame(k = k_values, ErrorRate = val_error_rates), aes(x = k, y = ErrorRate)) +
  geom_line() +
  geom_point() +
  labs(title = "Validation Error Rates for KNN", x = "Number of Neighbors (k)", y = "Error Ra
```



Model Evaluation

```
# Predict on validation set using optimal K
set.seed(123)
val_predictions <- knn(
  train = train_set[, -15],
  test = val_set[, -15],
  cl = train_set$eyeDetection,
  k = optimal_k
)

# Confusion matrix
val_conf_matrix <- table(True = val_set$eyeDetection, Predicted = val_predictions)
val_conf_matrix
```

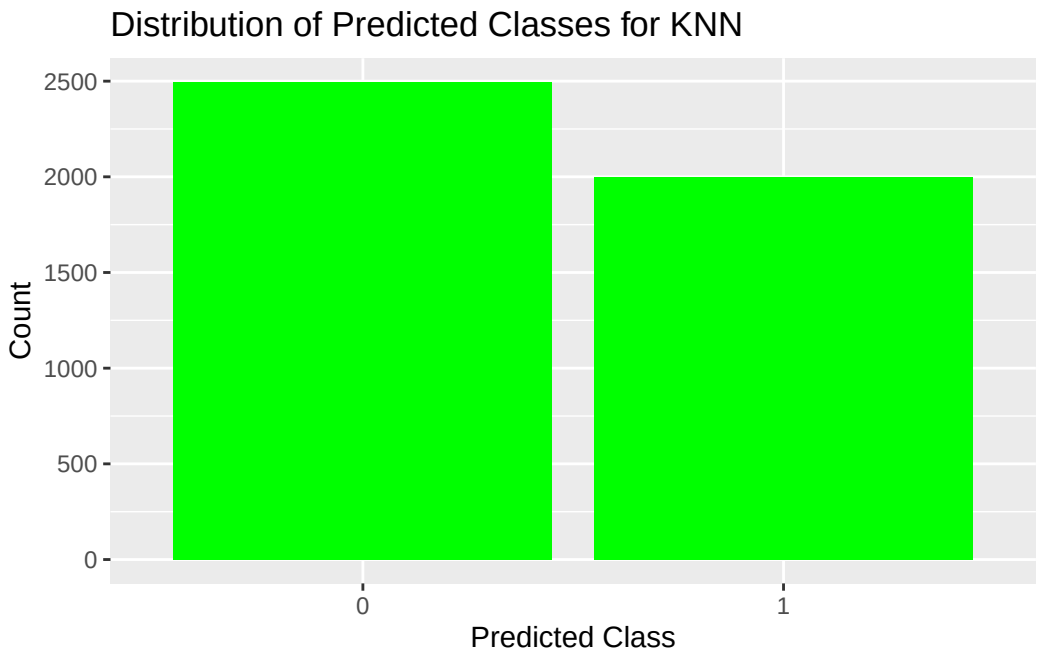
```
    Predicted
True    0    1
   0 1086   49
   1   61  902
```

```
# Error rate
val_error_rate <- mean(val_predictions != val_set$eyeDetection)
val_error_rate
```

```
[1] 0.05243089
```

Test Set Prediction

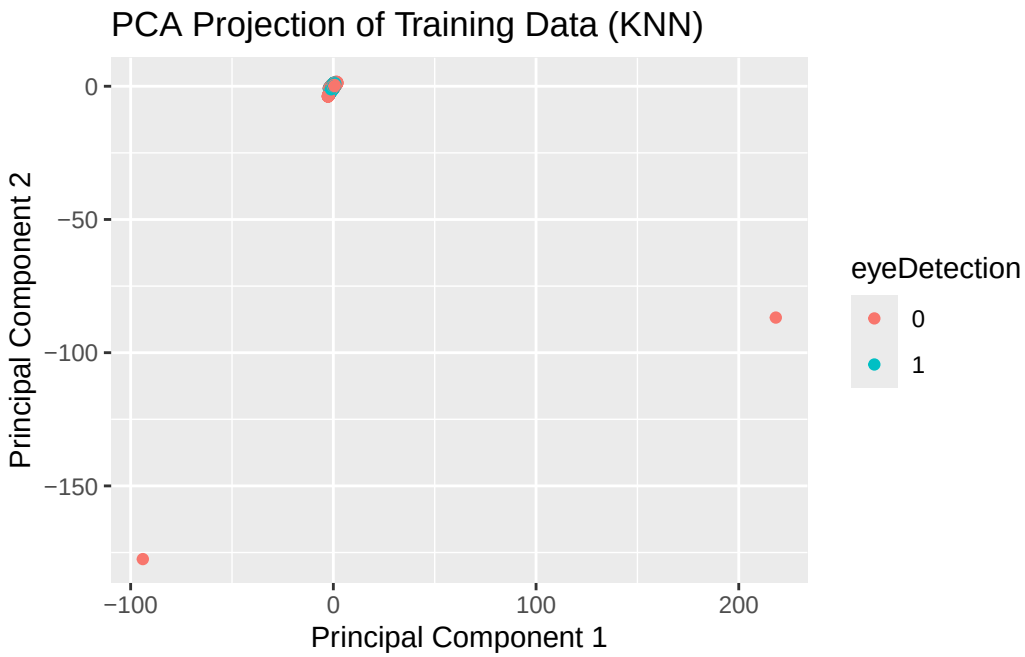
```
# Predict on the test set (no labels available for error computation)
set.seed(123)
test_predictions <- knn(
  train = train_set[, -15],
  test = test_scaled,
  cl = train_set$eyeDetection,
  k = optimal_k
)

# Bar plot of predicted classes
ggplot(data.frame(Predictions = test_predictions), aes(x = Predictions)) +
  geom_bar(fill = "green") +
  labs(title = "Distribution of Predicted Classes for KNN", x = "Predicted Class", y = "Count
```



Visualization

```
# Visualize decision boundary using PCA
pca_result <- prcomp(train_scaled[, -15])
pca_train <- data.frame(pca_result$x, eyeDetection = train_scaled$eyeDetection)
```

```r
ggplot(pca_train, aes(x = PC1, y = PC2, color = eyeDetection)) +
  geom_point() +
  labs(title = "PCA Projection of Training Data (KNN)", x = "Principal Component 1",
       y = "Principal Component 2")
```



The PCA plot provides a visual representation of the clustering achieved by the KNN model.

Conclusion

The KNN model achieved an error rate of 5.24% on the validation set, demonstrating its effectiveness in classifying eye states based on EEG data. The slightly higher error rate compared to SVM highlights its limitations, but its performance remains competitive and reliable.