Spam_Email_Classification

2025-01-16

Installing Required Packages

```r
library(dplyr)
library(tm)
library(SnowballC)
library(caTools)
library(rpart)
library(rpart.plot)
library(randomForest)
library(caret)
library(e1071)
library(ROCR)
```

Downloading the Data

```r
emails = read.csv("/Users/andresperez/Desktop/Personal/PROJECTS/R_Projects/Spam_Email_Cl
assification/emails.csv",stringsAsFactors=FALSE)
```

Part 1: Exploratory Data Analysis & Exploration

Number of Emails by Spam/Non-Spam

```r
# Group emails by spam status and count
spam_counts = emails %>% group_by(spam) %>% summarize(n=n())

print(spam_counts)
```

```
## # A tibble: 5 × 2
##   spam                                                                      n
##   <chr>                                                                 <int>
## 1 ""                                                                        2
## 2 " contended in delhi that since dpc contributed only 0 . 7 per cent of …     1
## 3 " if terminations proceedings go through "                                1
## 4 "0"                                                                    4358
## 5 "1"                                                                    1368
```

There are a total of 5730 emails in this dataset. However, the data must be cleaned so that the `spam` column only contains 0's and 1's and not anything else. This includes removing NA's and non-numeric rows.

```r
# Filter the data to include only rows where spam is "0" or "1"
emails <- emails %>% filter(spam %in% c("0", "1"))

# Verify the cleaned data
spam_counts_cleaned <- emails %>% group_by(spam) %>% summarize(n = n())

print(spam_counts_cleaned)
```

```
## # A tibble: 2 × 2
##   spam      n
##   <chr> <int>
## 1 0      4358
## 2 1      1368
```

Now, there is a total of 5726 emails in our dataset that are labeled as either spam(1) or no spam(0).

First Email Content

```r
# Let's revise the content of the first email
cat("Content of the first email:\n", emails$text[1])
```

```
## Content of the first email:
##  Subject: naturally irresistible your corporate identity  lt is really hard to recoll
ect a company : the  market is full of suqgestions and the information isoverwhelminq ;
but a good  catchy logo , stylish statlonery and outstanding website  will make the task
much easier .  we do not promise that havinq ordered a iogo your  company will automatic
aily become a world ieader : it isguite ciear that  without good products , effective bu
siness organization and practicable aim it  will be hotat nowadays market ; but we do pr
omise that your marketing efforts  will become much more effective . here is the list of
clear  benefits : creativeness : hand - made , original logos , specially done  to refle
ct your distinctive company image . convenience : logo and stationery  are provided in a
ll formats ; easy - to - use content management system letsyou  change your website cont
ent and even its structure . promptness : you  will see logo drafts within three busines
s days . affordability : your  marketing break - through shouldn ' t make gaps in your b
udget . 100 % satisfaction  guaranteed : we provide unlimited amount of changes with no
extra fees for you to  be surethat you will love the result of this collaboration . have
a look at our  portfolio _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ not interested . . . _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

Maximum Number of Characters

```
# Calculate the character count for each email
emails$char.count <- nchar(emails$text)

# Find the maximum character count
max_char_count <- max(emails$char.count)

# Display the maximum character count
cat("Maximum number of characters in an email:", max_char_count, "\n")
```

```
## Maximum number of characters in an email: 31055
```

Row of Email with Most Characters

```
# Find the row with the maximum character count
max_char_row <- which.max(emails$char.count)

# Display the row number and corresponding character count
cat("Row of the email with the most characters:", max_char_row, "\n")
```

```
## Row of the email with the most characters: 2338
```

```
cat("Character count in this email:", emails$char.count[max_char_row], "\n")
```

```
## Character count in this email: 31055
```

```
# Display a summary of character counts
cat("Summary of character counts:\n")
```

```
## Summary of character counts:
```

```
print(summary(emails$char.count))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    13.0   508.2   979.0  1542.0  1891.8 31055.0
```

Minimum Number of Characters

```
# Find the minimum character count
min_char_count <- min(emails$char.count)

# Display the minimum character count
cat("Minimum number of characters in an email:", min_char_count, "\n")
```

```
## Minimum number of characters in an email: 13
```

Row of Email with Least Characters

```
# Find the row with the minimum character count
min_char_row <- which.min(emails$char.count)

# Display the row number and corresponding character count
cat("Row of the email with the least characters:", min_char_row, "\n")
```

```
## Row of the email with the least characters: 1991
```

```
cat("Character count in this email:", emails$char.count[min_char_row], "\n")
```

```
## Character count in this email: 13
```

Document - Term Matrix (DTM)

Preprocess Text Data

```
# Create a text corpus and preprocess the data
corpus <- VCorpus(VectorSource(emails$text))
corpus <- tm_map(corpus, content_transformer(tolower)) # Convert to lowercase
corpus <- tm_map(corpus, removePunctuation)          # Remove punctuation
corpus <- tm_map(corpus, removeWords, stopwords("english")) # Remove stopwords
corpus <- tm_map(corpus, stemDocument)               # Perform stemming
cat("Text preprocessing completed.\n")
```

```
## Text preprocessing completed.
```

Create DTM

```
# Create the document-term matrix (DTM)
dtm <- DocumentTermMatrix(corpus)

# Display the structure of the DTM
cat("Document-Term Matrix (DTM):\n")
```

```
## Document-Term Matrix (DTM):
```

```
print(dtm)
```

```
## <<DocumentTermMatrix (documents: 5726, terms: 28607)>>
## Non-/sparse entries: 479800/163323882
## Sparsity           : 100%
## Maximal term length: 24
## Weighting          : term frequency (tf)
```

Number of Stepwords

```
# Display the number of stopwords used
cat("Number of stopwords in English:", length(stopwords("english")), "\n")
```

```
## Number of stopwords in English: 174
```

Sparse DTM

Reduce Sparse Terms

```
# Create a sparse DTM by removing terms with low frequency
spdtm <- removeSparseTerms(dtm, 0.995)

# Display the structure of the sparse DTM
cat("Sparse Document-Term Matrix (spDTM):\n")
```

```
## Sparse Document-Term Matrix (spDTM):
```

```
print(spdtm)
```

```
## <<DocumentTermMatrix (documents: 5726, terms: 2320)>>
## Non-/sparse entries: 392959/12891361
## Sparsity           : 97%
## Maximal term length: 15
## Weighting          : term frequency (tf)
```

Word Frequency Analysis

Most Frequent Word Stem

```
# Convert the sparse DTM to a data frame
emailsSparse <- as.data.frame(as.matrix(spdtm))
colnames(emailsSparse) <- make.names(colnames(emailsSparse))

# Find the most frequent word stem
stem_freq_max <- names(which.max(colSums(emailsSparse)))

# Display the most frequent word stem
cat("Most frequent word stem:", stem_freq_max, "\n")
```

```
## Most frequent word stem: enron
```

Word Stem in Non-Spam Emails (≥ 5000 Occurrences)

```
# Add the spam column to emailsSparse
emailsSparse$spam <- as.numeric(emails$spam)
emailsSparse <- emailsSparse[!is.na(emailsSparse$spam), ]

# Count word stems appearing at least 5000 times in non-spam emails
nonspam_word_count <- sum(
  colSums(emailsSparse[emailsSparse$spam == 0, sapply(emailsSparse, is.numeric)]) >= 500
0
)

# Display the count
cat("Number of word stems appearing at least 5000 times in non-spam emails:", nonspam_wo
rd_count, "\n")
```

```
## Number of word stems appearing at least 5000 times in non-spam emails: 6
```

Word Stems in Spam Emails (≥ 1000 Occurrences)

```
# Count word stems appearing at least 1000 times in spam emails
spam_word_count <- sum(
  colSums(emailsSparse[emailsSparse$spam == 1, sapply(emailsSparse, is.numeric)]) >= 100
0
)

# Display the count
cat("Number of word stems appearing at least 1000 times in spam emails:", spam_word_coun
t, "\n")
```

```
## Number of word stems appearing at least 1000 times in spam emails: 4
```

Part 2: Model Building & Evaluation

Split the Data into Training and Testing Sets

```
# Create a 70/30 split using caret
set.seed(123)
train_index <- createDataPartition(emailsSparse$spam, p = 0.7, list = FALSE)
train_data <- emailsSparse[train_index, ]
test_data <- emailsSparse[-train_index, ]

# Check split sizes
cat("Training Set Size:", nrow(train_data), "\n")
```

```
## Training Set Size: 4009
```

```
cat("Testing Set Size:", nrow(test_data), "\n")
```

```
## Testing Set Size: 1717
```

Logistic Regression

Train the Model

```
# Train Logistic Regression
spamLog = glm(spam~.,data=train_data, family=binomial)
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

Evaluate on the Training Set

```
# Predict probabilities on the training set
log_pred_train <- predict(spamLog, type = "response")

# Calculate accuracy with a threshold of 0.5
log_train_accuracy <- mean((log_pred_train >= 0.5) == train_data$spam)

# Print accuracy
cat("Logistic Regression Training Accuracy:", log_train_accuracy, "\n")
```

```
## Logistic Regression Training Accuracy: 0.9995011
```

Evaluate on the Test Set

```
# Predict probabilities on the test set
log_pred_test <- predict(spamLog, newdata = test_data, type = "response")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases
```

```
# Calculate accuracy with a threshold of 0.5
log_test_accuracy <- mean((log_pred_test >= 0.5) == test_data$spam)

# Print accuracy
cat("Logistic Regression Test Accuracy:", log_test_accuracy, "\n")
```

```
## Logistic Regression Test Accuracy: 0.8124636
```

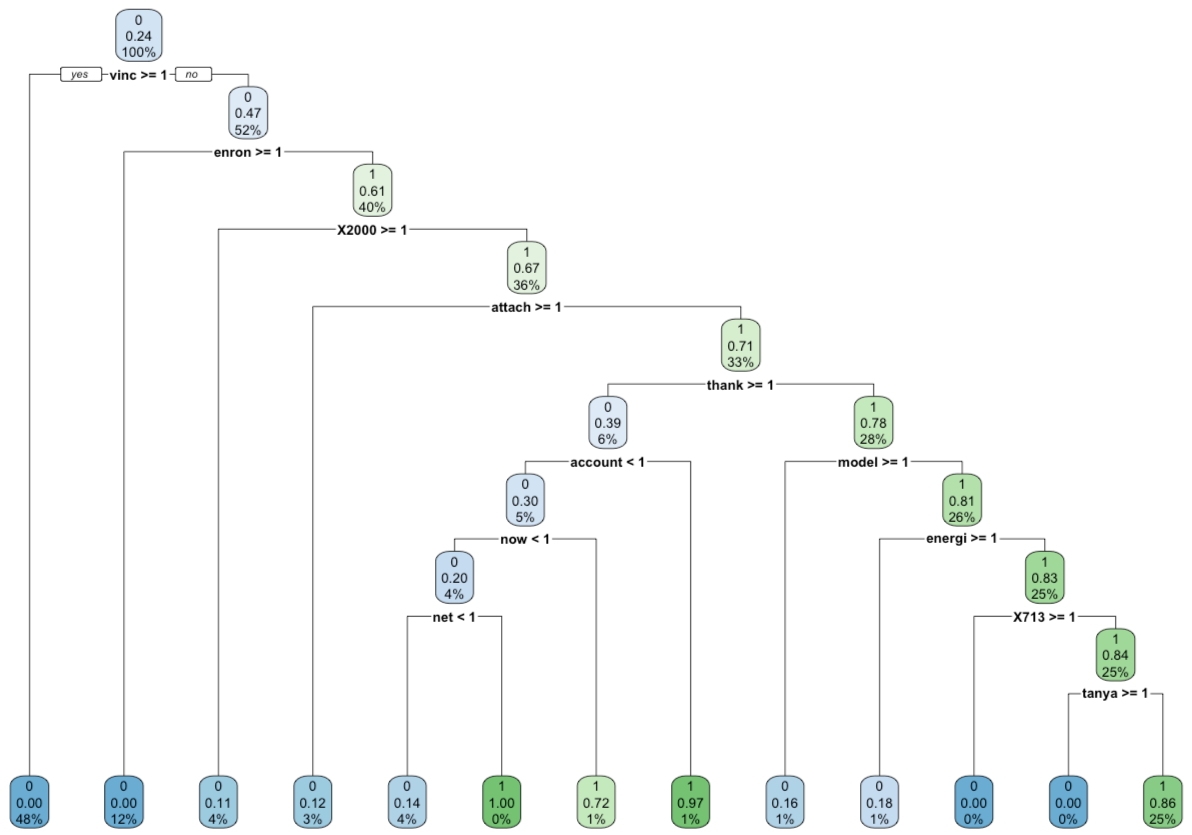
CART (Decision Tree)

Train the Model

```
spam_cart <- rpart(spam ~ ., data = train_data, method = "class")
```

Visualizing the Decision Tree

```
rpart.plot(spam_cart)
```



Evaluate on the Train Set

```
# Predict probabilities
cart_pred_train <- predict(spam_cart, type = "prob")[, 2]
# Calculate accuracy
cart_train_accuracy <- mean((cart_pred_train >= 0.5) == train_data$spam)
cat("CART Training Accuracy:", cart_train_accuracy, "\n")
```

```
## CART Training Accuracy: 0.9446246
```

Evaluate on the Test Set

```
# Predict probabilities
cart_pred_test <- predict(spam_cart, newdata = test_data, type = "prob")[, 2]
# Calculate accuracy
cart_test_accuracy <- mean((cart_pred_test >= 0.5) == test_data$spam)
cat("CART Test Accuracy:", cart_test_accuracy, "\n")
```

```
## CART Test Accuracy: 0.9405941
```

Random Forest

Train the Model

```
train_data$spam <- as.factor(train_data$spam)
test_data$spam <- as.factor(test_data$spam)

set.seed(123)
spam_rf <- randomForest(spam ~ ., data = train_data)
```

Evaluate on the Train & Test Set

```
# Predict on training set
rf_pred_train <- predict(spam_rf, type = "response")

# Predict on testing set
rf_pred_test <- predict(spam_rf, newdata = test_data, type = "response")
```

Training Accuracy

```
# Calculate training accuracy
rf_train_accuracy <- mean(rf_pred_train == train_data$spam)
cat("Random Forest Training Accuracy:", rf_train_accuracy, "\n")
```

```
## Random Forest Training Accuracy: 0.9855326
```

Testing Accuracy

```
# Calculate testing accuracy
rf_test_accuracy <- mean(rf_pred_test == test_data$spam)
cat("Random Forest Test Accuracy:", rf_test_accuracy, "\n")
```

```
## Random Forest Test Accuracy: 0.987187
```

Confusion Matrix

```
# Confusion matrix for testing set
conf_matrix <- confusionMatrix(rf_pred_test, test_data$spam)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1321   15
##          1    7  374
##
##                Accuracy : 0.9872
##                  95% CI : (0.9807, 0.992)
##     No Information Rate : 0.7734
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9632
##
##  Mcnemar's Test P-Value : 0.1356
##
##             Sensitivity : 0.9947
##             Specificity : 0.9614
##          Pos Pred Value : 0.9888
##          Neg Pred Value : 0.9816
##              Prevalence : 0.7734
##          Detection Rate : 0.7694
##    Detection Prevalence : 0.7781
##       Balanced Accuracy : 0.9781
##
##        'Positive' Class : 0
##
```

Calculate AUC for Each Model

Logistic Regression

```
# Get probabilities for the positive class (e.g., "1")
log_prob_test <- predict(spamLog, newdata = test_data, type = "response")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases
```

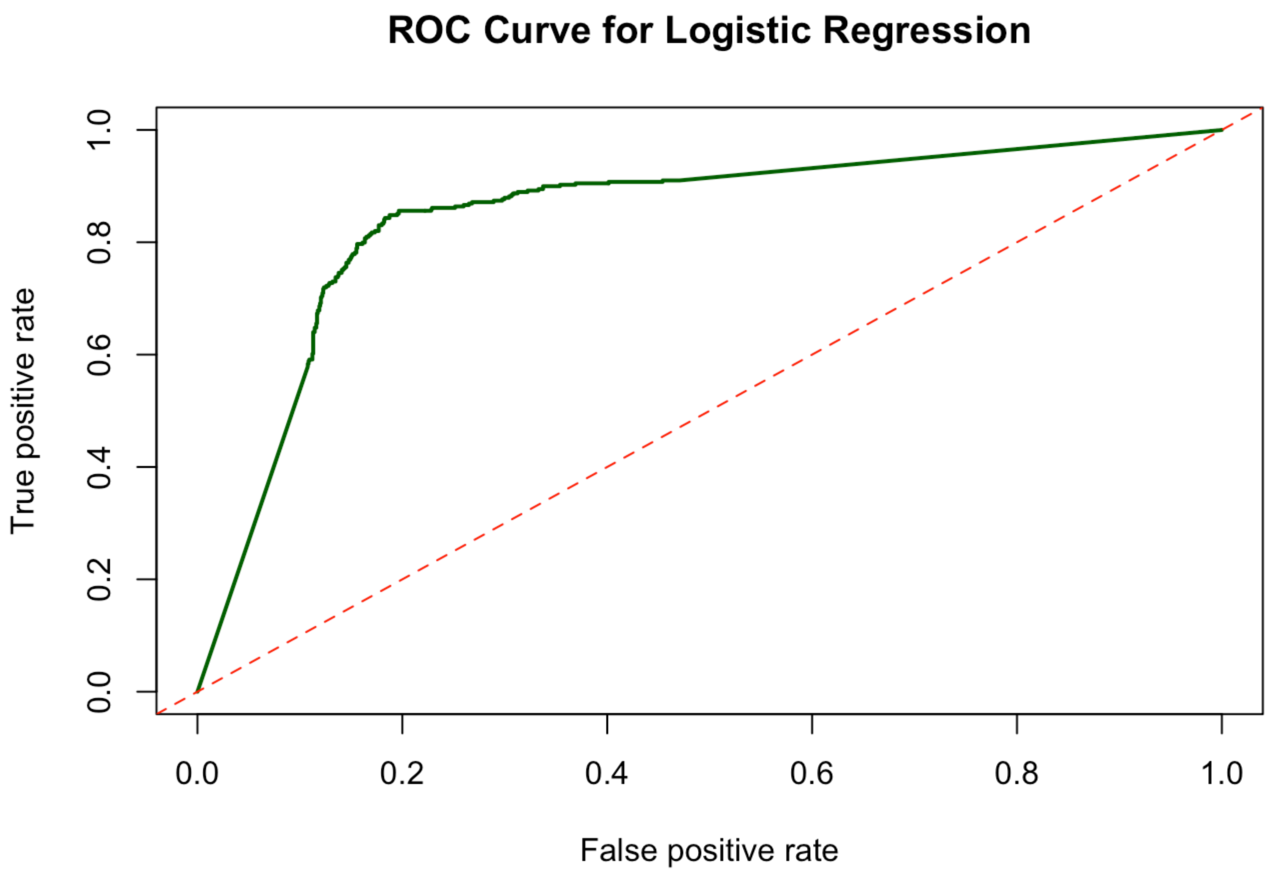
```
# Ensure Test$spam is numeric
test_data$spam <- as.numeric(as.character(test_data$spam))

# Create prediction object
spamLog.predictionTest <- prediction(as.numeric(log_prob_test), test_data$spam)

# Calculate AUC
spamLog.auc <- as.numeric(performance(spamLog.predictionTest, "auc")@y.values)
cat("Logistic Regression Test AUC:", spamLog.auc, "\n")
```

```
## Logistic Regression Test AUC: 0.8479864
```

```
# Plot the ROC curve
spamLog.perf <- performance(spamLog.predictionTest, "tpr", "fpr")
plot(spamLog.perf, col = "darkgreen", main = "ROC Curve for Logistic Regression", lwd =
2)
abline(a = 0, b = 1, lty = 2, col = "red")  # Add a diagonal line
```

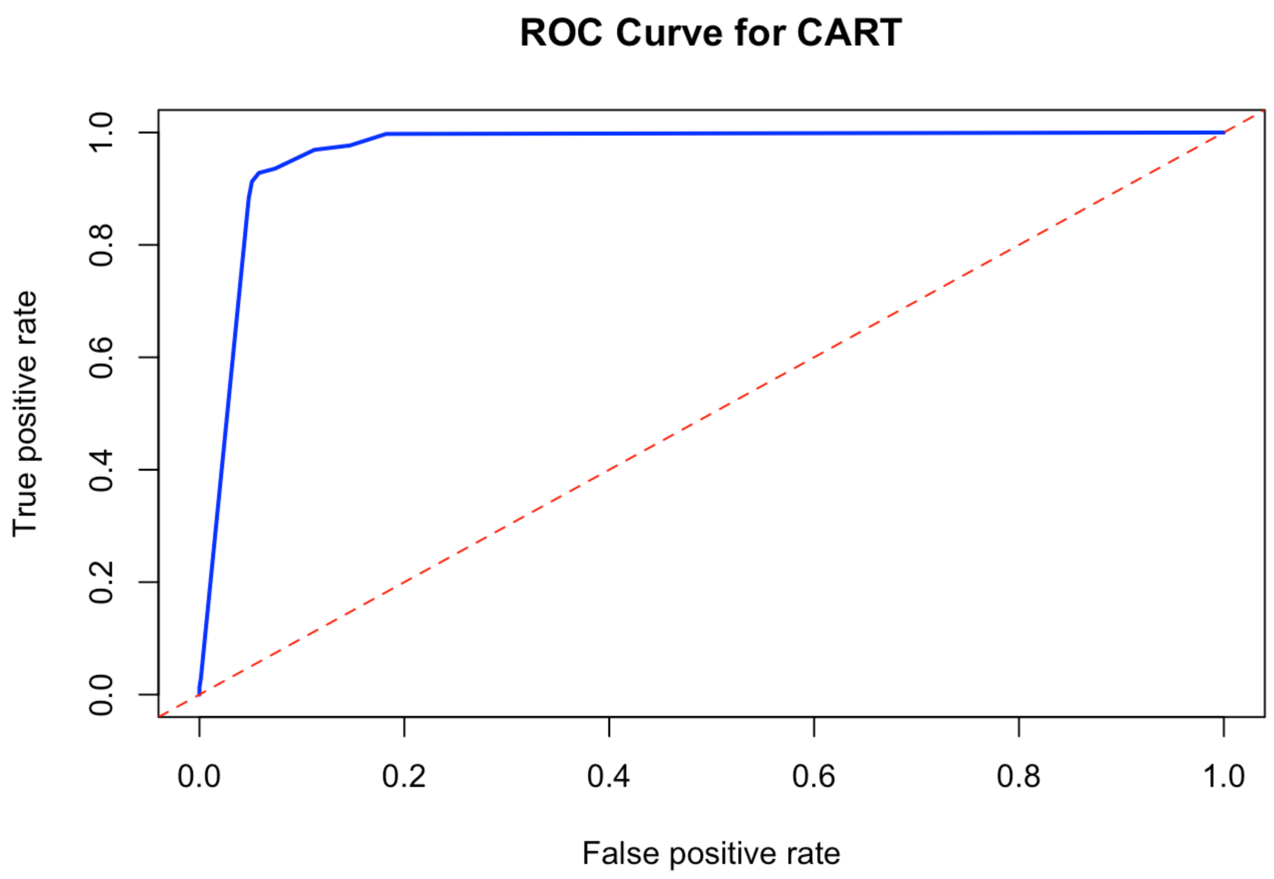


CART

```
# Get probabilities for the positive class (e.g., "1")
cart_prob_test <- predict(spam_cart, newdata = test_data, type = "prob")[, 2]

# Create prediction and performance objects
cart_pred_obj <- prediction(cart_prob_test, test_data$spam)
cart_perf <- performance(cart_pred_obj, "tpr", "fpr")

# Plot the ROC curve
plot(cart_perf, col = "blue", main = "ROC Curve for CART", lwd = 2)
abline(a = 0, b = 1, lty = 2, col = "red")  # Add a diagonal line
```



```
# Calculate AUC
cart_auc <- as.numeric(performance(cart_pred_obj, "auc")@y.values)
cat("CART Test AUC:", cart_auc, "\n")
```

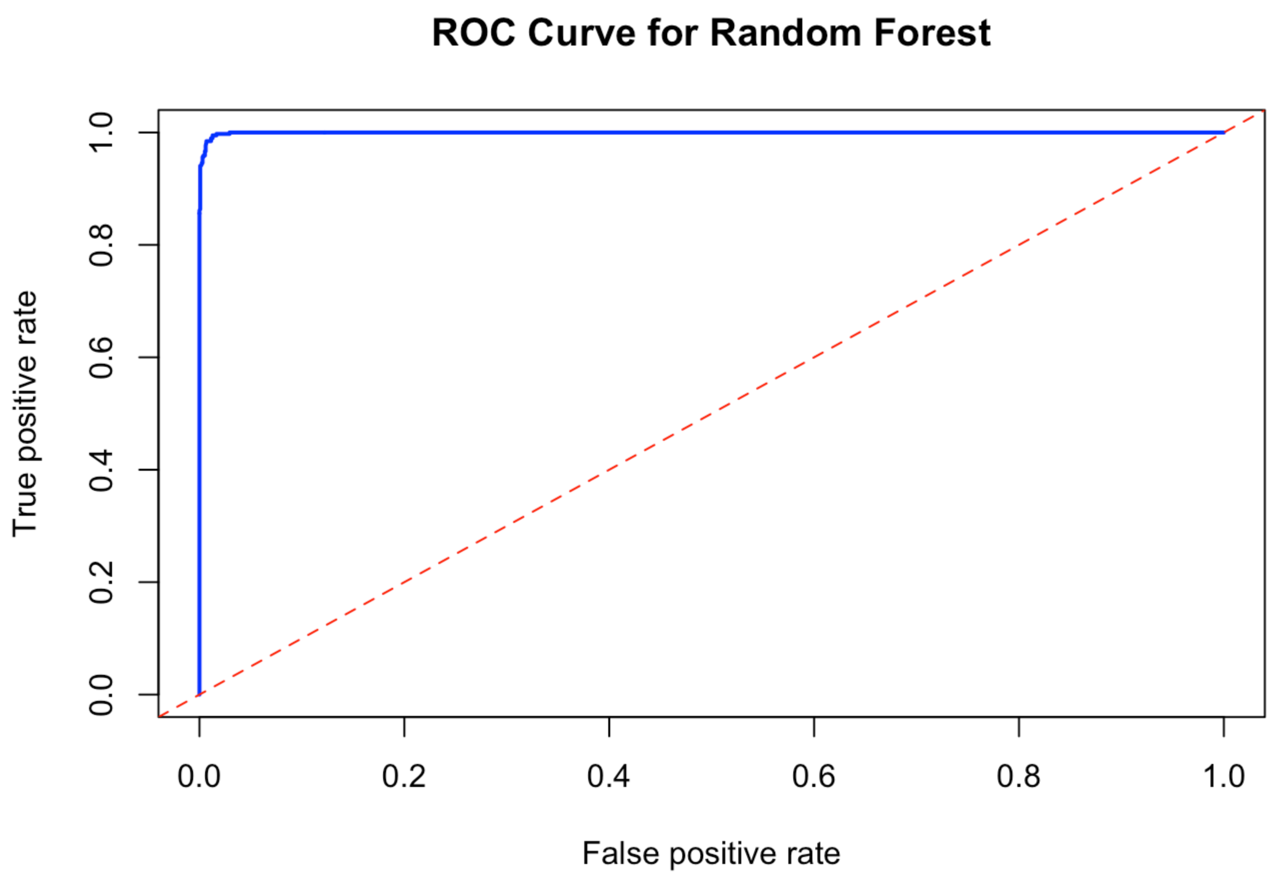
```
## CART Test AUC: 0.9670127
```

Random Forest

```
# Get probabilities for the positive class (e.g., "1")
rf_prob_test <- predict(spam_rf, newdata = test_data, type = "prob")[, 2]

# Create prediction and performance objects
rf_pred_obj <- prediction(rf_prob_test, test_data$spam)
rf_perf <- performance(rf_pred_obj, "tpr", "fpr")

# Plot the ROC curve
plot(rf_perf, col = "blue", main = "ROC Curve for Random Forest", lwd = 2)
abline(a = 0, b = 1, lty = 2, col = "red")  # Add a diagonal line
```



```
# Calculate AUC
rf_auc <- as.numeric(performance(rf_pred_obj, "auc")@y.values)
cat("Random Forest Test AUC:", rf_auc, "\n")
```

```
## Random Forest Test AUC: 0.9994948
```

Compare Model Performance - Accuracy

```
# Logistic Regression Accuracy
log_accuracy <- mean((log_prob_test >= 0.5) == test_data$spam)

# CART Accuracy
cart_accuracy <- mean((cart_prob_test >= 0.5) == test_data$spam)

# Random Forest Accuracy
rf_accuracy <- mean((rf_prob_test >= 0.5) == test_data$spam)
```

Compare Model Performance - AUC

```
# Logistic Regression AUC
spamLog.auc  # From previous calculation
```

```
## [1] 0.8479864
```

```
# CART AUC
cart_auc  # From previous calculation
```

```
## [1] 0.9670127
```

```
# Random Forest AUC
rf_auc  # From previous calculation
```

```
## [1] 0.9994948
```

Summary Table

```
results <- data.frame(
  Model = c("Logistic Regression", "CART", "Random Forest"),
  Accuracy = c(log_accuracy, cart_accuracy, rf_accuracy),
  AUC = c(spamLog.auc, cart_auc, rf_auc)
)
print(results)
```

```
##                 Model  Accuracy       AUC
## 1 Logistic Regression 0.8124636 0.8479864
## 2                CART 0.9405941 0.9670127
## 3       Random Forest 0.9877694 0.9994948
```

Conclusion

In this analysis, three models—Logistic Regression, CART, and Random Forest—were trained and evaluated to classify emails as spam or non-spam. The performance of each model was assessed using two key metrics: **Accuracy** and **AUC (Area Under the ROC Curve).**

Model Performance Overview

1. **Logistic Regression**:
    - Accuracy: **81.25%**
    - AUC: **0.848**
    - As a simple linear model, Logistic Regression performed reasonably well. Its ability to provide interpretable results makes it a viable choice for tasks where simplicity and interpretability are prioritized. However, its lower accuracy and AUC indicate it struggles with capturing complex relationships in the dataset.
2. **CART (Classification and Regression Trees)**:

- Accuracy: **94.06%**
- AUC: **0.967**
- CART significantly outperformed Logistic Regression in both accuracy and AUC. Its interpretability—through decision trees—is a major strength, allowing easy understanding of the classification rules. However, decision trees are prone to overfitting, which can reduce generalization ability on unseen data.

3. **Random Forest**:
    - Accuracy: **98.78%**
    - AUC: **0.999**
    - Random Forest emerged as the best-performing model, achieving the highest accuracy and AUC. Its ensemble approach effectively handles complex relationships and reduces overfitting compared to a single decision tree. This makes it a robust choice for classification tasks requiring high predictive power.

Comparison and Final Recommendation

The **Random Forest** model demonstrated exceptional performance with an accuracy of **98.78%** and an AUC of **0.999**, surpassing both Logistic Regression and CART in all evaluated metrics. CART also performed well, particularly in terms of interpretability, while Logistic Regression offered a simpler alternative with moderate performance.

Given the results, **Random Forest is recommended** for spam email classification due to its superior predictive power and ability to handle complex patterns in the dataset. While it is computationally more intensive, its accuracy and AUC justify its use for this task. For scenarios where interpretability is key, CART may serve as a strong secondary option. Logistic Regression, while the simplest, is not ideal for this dataset due to its comparatively lower performance.