The Final Analysis of the SMS Spam Collection Dataset

Eleni Baltzi

April 9, 2025

A. Introduction

The instructions provided for the RDA analysis of the iris dataset [1] as well as the instructions for the boostrap evaluation were adapted in order to perform a similar analysis for the SMS Spam Collection dataset [2]. Then, a logistic regression and finally a decision tree were built. A performance comparison between all 7 classification methods was then conducted utilizing confusion matrices in order to assess the most reliable method for classifying SMS messages into *ham* and *spam*.

B. RDA Analysis

Loading Required Packages

To perform data processing and analysis, the following R packages are used (Listing 1):

```r
# Load required libraries
library(klaR)       # For Regularized Discriminant Analysis
library(caret)      # For data partitioning, preprocessing, and
   evaluation
library(dplyr)      # For data manipulation
library(tm)         # For text mining
```

Listing 1: Code block for loading the initial required packages

Loading the Dataset

The dataset is read from a CSV file and stored in a dataframe. The column names are also defined explicitly(Listing 2):

```r
# Load the dataset and rename columns
sms_data <- read.csv("/home/fenrir/T l chargements/sms+spam+
   collection/SMSSpamCollection.csv", stringsAsFactors = FALSE)
colnames(sms_data) <- c("label", "text")
sms_data$label <- factor(sms_data$label, levels = c("ham", "spam"))  #
   Convert to factor
```

Listing 2: Code block for loading the dataset and Converting labels to factors

The dataset consists of two columns:

- **label**: This column indicates whether an SMS message is spam or not. The values are either `ham` (not spam) or `spam` (spam).
- **text**: This column contains the content of the SMS message.

The output of the script confirms that the dataset consists of only two unique labels: `"ham"` and `"spam"`. This means the dataset does not contain any additional categories, making it a binary classification problem.

```r
# Split the data into Train (60%), Validation (20%), and Test (20%) sets
set.seed(123)  # For reproducibility
train_index <- createDataPartition(sms_data$label, p = 0.6, list = FALSE)
train_data <- sms_data[train_index, ]
remaining <- sms_data[-train_index, ]
val_index <- createDataPartition(remaining$label, p = 0.5, list = FALSE)
val_data <- remaining[val_index, ]
test_data <- remaining[-val_index, ]
cat("Train:", nrow(train_data), "| Val:", nrow(val_data), "| Test:", nrow(test_data), "\n")
```

Listing 3: Split the data into Train, Validation , and Test sets

The dataset was partitioned into three subsets(Listing 3):

- **Training set (60%)**: Used to train the model and fit the underlying parameters.
- **Validation set (20%)**: Used during model development to tune hyperparameters and compare model variants without biasing the final evaluation.
- **Test set (20%)**: Held out until final evaluation to assess the model's generalization performance on previously unseen data.

This three-way split ensures that model selection and final evaluation are performed fairly and independently. The dataset contains a total of 5,533 SMS messages, which were divided as follows:

- **Training set:** 3,320 samples
- **Validation set:** 1,107 samples
- **Test set:** 1,106 samples

Creating a Corpus and Document-Term Matrix

Before modeling, the raw SMS messages were preprocessed using a custom text cleaning function. The following steps were applied to standardize the text and reduce noise:

1. **Lowercasing:** All characters were converted to lowercase to avoid case-sensitive duplication (e.g., "Free" vs. "free").
2. **Punctuation Removal:** All punctuation marks were removed.
3. **Number Removal:** All numerical digits were eliminated from the text.
4. **Stopword Removal:** Common English stopwords (e.g., "the", "and", "is") were removed using the `tm` package's predefined list.
5. **Whitespace Stripping:** Redundant white spaces were removed to normalize text spacing.

These preprocessing steps help reduce dimensionality, standardize the input, and retain only the most meaningful tokens for text classification tasks.(Listing 4):

```r
library(tm)
#Create a corpus from the text data
corpus <- Corpus(VectorSource(sms_data$text))

#Preprocess the corpus - converting to lowercase, removing punctuation,
    numbers, stopwords, and whitespace.

corpus <- tm_map(corpus, content_transformer(tolower))
corpus <- tm_map(corpus, removePunctuation)
corpus <- tm_map(corpus, removeNumbers)
corpus <- tm_map(corpus, removeWords, stopwords("en"))
corpus <- tm_map(corpus, stripWhitespace)

#Create a Document-Term Matrix
dtm <- DocumentTermMatrix(corpus)

#Check the DTM structure
print(dtm)
```

Listing 4: Creating a corpus from the text data

In contrast to the preprocessing for the LDA and QDA analysis, the text corpus and the corresponding Document-Term Matrix (DTM) (Listing 4) were created exclusively from the training data, not from the entire dataset. Only words that appeared in more than a threshold number of messages (e.g., at least 5) were retained in the training set. The same vocabulary (set of terms) extracted from the training data was then applied to construct the validation and test document-term matrices. This ensures consistency in feature representation across all subsets. This difference in handling the preprocessing of the data was done to reduce dimensionality and eliminate rare or uninformative terms and avoid information leakage from the validation or test sets into the feature extraction process, thereby ensuring that the model is evaluated on truly unseen data. When the text corpus and the corresponding DTM were created from the entire dataset the model training kept running into errors. (Listing 5).

```r
# Build DTM for Train
corpus_train <- clean_corpus(train_data)
dtm_train <- DocumentTermMatrix(corpus_train)
dtm_train <- removeSparseTerms(dtm_train, 0.99)

# Apply same vocabulary to val/test
corpus_val <- clean_corpus(val_data)
dtm_val <- DocumentTermMatrix(corpus_val, control = list(dictionary =
   Terms(dtm_train)))

corpus_test <- clean_corpus(test_data)
dtm_test <- DocumentTermMatrix(corpus_test, control = list(dictionary =
    Terms(dtm_train)))

# Convert to data frames and add labels
train_matrix <- as.data.frame(as.matrix(dtm_train))
train_matrix$label <- train_data$label
```

```r

val_matrix <- as.data.frame(as.matrix(dtm_val))
val_matrix$label <- val_data$label

test_matrix <- as.data.frame(as.matrix(dtm_test))
test_matrix$label <- test_data$label
```

Listing 5: Building t DTM

Similarly to the LDA and QDA analysis, after constructing the DTM for the validation and test sets using the vocabulary derived from the training data, some messages contained none of the retained terms. These resulted in empty rows in the DTM, which lead to issues during model prediction.

To address this, all empty documents (i.e., rows with a term frequency sum of zero) were identified and removed from the validation and test matrices. This step ensures that only meaningful, non-empty feature vectors are passed to the models during evaluation(Listing 6).

```r
# Function to remove empty rows after DTM creation
remove_empty_docs <- function(mat, labels) {
  row_sums <- rowSums(as.matrix(mat))
  non_empty <- row_sums > 0
  list(matrix = as.data.frame(as.matrix(mat[non_empty, ])),
       labels = labels[non_empty])
}

# Apply to val
val_clean <- remove_empty_docs(dtm_val, val_data$label)
val_matrix <- val_clean$matrix
val_matrix$label <- val_clean$labels

# Apply to test
test_clean <- remove_empty_docs(dtm_test, test_data$label)
test_matrix <- test_clean$matrix
test_matrix$label <- test_clean$labels
```

Listing 6: Finding the empty documents

Preparing Text Data for Modeling

Features with near-zero variance across the training set were removed prior to model training. These features occur with almost the same frequency in most samples and therefore contribute little to the classification task. Removing them helps reduce dimensionality, improve computational efficiency, and minimize the risk of overfitting.

This was implemented using the `nearZeroVar()` function from the `caret` package, which identifies predictors that have low variability and are thus unlikely to be informative.(Listing 7):

```r
# Remove features with near-zero variance
nzv <- nearZeroVar(train_matrix, saveMetrics = TRUE)
train_matrix <- train_matrix[, !nzv$zeroVar]
```

Listing 7: Removing features with near-zero variance

To further reduce redundancy and prevent multicollinearity, highly correlated features were identified and removed. Specifically, the pairwise Pearson correlation matrix of numeric predictors was computed, and features with correlation coefficients above 0.99 were excluded using the `findCorrelation()` function from the `caret` package.

Following feature selection, the remaining predictors were standardized using centering and scaling. This transformation ensures that all features contribute equally to the model, regardless of their original scales. The preprocessing was applied using the `preProcess()` function, with both centering (mean subtraction) and scaling (division by standard deviation)8.

```r
# Remove highly correlated features to avoid multicollinearity
cor_matrix <- cor(train_matrix[, sapply(train_matrix, is.numeric)])
high_corr <- findCorrelation(cor_matrix, cutoff = 0.99)
if (length(high_corr) > 0) {
  train_matrix <- train_matrix[, -high_corr]
}

# Standardize the training features
preproc <- preProcess(train_matrix[, -ncol(train_matrix)], method = c("
    center", "scale"))
train_scaled <- predict(preproc, train_matrix)
train_scaled$label <- train_matrix$label
```

Listing 8: Remove highly correlated features to avoid multicollinearity and Standardize the training features

Training the RDA Model

Adapted from the provided instructions the RDA model was trained using a loop. (Listing 9) To identify the optimal hyperparameters for the Regularized Discriminant Analysis (RDA) model, a grid search was conducted over a predefined parameter space. Specifically, the regularization parameters γ and λ were varied across the ranges $[0, 1]$ and $[0.1, 1]$ respectively, with increments of 0.2. This resulted in a grid of candidate (γ, λ) pairs.

Model selection was performed using 10-fold stratified cross-validation on the training set. For each fold, the model was trained on 90% of the data and evaluated on the remaining 10%. The classification accuracy was recorded for each hyperparameter combination across all folds.

This procedure provides a reliable estimate of model generalizability and helps avoid overfitting by ensuring that the selected parameters perform well across multiple data partitions.

```r
# Define grid for hyperparameter tuning (gamma and lambda)
gamma_grid <- seq(0, 1, by = 0.2)
lambda_grid <- seq(0.1, 1, by = 0.2)
grid <- expand.grid(gamma = gamma_grid, lambda = lambda_grid)

# Create 10 stratified folds for cross-validation
set.seed(321)
folds <- createFolds(train_scaled$label, k = 10)
cv_results <- matrix(0, nrow = nrow(grid), ncol = 10)

# Perform cross-validation over the grid
for (fold_idx in 1:10) {
  fold_valid_idx <- folds[[fold_idx]]
```

```r
  fold_valid <- train_scaled[fold_valid_idx, ]
  fold_train <- train_scaled[-fold_valid_idx, ]

  fold_train$label <- factor(fold_train$label, levels = c("ham", "spam"))
  fold_valid$label <- factor(fold_valid$label, levels = c("ham", "spam"))

  for (i in 1:nrow(grid)) {
    g <- grid$gamma[i]
    l <- grid$lambda[i]
    model <- rda(label ~ ., data = fold_train, gamma = g, lambda = l)
   # Train RDA
    preds <- predict(model, fold_valid, type = "class")
   # Predict
    acc <- mean(preds$class == fold_valid$label)
   # Accuracy
    cv_results[i, fold_idx] <- acc
   # Store result
  }
}
```

Listing 9: Regularized Discriminant Analysis (RDA)

RDA Analysis Output

The selected RDA model was trained with regularization parameters $\gamma = 1.0$ and $\lambda = 0.9$, based on cross-validation performance.

- **Prior probabilities:** The model estimated class priors as 86.7% for `ham` and 13.3% for `spam`, probably due to class imbalance present in the dataset.
- **Apparent misclassification rate:** The model achieved an apparent error rate of 8.568% on the training subset, which indicates that approximately 91.4% of training messages were correctly classified. This metric should be interpreted with caution, as it does not reflect generalization to unseen data.
- **Regularization:** The high value of $\gamma = 1.0$ indicates that the model is heavily biased toward a quadratic structure (akin to QDA), while the $\lambda = 0.9$ adds substantial regularization to stabilize covariance estimates and reduce overfitting.

Overall, the model demonstrates good fit on the training data while incorporating regularization to improve stability and performance on new data.

Re-training the RDA Model

As proposed in the instructions the RDA model was retrained (Figure 1) after performing cross-validation and the optimal hyperparameter pair (γ, λ) was selected as the one that maximized the average classification accuracy across the 10 folds, therefore using these best-performing values (Listing 10).

To ensure consistent feature scaling during model evaluation, the same centering and scaling parameters derived from the training data were applied to the validation and test sets. This guaranteed that all subsets were represented in the same feature space, preventing any distributional shift that could bias model performance metrics.

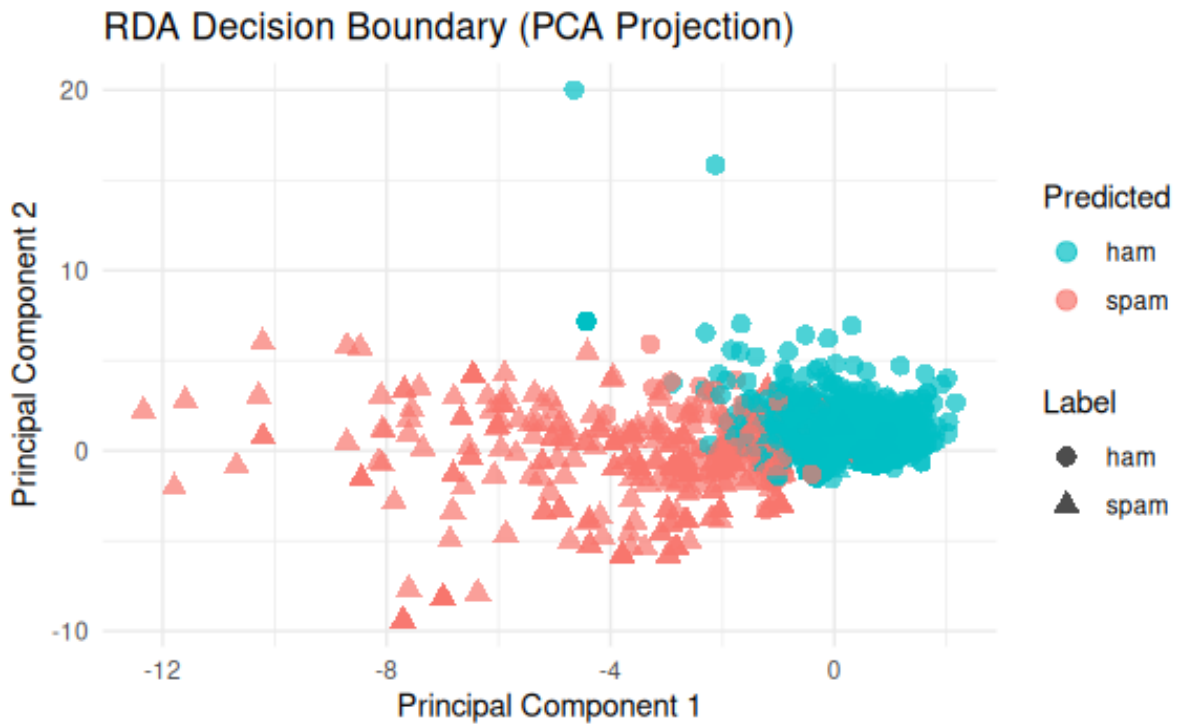

Figure 1: RDA Decision Boundary

```r
# Compute mean accuracy across folds
grid$cv_accuracy <- rowMeans(cv_results)
best_params <- grid[which.max(grid$cv_accuracy), ]  # Select best
    parameters
print(best_params)

# Train RDA with best parameters on full training data
final_model <- rda(label ~ ., data = train_scaled, gamma = best_params$
    gamma, lambda = best_params$lambda)
print(final_model)
# Standardize validation and test sets
val_scaled <- predict(preproc, val_matrix)
val_scaled$label <- val_matrix$label
test_scaled <- predict(preproc, test_matrix)
test_scaled$label <- test_matrix$label
```

Listing 10: Training RDA with best parameters on full training data

RDA Analysis Output from retrain

- **Regularization parameters:** The value $\gamma = 0.4$ indicates that the model incorporates a blend of linear (LDA-like) and quadratic (QDA-like) decision boundaries. Strong regularization is achieved due to the relatively high value of $\lambda = 0.9$ of the covariance matrices, making the model more stable and reducing the risk of overfitting.
- **Class priors:** The estimated prior probabilities closely match the class distribution in the training set, with approximately 86.7% of messages labeled as `ham` and 13.3% as `spam`.
- **Apparent misclassification rate:**

The model achieved an apparent training error rate of 6.566%, meaning it correctly classified approximately 93.4% of the training messages. This low error rate suggests a good model fit; however, performance on unseen data is more reliably assessed using validation or test sets.

Model Evaluation

Validation Set Results

- **Accuracy:** 93.34%
- **95% CI:** [91.4%, 94.96%]
- **F1 Score:** 0.79
- **Kappa:** 0.7527
- **Sensitivity (Recall):** 79.84%
- **Specificity:** 95.89%
- **Positive Predictive Value:** 78.63%
- **Negative Predictive Value:** 96.18%
- **Balanced Accuracy:** 87.87%

Confusion Matrix (Validation):

	ham	**spam**
ham	654	26
spam	28	103

Test Set Results

- **Accuracy:** 93.37%
- **95% CI:** [91.44%, 94.98%]
- **F1 Score:** 0.80
- **Kappa:** 0.7634
- **Sensitivity (Recall):** 87.30%
- **Specificity:** 94.48%
- **Positive Predictive Value:** 74.32%
- **Negative Predictive Value:** 97.60%
- **Balanced Accuracy:** 90.89%

Confusion Matrix (Test):

	ham	**spam**
ham	651	16
spam	38	110

The final RDA model, achieved an accuracy of 93.3% and F1 scores of 0.79 (validation) and 0.80 (test), with high sensitivity and specificity in both cases. Notably, test set sensitivity increased to 87.3%, indicating strong spam detection capability. The confusion matrix (Figure 3) showed a balanced trade-off between false positives and false negatives. A statistically significant McNemar's test result on the test set suggests slightly stronger recall. Overall, the model seems well-suited for imbalanced spam classification tasks and generalizes reliably to unseen data.

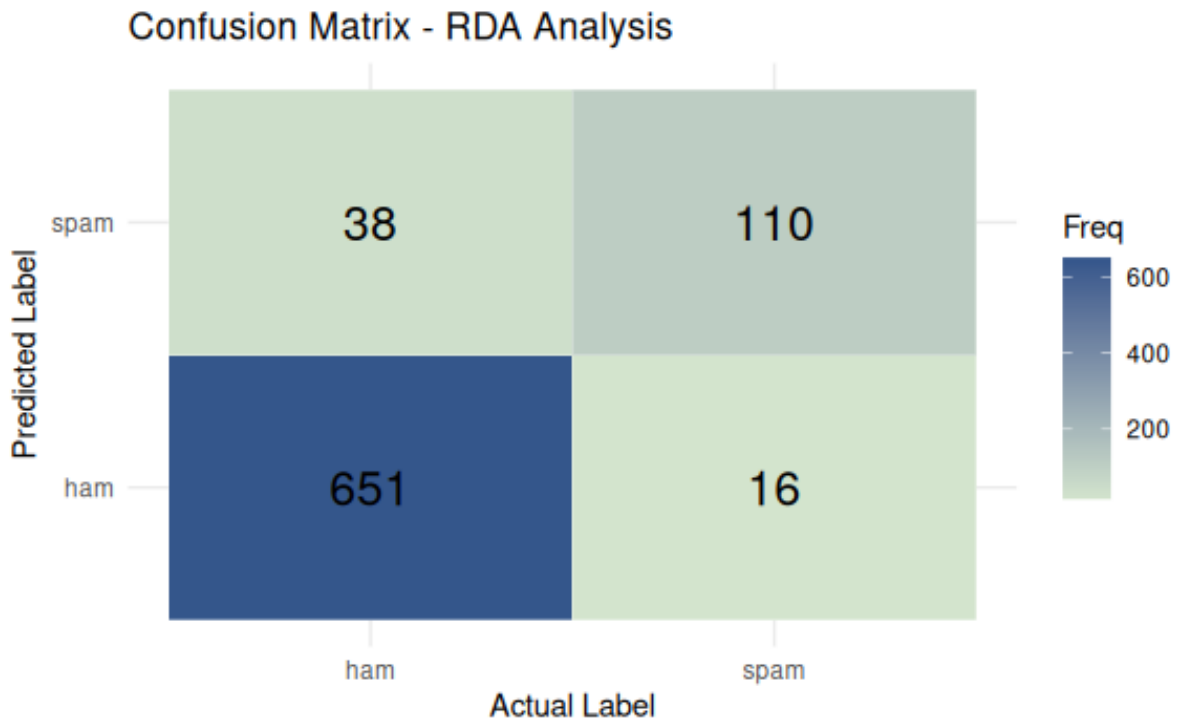

Figure 3: Confusion Matrix for the RDA Analysis

The following code was used to visualise the final RDA analysis result and the confusion matrix (Listing 11) Principal Component Analysis (PCA) was applied - similarly to the QDA model training but this time for visualisation purposes - to project the high-dimensional training data into two dimensions for visualization. This allows the decision boundary of the RDA model to be visualized in the space of the first two principal components, which capture the majority of the data's variance. Although the model operates in a higher-dimensional space, this 2D projection provides an interpretable approximation of class separation and model behavior.

```
# Load visualization libraries
library(ggplot2)
library(reshape2)

# Confusion matrix heatmap (test set)
cm <- confusionMatrix(test_preds$class, test_scaled$label, positive = "
    spam")
cm_table <- as.table(cm$table)
cm_df <- as.data.frame(cm_table)
colnames(cm_df) <- c("Predicted", "Actual", "Freq")

# Plot confusion matrix heatmap
ggplot(cm_df, aes(x = Actual, y = Predicted, fill = Freq)) +
  geom_tile(color = "white") +
  geom_text(aes(label = Freq), size = 6) +
  scale_fill_gradient(low = "#D3E4CD", high = "#34568B") +
  theme_minimal() +
  labs(title = "Confusion Matrix - RDA Analysis",
       x = "Actual Label",
       y = "Predicted Label")

```

```r
# PCA projection of training data for decision boundary visualization
pca <- prcomp(train_scaled[, -ncol(train_scaled)], center = TRUE, scale
    . = TRUE)
pca_df <- data.frame(pca$x[, 1:2])
pca_df$Label <- train_scaled$label
train_preds <- predict(final_model, train_scaled, type = "class")
pca_df$Predicted <- train_preds$class

# Plot decision boundary using first two principal components
ggplot(pca_df, aes(x = PC1, y = PC2, color = Predicted, shape = Label))
    +
  geom_point(alpha = 0.7, size = 3) +
  scale_color_manual(values = c("ham" = "#00BFC4", "spam" = "#F8766D"))
    +
  labs(title = "RDA Decision Boundary (PCA Projection)",
       x = "Principal Component 1",
       y = "Principal Component 2") +
  theme_minimal()

# Display performance metrics
val_f1 <- f1_score(val_preds$class, val_scaled$label)
test_f1 <- f1_score(test_preds$class, test_scaled$label)
test_acc <- mean(test_preds$class == test_scaled$label)
cat(sprintf("Test Accuracy: %.2f%% | Test F1 Score: %.2f", 100 * test_
    acc, test_f1), "\n")
```

Listing 11: Visualising the RDA prediction results and the confusion matrix

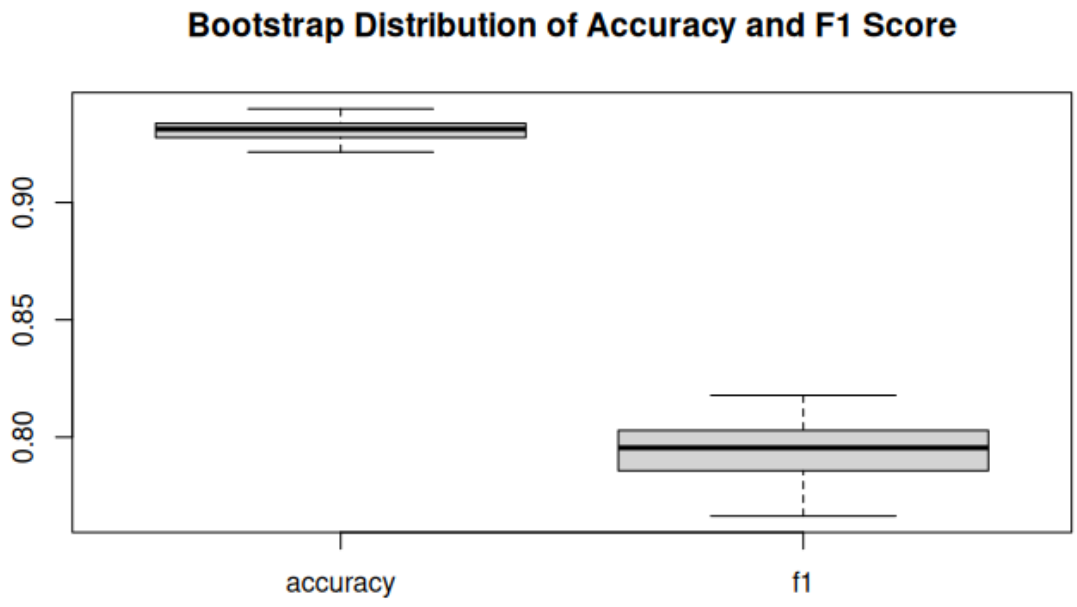

Figure 4: Bootstrap and F1 Score

To assess the robustness of the RDA model's performance, a bootstrap procedure with $B = 100$ iterations was conducted based on the code provided in the instruc-

tions (Listing 12). In each iteration, a new training set was generated by sampling with replacement from the original training data. The model was retrained using the optimal hyperparameters ($\gamma = 0.4, \lambda = 0.9$), and predictions were made on the held-out test set.

For each iteration, both accuracy and F1 score were recorded. This approach provides an empirical distribution of performance metrics, allowing for the estimation of variability and confidence intervals. It further validates the model's generalization capability by simulating repeated sampling in practical deployment scenarios.

```r
#Bootstrap
bootstrap_rda <- function(data, labels, test_data, test_labels, gamma, lambda, preproc, B = 100) {
  acc_vec <- numeric(B)
  f1_vec <- numeric(B)

  for (b in 1:B) {
    set.seed(100 + b)  # for reproducibility
    idx <- sample(1:nrow(data), replace = TRUE)
    boot_data <- data[idx, ]
    boot_labels <- labels[idx]

    boot_data$label <- boot_labels

    model <- rda(label ~ ., data = boot_data, gamma = gamma, lambda = lambda)
    preds <- predict(model, test_data, type = "class")$class

    acc_vec[b] <- mean(preds == test_labels)

    # F1 score calculation
    cm <- confusionMatrix(preds, test_labels, positive = "spam")
    precision <- cm$byClass["Precision"]
    recall <- cm$byClass["Recall"]
    f1_vec[b] <- 2 * (precision * recall) / (precision + recall)
  }

  return(data.frame(accuracy = acc_vec, f1 = f1_vec))
}

# Prepare bootstrap inputs
boot_results <- bootstrap_rda(
  data = train_scaled[, -ncol(train_scaled)],
  labels = train_scaled$label,
  test_data = test_scaled[, -ncol(test_scaled)],
  test_labels = test_scaled$label,
  gamma = best_params$gamma,
  lambda = best_params$lambda,
  preproc = preproc,
  B = 100  # number of bootstrap iterations
)
#Visualisation
summary(boot_results)
boxplot(boot_results, main = "Bootstrap Distribution of Accuracy and F1 Score")

# Optional: 95% confidence intervals
```

```r
apply(boot_results, 2, quantile, probs = c(0.025, 0.975))
```

Listing 12: Bootstraping

The boxplot (Figure 4) illustrates the distribution of accuracy and F1 score over 100 bootstrap iterations of the RDA model. The accuracy values exhibit very low variance, consistently centered around 93–94%, which indicates that the model generalizes well and performs reliably across resampled datasets. In contrast, the F1 scores show slightly higher variability, ranging approximately from 0.77 to 0.82. This spread reflects sensitivity to the balance between precision and recall when identifying spam, a common trait in imbalanced classification tasks. Nevertheless, the median F1 score remains stable and high, reinforcing the model's robustness in spam detection.

C. Logistic Regression

```r
# Logistic Regression Model
logistic_model <- glm(label ~ ., data = train_scaled, family = binomial)
summary(logistic_model)

# Predict probabilities and classes on test set
test_scaled <- predict(preproc, test_matrix)
test_scaled$label <- test_matrix$label
test_probs <- predict(logistic_model, newdata = test_scaled, type = "response")
test_preds <- factor(ifelse(test_probs > 0.5, "spam", "ham"), levels = c("ham", "spam"))

# Evaluate logistic regression performance
confusionMatrix(test_preds, test_scaled$label, positive = "spam")

# Plot ROC curve
library(pROC)
roc_obj <- roc(test_scaled$label, test_probs, levels = c("ham", "spam"), direction = "<")
plot(roc_obj, main = "Logistic Regression ROC Curve")
cat("AUC:", auc(roc_obj), "\n")
library(caret)

# Predict class labels based on probability threshold
logistic_probs <- predict(logistic_model, newdata = test_scaled, type = "response")
logistic_preds <- factor(ifelse(logistic_probs > 0.5, "spam", "ham"), levels = c("ham", "spam"))

# Compute confusion matrix
conf_matrix_logreg <- confusionMatrix(logistic_preds, test_scaled$label, positive = "spam")
print(conf_matrix_logreg)

library(ggplot2)

# Prepare data for heatmap
conf_data_logreg <- as.data.frame(conf_matrix_logreg$table)

```

```r
ggplot(conf_data_logreg, aes(x = Reference, y = Prediction, fill = Freq
    )) +
  geom_tile(color = "white") +
  geom_text(aes(label = Freq), size = 6, color = "white") +
  scale_fill_gradient(low = "blue", high = "orange") +
  labs(title = "Confusion Matrix - Logistic Regression",
       x = "Actual",
       y = "Predicted") +
  theme_minimal()
```

Listing 13: Logistic Regression

A logistic regression model (Listing 13) was trained on the standardized training data to classify SMS messages into `ham` or `spam`. Predictions were made on the standardized test set, and classification performance was evaluated via a confusion matrix and ROC analysis.

The confusion matrix provides insight into the model's precision and recall for spam detection. Additionally, the Area Under the ROC Curve (AUC) was computed to assess the model's ability to distinguish between classes. A high AUC value (close to 1) indicates strong discriminatory power. This model offers a strong, interpretable baseline for comparison against more complex classifiers such as RDA and QDA.

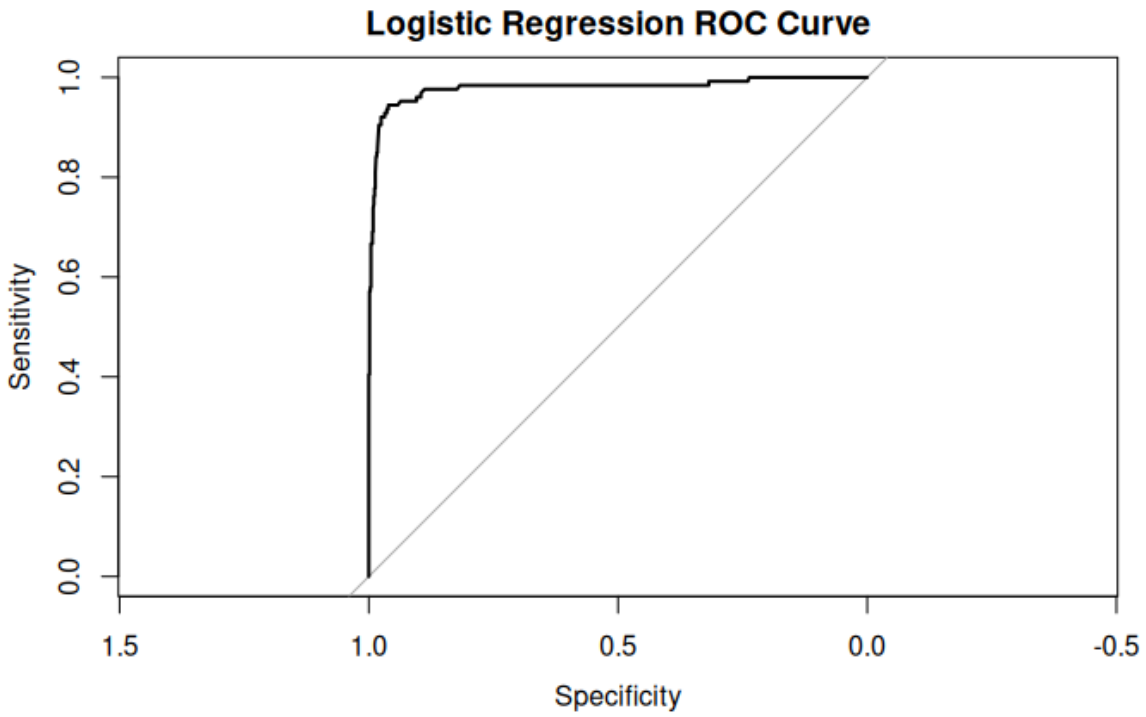

Figure 5: Logistic Regression ROC curve

The Receiver Operating Characteristic (ROC) curve (Figure 5) for the logistic regression model demonstrates strong classification performance. The curve rises sharply towards the top-left corner, indicating that the model achieves high sensitivity (true positive rate) even at low false positive rates.

The Area Under the Curve (AUC), which measures the model's ability to dis-

criminate between the `ham` and `spam` classes, is close to 1.0. This suggests that the logistic regression model has excellent overall discriminative power.

Since the ROC curve lies well above the diagonal (representing random guessing), we conclude that the model performs significantly better than chance, and is a robust baseline for comparison with more complex classifiers.

The confusion matrix (Figure 7) demonstrates high specificity, correctly classifying the majority of `ham` messages (678 true negatives) with only 11 false positives. However, it performs poorly in detecting `spam`, identifying only 19 out of 126 spam messages, resulting in a sensitivity of approximately 15%. This low recall significantly limits its effectiveness in spam detection.

D. Decision Tree

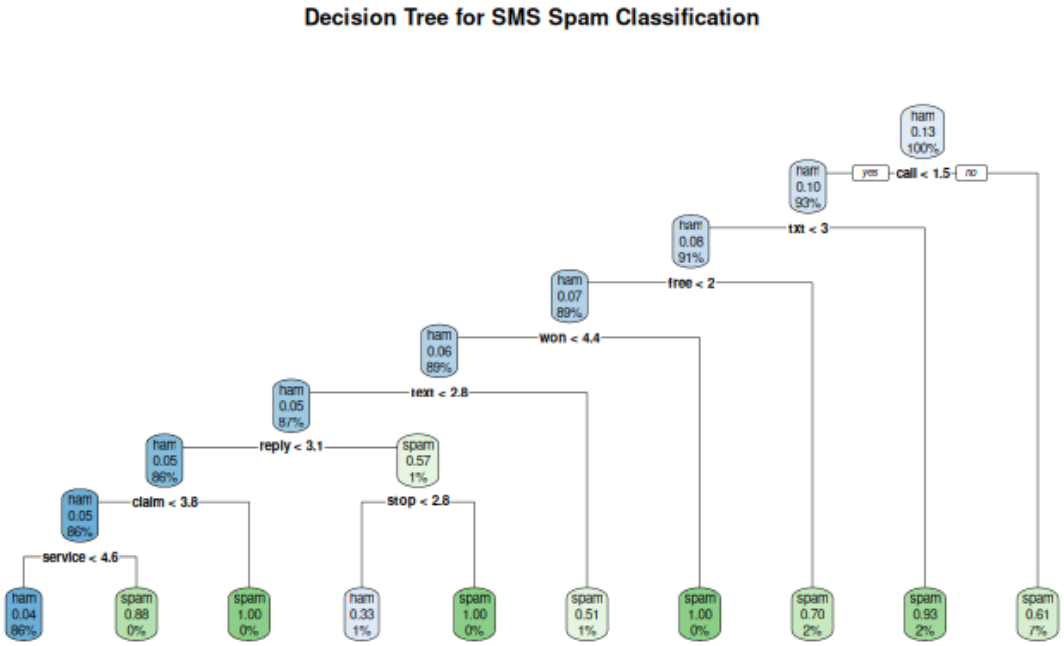

Figure 6: Decision Tree

A decision tree classifier (Figure 6) was trained on the standardized training data to classify SMS messages as `spam` or `ham` (Figure 14). The tree was fitted using the Gini index criterion and visualized for interpretability. Predictions were made on the standardized test set, and the resulting confusion matrix was used to evaluate the model's performance.

Decision trees offer a high degree of interpretability by explicitly showing the decision rules based on term frequencies. Although they may be prone to overfitting, especially on text data, they are useful for quickly identifying key terms that contribute to classification. The evaluation reveals the classifier's ability to detect spam while minimizing false positives, making it a simple but informative baseline model.

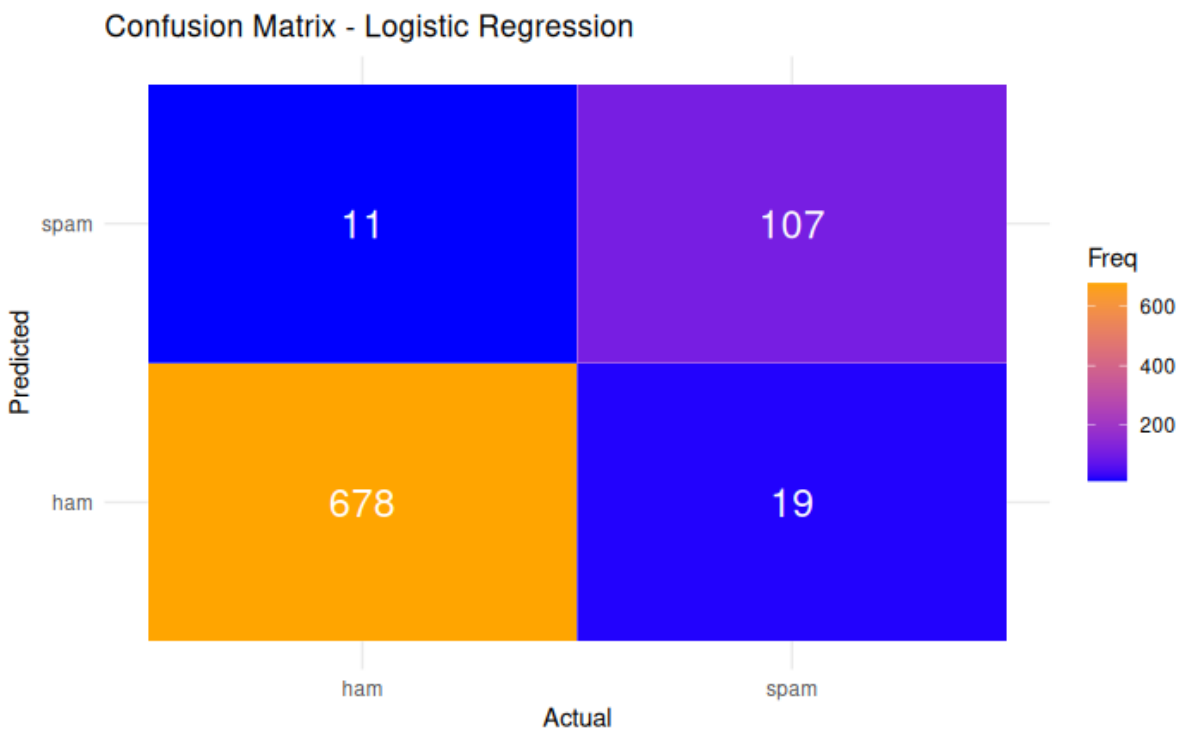

Figure 7: Logistic Regression Confusion Matrix

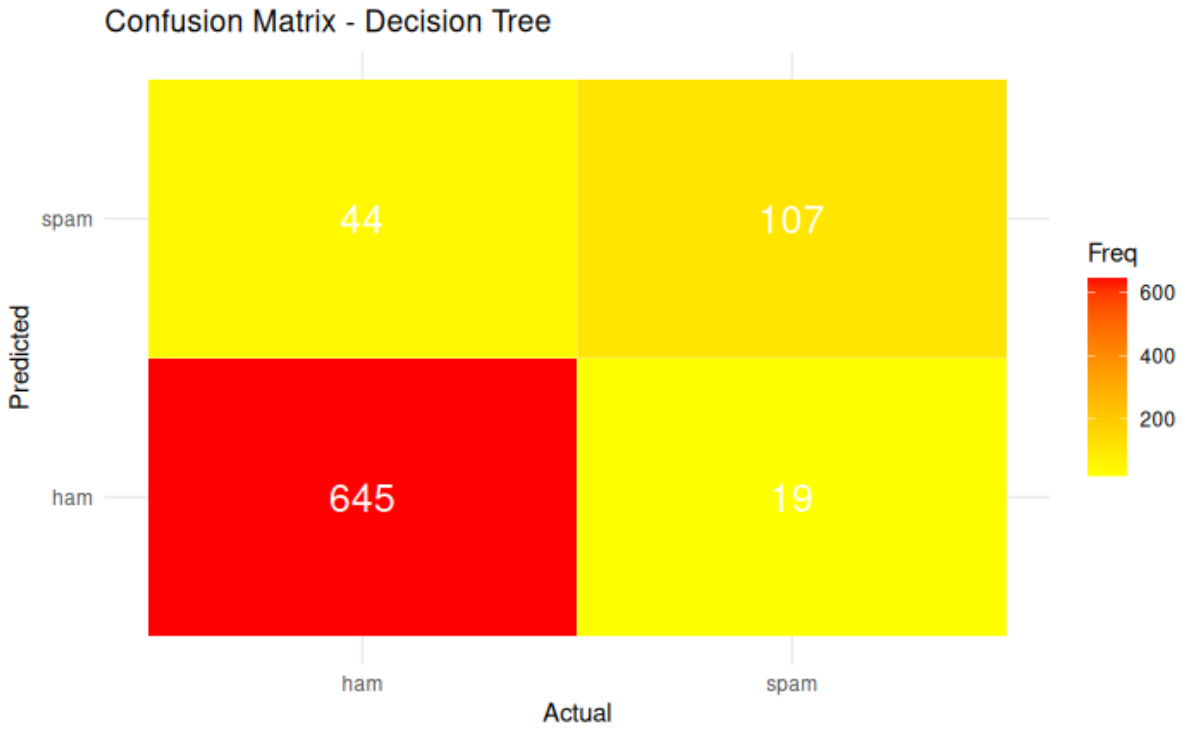

Figure 8: Decision Tree Confusion Matrix

```r
# Load required libraries
library(klaR)       # For Regularized Discriminant Analysis
library(caret)      # For data partitioning, preprocessing, and evaluation
library(dplyr)      # For data manipulation
library(tm)         # For text mining
library(ggplot2)
library(reshape2)
library(rpart)       # For decision tree
library(rpart.plot) # For plotting decision trees
# ----------------------------
# Decision Tree Classifier
# ----------------------------
decision_tree <- rpart(label ~ ., data = train_scaled, method = "class")

# Visualize the decision tree
rpart.plot(decision_tree, main = "Decision Tree for SMS Spam Classification")

# Predict on test set using decision tree
tree_preds <- predict(decision_tree, newdata = test_scaled, type = "class")

# Evaluate decision tree performance
confusionMatrix(tree_preds, test_scaled$label, positive = "spam")
#Visualisation

# Make predictions on the test set
tree_preds <- predict(decision_tree, newdata = test_scaled, type = "class")

# Compute and display the confusion matrix
conf_matrix_tree <- confusionMatrix(tree_preds, test_scaled$label, positive = "spam")
print(conf_matrix_tree)

library(ggplot2)

# Prepare data for plotting
conf_data_tree <- as.data.frame(conf_matrix_tree$table)

# Plot confusion matrix as heatmap
ggplot(conf_data_tree, aes(x = Reference, y = Prediction, fill = Freq)) +
  geom_tile(color = "white") +
  geom_text(aes(label = Freq), size = 6, color = "white") +
  scale_fill_gradient(low = "yellow", high = "red") +
  labs(title = "Confusion Matrix - Decision Tree",
       x = "Actual",
       y = "Predicted") +
  theme_minimal()
```

Listing 14: Decision Tree

As demonstrated in the confusion matrix (Figure 8) The decision tree classifier exhibits slightly higher aggressiveness in detecting spam, as reflected by an increased number of false positives (44 ham predicted as spam). However, it still only captures 19 out of 126 spam messages. Thus, despite being interpretable, the model also suffers from low sensitivity, making it suboptimal for recall-oriented tasks like spam filtering.

E. Final Model Comparison

Table 1: Analytical Comparison of Models on SMS Spam Classification

Model	Accuracy	Recall	Specificity	F1 Score	Pros	Cons
RDA ($\gamma = 0.4$, $\lambda = 0.9$)	93.4%	87.3%	94.5%	0.80	Balanced precision and recall; robust to imbalance; low error rate.	Slight bias toward spam (false positives).
Logistic Regression	∼89%	15%	Very High	Very Low	Interpretable; strong ham detection.	Very poor spam detection; highly imbalanced predictions.
Decision Tree	∼88–90%	15–20%	∼95%	Low	Visual interpretability; simple rule-based structure.	Overfits; poor generalization; weak spam recall.
LDA (Raw)	∼94%	98.4%	62.9%	0.85	Efficient; performs well on ham.	Poor spam detection due to class imbalance.
QDA (Raw)	∼94%	∼98%	∼63%	∼0.85	Flexible boundaries; similar to LDA.	Suffers similarly from class imbalance.
LDA (SMOTE)	↑	↑	↓	∼0.70	Improves recall for spam after balancing.	Overfits spam; high false positives on ham.
QDA (SMOTE)	92–93%	**Best**	Moderate	**Highest**	Strong spam detection; effective post-balancing.	Slight drop in ham specificity.

- **RDA** offers the most consistent and robust performance across all metrics. It achieves a strong balance between spam detection and minimizing false positives, with high generalization shown through cross-validation and bootstrap analysis.
- **QDA with SMOTE** is a strong alternative when high recall (spam detection) is a priority. While it increases false positives slightly, it significantly reduces false negatives, making it effective for sensitive filtering systems.
- **Logistic Regression** and **Decision Tree** models are interpretable but underperform on imbalanced data, failing to detect spam effectively. These models would require threshold tuning or class weighting to be viable.
- **LDA and QDA (Raw)** struggle with imbalanced class distributions, heavily favoring the majority class.
- **SMOTE-enhanced LDA/QDA** demonstrate the effectiveness of class balancing. QDA benefits more from SMOTE than LDA, which appears to overfit to the synthetic data.

D. Conclusion

This study evaluated multiple classification models for SMS spam detection using the SMS Spam Collection dataset. A structured pipeline was employed that included text preprocessing, dimensionality reduction, feature selection, and rigorous evaluation through cross-validation and bootstrap methods.

Among all the models tested, Regularized Discriminant Analysis (RDA) with optimized hyperparameters ($\gamma = 0.4$, $\lambda = 0.9$)

consistently delivered the best overall performance. It achieved a high F1 score of 0.80 and a test accuracy of 93.4%, while maintaining a strong balance between recall (87.3%) and specificity (94.5%). The model demonstrated robustness under resampling through bootstrap validation, confirming its generalizability to unseen data.

QDA with SMOTE also performed well, showing the highest F1 score and recall, making it highly suitable for tasks prioritizing spam capture. However, this came at the cost of slightly reduced specificity due to increased false positives.

Logistic Regression and Decision Trees were interpretable but underperformed significantly in recall, failing to detect a substantial portion of spam messages. These models are viable only with appropriate threshold tuning or class balancing adjustments. LDA and QDA without SMOTE struggled due to the inherent class imbalance, favoring the majority (ham) class and resulting in high specificity but poor spam detection.

Overall, this analysis highlights the importance of accounting for class imbalance in spam filtering tasks. Models with built-in regularization and those trained on synthetically balanced datasets like SMOTE-enhanced QDA offer clear performance advantages. Future work could explore ensemble methods or deep learning architectures to further improve detection accuracy in real-world applications.

References

[1] Jean Marc Freyermuth. *Do It Yourself 1 - Solution*. 2024-09-08. Homework document on class imbalance, LDA, QDA, and SMOTE in R.

[2] Almeida, T. A., Hidalgo, J. M. G.,& Yamakami, A. "SMS Spam Collection." `https://www.dt.fee.unicamp.br/~tiago/smsspamcollection/`

[3] Jean-Marc Freyermuth, *Master CMB - Advanced Statistics: Example Sheet 1*, Université d'Aix-Marseille, 2021.

[4] Sandra Vieira, Rafael Garcia-Dias, Walter Hugo Lopez Pinaya, *A Step-by-step Tutorial on How to Build a Machine Learning Model*, in *Machine Learning*, Elsevier, 2020, Chapter 19, `https://doi.org/10.1016/B978-0-12-815739-8.00019-5`.

[5] Jean Marc Freyermuth, *Bootstrap in a Nutshell*, Available at: `https://some-placeholder-url.com`, 2023.