Comparative Machine Learning Analysis on UCI Heart Disease and Spambase Datasets

Sihui Lyu[*]
lyu.sihu@northeastern.edu
Northeastern University
San Jose, California, USA

Rohan Verma[*]
verma.roha@northeastern.edu
Northeastern University
San Jose, California, USA

Abstract

This project investigates and compares the performance of several traditional machine learning models on two classic datasets from the UCI Machine Learning Repository: the Heart Disease dataset and the Spambase dataset. These datasets represent two distinct data types—small-scale, mixed-type clinical data and large-scale, high-dimensional text frequency data. Our objective is to examine how different models perform under varying data characteristics, focusing on preprocessing strategies, hyperparameter tuning, and evaluation metrics. We apply Naive Bayes, Logistic Regression, Random Forest, Support Vector Machine, k-Nearest Neighbors, and Decision Tree models, using standardized preprocessing pipelines and grid search with cross-validation. The results reveal clear differences in model behavior between tabular and text datasets, offering practical insights into model selection and evaluation. Future work will explore boosting algorithms and advanced feature engineering to further enhance performance.

Keywords: Comparative machine learning, UCI Heart Disease, UCI Spambase, tabular data, spam detection, preprocessing pipeline, hyperparameter tuning, Random Forest, Support Vector Machine (SVM), Logistic Regression, Naive Bayes, ROC-AUC

1 Introduction

Machine learning models can exhibit different behaviors depending on the characteristics of the dataset they are applied to. In this project, we systematically compare model performance across two distinct types of data drawn from the UCI Machine Learning Repository:

- UCI Heart Disease: a small, mixed-type clinical dataset with 303 samples and 13 features.
- UCI Spambase: a larger dataset with 4,601 samples and 57 numerical frequency features extracted from email messages.

Our objective is to analyze and compare the effectiveness of traditional machine learning models on these two datasets, focusing on how preprocessing strategies, model selection, and hyperparameter tuning influence performance. We evaluate a suite of models—Naive Bayes, Logistic Regression, Random Forest, Support Vector Machine (SVM), k-Nearest Neighbors (KNN), and Decision Tree — using standard metrics including accuracy, precision, recall, F1-score, and ROC-AUC. This comparative analysis highlights how data characteristics shape model behavior and provides practical guidance for model selection in different application domains.

[*]Both authors contributed equally to this research.

2 Dataset A: UCI Heart Disease

2.1 Data Exploration and Preprocessing

2.1.1 Data Description. We used the Cleveland Heart Disease dataset, an open dataset published on the UCI Machine Learning Repository. This dataset contains 303 observations with 13 attributes, and consists of a mix of numerical and categorical variables. Table 1 shows the 13 attributes.

2.1.2 Data Cleaning. After reviewing the dataset structure, we performed several cleaning steps to handle missing values and define the target variable:

1. **Assigning Column Names.** The original data file does not include a header row. We manually assigned column names based on the UCI dataset description.
2. **Handling Missing Values.** There were 4 missing values in the 'ca' column and 2 in the 'thal' column, represented by '?'. These were replaced with NaN and converted to numeric types. Since both columns are categorical with limited values, we filled the missing values using mode imputation to maintain the original distribution.
3. **Defining the Target Variable.** The original dataset includes a 'num' column indicating heart disease severity on a scale from 0 to 4, where 0 indicates no disease and 1–4 indicate increasing levels of severity. Following common practice, we transformed this into a binary classification problem:

$$\text{target} = \begin{cases} 0 & \text{no heart disease (num = 0)} \\ 1 & \text{presence of heart disease (num > 0)} \end{cases}$$

This simplifies the task to distinguishing disease vs. no disease.

Table 1. Description of the 13 attributes in the Cleveland Heart Disease dataset.

Column	Description
`age`	Age (years)
`sex`	Sex (1 = male; 0 = female)
`cp`	Chest pain type (1 = typical angina; 2 = atypical angina; 3 = non-anginal pain; 4 = asymptomatic)
`trestbps`	Resting blood pressure (mm Hg)
`chol`	Serum cholesterol (mg/dl)
`fbs`	Fasting blood sugar > 120 mg/dl (1 = true; 0 = false)
`restecg`	Resting electrocardiographic results (0 = normal; 1 = ST–T wave abnormality; 2 = left ventricular hypertrophy)
`thalach`	Maximum heart rate achieved
`exang`	Exercise-induced angina (1 = yes; 0 = no)
`oldpeak`	ST depression induced by exercise relative to rest
`slope`	Slope of the peak exercise ST segment (1 = upsloping; 2 = flat; 3 = downsloping)
`ca`	Number of major vessels (0–3) colored by fluoroscopy
`thal`	Thalassemia (3 = normal; 6 = fixed defect; 7 = reversible defect)

2.1.3 Exploratory Data Analysis (EDA) .

After data cleaning, we conducted exploratory data analysis to better understand the dataset's structure and feature–target relationships.

1. **Numerical Features:** For numerical features such as age, chol, and thalach, the distributions are mostly concentrated, with chol showing a right-skewed pattern and a few extreme values.

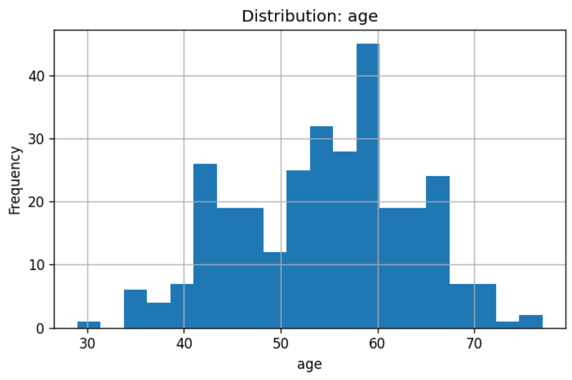

Figure 1. Distribution of Age in the UCI Heart Disease Dataset.

2. **Categorical Features:** For categorical variables, such as chest pain type (cp), asymptomatic cases (cp = 4) are the most common among positive heart disease patients, indicating its potential predictive value.
3. **Feature–Target Correlations:** Finally, we visualized feature–target correlations, which show that most features are moderately correlated with the target, while fbs stands out as weakly correlated.

This exploration helped us identify patterns such as the prevalence of specific chest pain types among positive cases, as well as correlations between features like thalach (maximum heart rate) and heart disease presence. These findings informed our feature preprocessing and model selection in subsequent steps.

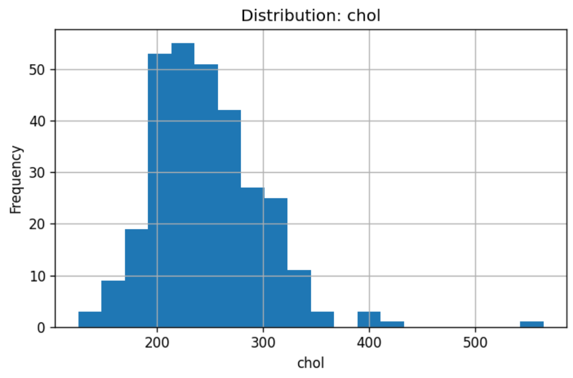

Figure 2. Distribution of Serum Cholesterol (chol) in the UCI Heart Disease Dataset.

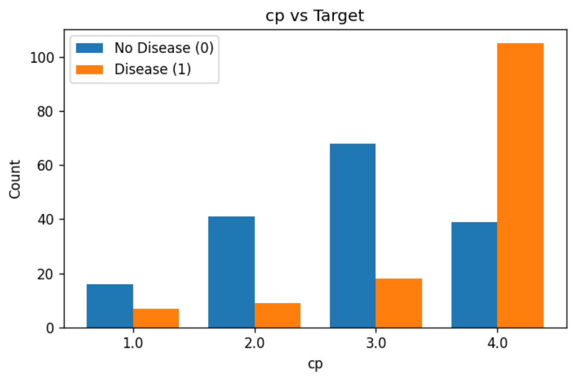

Figure 3. Chest Pain Type (cp) vs Target Class Distribution.

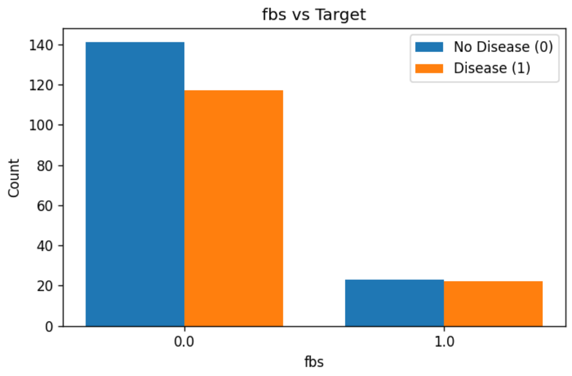

Figure 4. Fasting Blood Sugar (fbs) vs Target Class Distribution.

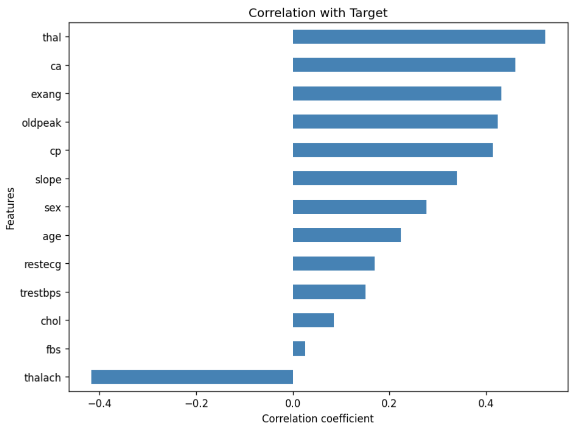

Figure 5. Feature Correlation with the Target Variable.

2.1.4 Preprocessing for Modeling.

Before model training, we applied a structured preprocessing pipeline to handle different feature types appropriately. This was implemented using scikit-learn's ColumnTransformer, which allows applying different transformations to different subsets of features within a single pipeline.

We divided the features into three groups:

1. **Numerical features (age, trestbps, chol, thalach, oldpeak):** Missing values were imputed using the median, and features were standardized using StandardScaler to ensure that all numeric features are on a comparable scale. This is particularly important for models that are sensitive to feature magnitudes, such as Logistic Regression and SVM.
2. **Categorical features requiring one-hot encoding (cp, thal, slope):** Missing values were imputed with the most frequent category, followed by one-hot encoding (dropping the first category to avoid multicollinearity). This representation allows linear and tree-based models to leverage categorical information effectively.
3. **Binary or ordinal-like categorical variables (sex, fbs, restecg, exang, ca):** These were imputed with the most frequent value and retained as numeric values without one-hot encoding.

The resulting preprocessing pipeline ensured that each feature type was transformed appropriately and consistently during both training and evaluation. After preprocessing, the transformed feature matrix contained the original numerical features, the expanded one-hot encoded features, and the binary/ordinal features.

Finally, we performed a stratified 80/20 train–test split to preserve the original class distribution in both sets. This helps ensure that evaluation metrics reflect real-world class proportions and reduces sampling bias.

2.2 Models and Methodology

We implemented and compared five supervised machine learning models using the preprocessed Cleveland Heart Disease dataset (Section 2). All models were trained on 80% of the data and evaluated on a held-out 20% test set.

To ensure consistency and prevent data leakage, we used scikit-learn Pipelines to integrate preprocessing and model fitting. Hyperparameter tuning was conducted using GridSearchCV with 5-fold Stratified Cross-Validation on the training set only. The F1-score was used as the primary selection metric to balance precision and recall.

2.2.1 Method 1: Random Forest.

For the Random Forest classifier, the following hyperparameters were tuned:

- **Number of trees** (`n_estimators`) $\in \{200, 400, 600\}$ — increasing this generally improves performance but also increases training time.
- **Maximum depth** (`max_depth`) $\in$ {None, 4, 6, 10} — controls model complexity.
- **Minimum samples to split** (`min_samples_split`) $\in \{2, 5, 10\}$ — regularizes tree growth.

This grid explores both underfitting and overfitting regimes. Random Forest was selected for its ability to capture non-linear relationships and feature interactions with minimal preprocessing. The tuned hyperparameters and performance metrics are reported in Section 2.3.

2.2.2 Method 2: SVM (RBF Kernel).

For the SVM classifier, two key parameters were explored using SVC:

- **C** — Regularization parameter.
- **kernel**— Specifies the kernel type used in the algorithm.

The RBF (Radial Basis Function) kernel was selected due to its strong performance on non-linear datasets. The optimal hyperparameters and corresponding evaluation results are presented in Section 2.3.

2.2.3 Method 3: k-Nearest Neighbors (KNN).

For the KNN model, we tuned:

- **n_neighbors**— Number of neighbors considered for each prediction.
- **weights**— Neighbor weighting function (uniform vs. distance).
- **p**— Power parameter for the Minkowski distance metric (p = 1 for Manhattan, p = 2 for Euclidean).

Smaller n_neighbors values can lead to overfitting, whereas larger values smooth the decision boundary and may improve generalization. The selected hyperparameters and evaluation results are summarized in Section 2.3.

2.2.4 Method 4: Logistic Regression. We tuned the regularization strength C over 0.01, 0.1, 1, 5, 10, covering both strong and weak regularization regimes. Smaller values correspond to stronger regularization (simpler models), whereas larger values allow the model to fit the data more closely. The model used an L2 penalty, the lbfgs solver, and a maximum of 2000 iterations to ensure convergence. The selected hyperparameters and model performance are reported in Section 2.3.

2.2.5 Method 5: Decision Tree. For the Decision Tree model, several hyperparameters were tuned to control complexity and reduce overfitting:

- `criterion` $\in$ {`gini`, `entropy`}
- `splitter` $\in$ {`best`, `random`}
- `max_depth`, `min_samples_split`, `min_samples_leaf`

Unrestricted trees tend to overfit, so these hyperparameters were carefully tuned to balance model complexity and generalization. The tuning outcomes and evaluation metrics are presented in Section 2.3.

2.3 Results

2.3.1 Model Performance Comparison. Table 2 summarizes the performance of all five models on the held-out test set using the best hyperparameters obtained from GridSearchCV. The evaluation metrics include Accuracy, Precision, Recall, F1-score, and ROC-AUC, which provide a comprehensive view of each model's predictive ability.

2.3.2 Confusion Matrices and ROC Curves. Figures 6-15 present the confusion matrices and ROC curves for each model.

- **Confusion matrices** provide a detailed view of correct and incorrect classifications, allowing analysis of false positives and false negatives.
- **ROC curves** visualize the trade-off between sensitivity and specificity, while ROC-AUC scores summarize the model's ability to distinguish between classes.

2.3.3 Summary.

- **Random Forest** achieved the best recall, making it suitable when identifying positive cases is critical.
- **SVM** achieved the best precision, indicating fewer false positives.

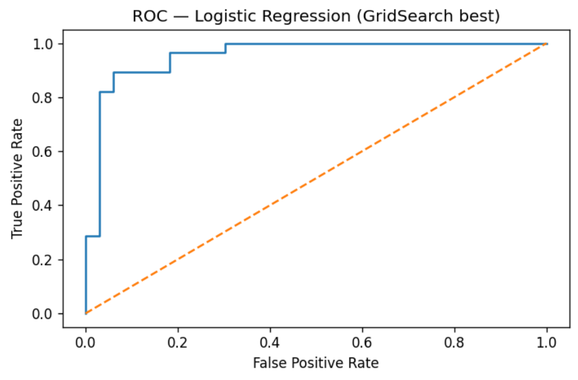

Figure 6. ROC Curve for Logistic Regression (Best Model after GridSearchCV).

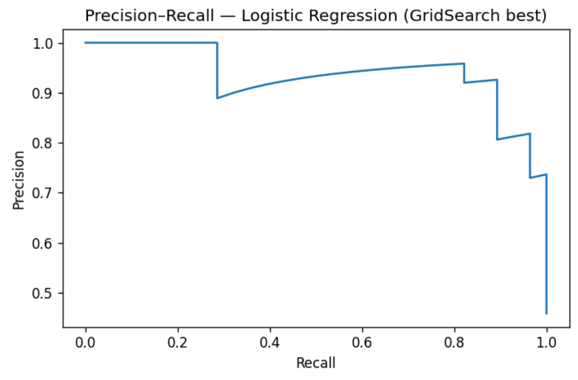

Figure 7. Precision–Recall Curve for Logistic Regression (Best Model after GridSearchCV).

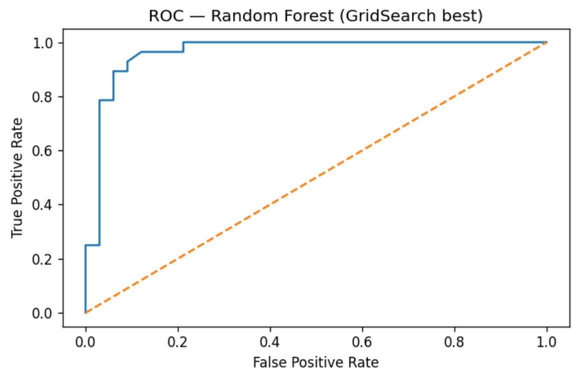

Figure 8. ROC Curve for Random Forest (Best Model after GridSearchCV).

- **KNN** showed the highest ROC-AUC, but with slightly lower precision.
- **Logistic Regression** provided a good balance between interpretability and performance.

Table 2. Model performance comparison on the Cleveland Heart Disease dataset.

Model	Accuracy	Precision	Recall	F1	ROC-AUC
`Random Forest`	0.9180	0.8710	0.9643	0.9153	0.9627
`SVM (RBF)`	0.9180	0.9259	0.8929	0.9091	0.9654
`KNN`	0.9016	0.8929	0.8929	0.8929	**0.9784**
`Logistic Regression`	0.8525	0.8065	0.8929	0.8475	0.9556
`Decision Tree`	0.8197	0.8696	0.7143	0.7843	0.8490

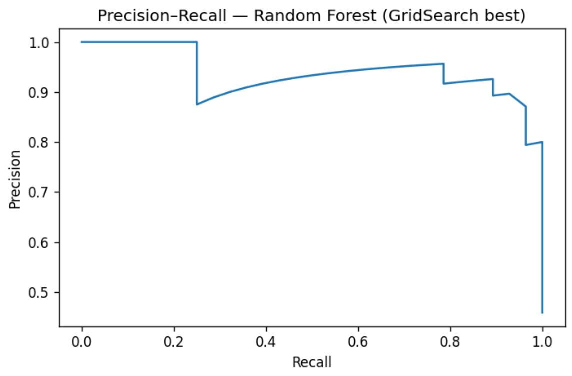

Figure 9. Precision–Recall Curve for Random Forest (Best Model after GridSearchCV).

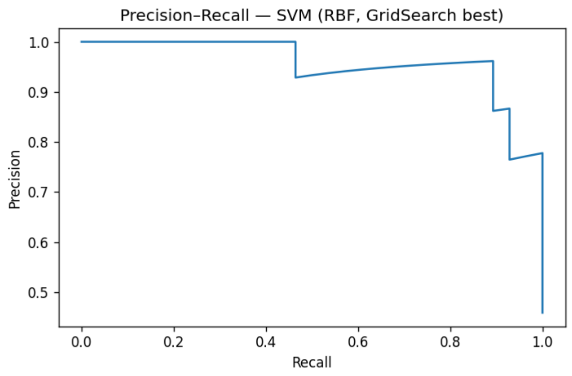

Figure 11. Precision–Recall curve for SVM (RBF kernel), best model after GridSearchCV.

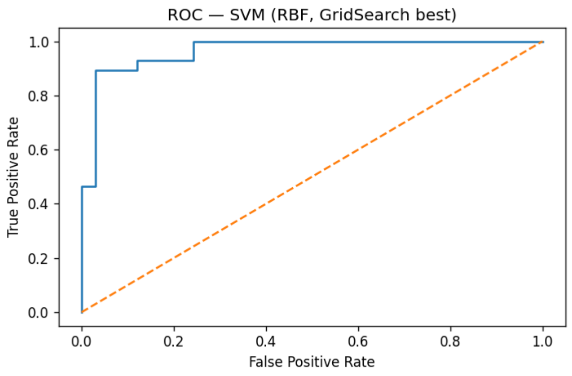

Figure 10. ROC Curve for SVM with RBF Kernel (Best Model after GridSearchCV).

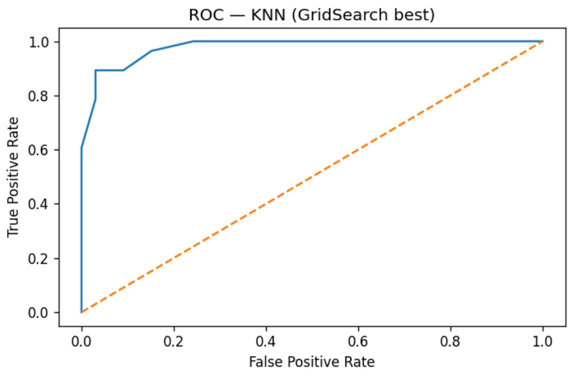

Figure 12. ROC curve for KNN, best model after GridSearchCV.

- **Decision Tree** underperformed relative to the others due to overfitting, despite tuning.

Depending on the application focus (e.g., prioritizing recall vs. precision), different models may be preferred. For example, in medical applications such as heart disease prediction, higher recall is often prioritized to minimize missed cases.

2.4 Discussion

In this study, we compared five supervised machine learning algorithms on the Cleveland Heart Disease dataset. Random Forest and SVM achieved the highest test accuracy, while KNN obtained the highest ROC-AUC. Logistic Regression provided a strong balance between performance and interpretability, making it a good baseline. Decision Trees, despite tuning, showed signs of overfitting.

Depending on the application focus, different models may be preferred. For example, in medical diagnosis, higher recall is often prioritized to minimize missed cases, making Random Forest particularly suitable. Future work could explore model ensembles or feature selection techniques to further improve predictive performance.

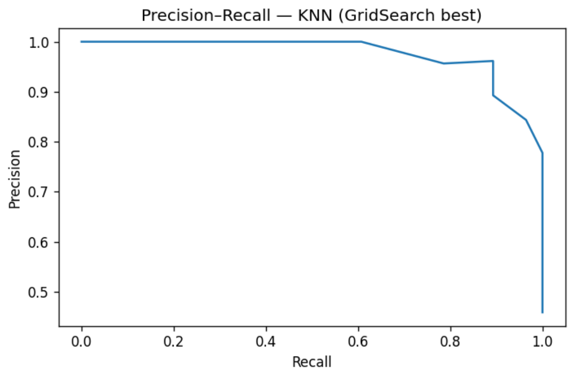

Figure 13. Precision–Recall curve for KNN, best model after GridSearchCV.

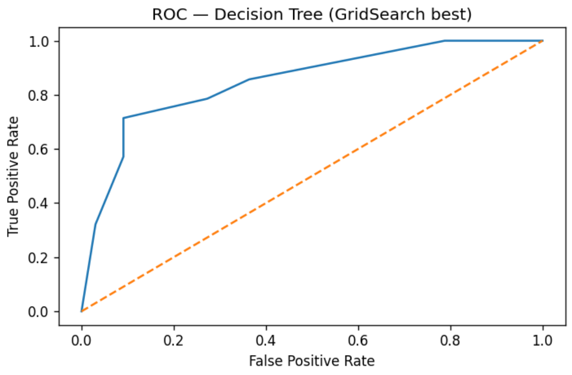

Figure 14. ROC curve for Decision Tree, best model after GridSearchCV.

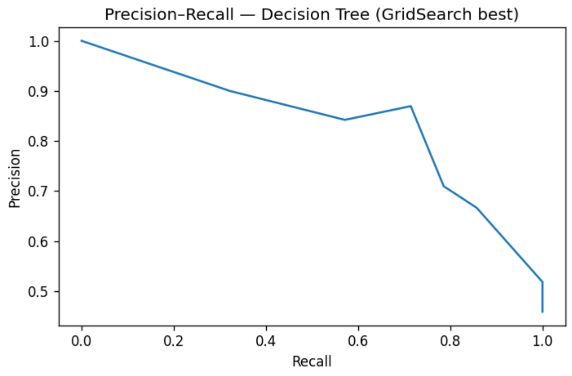

Figure 15. Precision–Recall curve for Decision Tree, best model after GridSearchCV.

3 Dataset B: UCI Spambase

3.1 Data Exploration and Preprocessing

3.1.1 Dataset Overview. Our team analyzed the UCI Spambase dataset, a binary classification dataset designed for email spam detection. The dataset contains 4,601 email samples collected from a corporate email server, with each email labeled as either spam (1) or non-spam (0). Our primary objective was to build machine learning models capable of automatically distinguishing spam emails from legitimate correspondence. We identified that the dataset comprises 57 numerical features extracted from email content, structured as follows:

- **48 word frequency features**: Represent the frequency of occurrence of specific words commonly associated with spam or legitimate emails. Examples include:
  – `word_freq_make`
  – `word_freq_address`
  – `word_freq_free`
  – `word_freq_remove`
- **6 character frequency features**: Capture the frequency of special characters such as `!`, `$`, `(`, `[`, and `;`. Examples include:
  – `char_freq_!`
  – `char_freq_$`
- **3 capital letter sequence features**: Measure statistics related to sequences of consecutive capital letters. These include:
  – `capital_run_length_average`
  – `capital_run_length_longest`
  – `capital_run_length_total`

3.1.2 Exploratory Data Analysis. Through our exploratory analysis, we observed that the target variable exhibits a relatively balanced distribution with 2,788 non-spam emails (60.60%) and 1,813 spam emails (39.40%). Our team recognized this near-balanced class distribution as advantageous for model training, as it reduces the risk of class imbalance bias and eliminates the need for specialized sampling techniques such as SMOTE or class weighting.

Key dataset characteristics we identified during exploration:

- **Total samples:** 4,601 emails
- **Missing values:** None (0)
- **Duplicate rows:** We found 394 duplicate entries and made the decision to retain them in our analysis to preserve the natural distribution of email patterns
- **Feature scale:** We noted that features exhibit wide variation in scale, ranging from 0 to 42.81 for some word frequency measures

Our statistical analysis of the first 10 features revealed that most features have highly skewed distributions with medians at zero and high standard deviations, indicating that certain words appear infrequently but with high intensity when present. For instance, we observed that word_freq_3d has a mean of 0.065 but a maximum value of 42.81, suggesting extreme values in some spam messages.

Our feature distribution analysis comparing spam versus non-spam emails shows distinct patterns. We found that

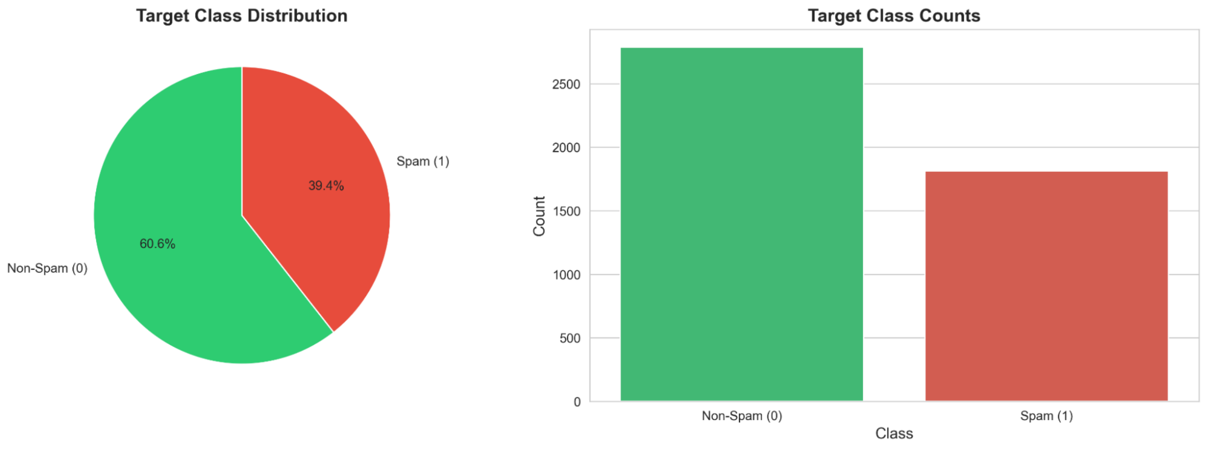

Figure 16. Target class distribution for the Spambase dataset.

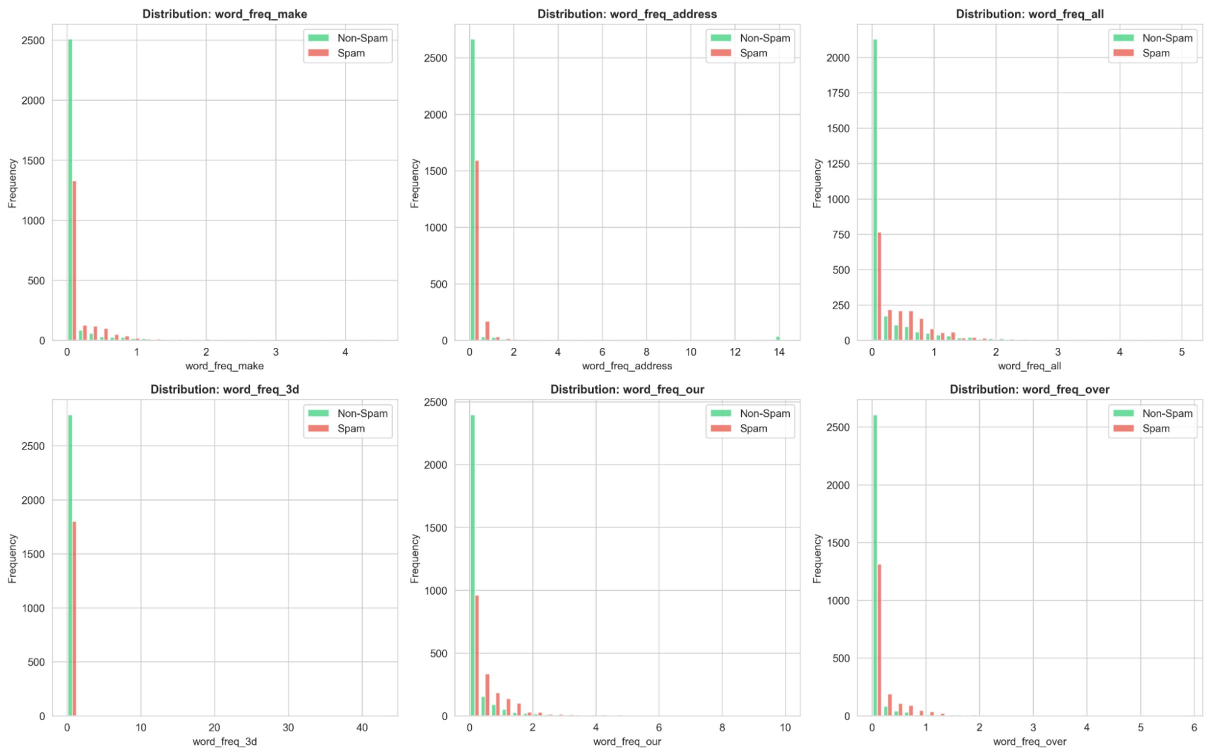

Figure 17. Distributions of selected word frequency features for spam and non-spam classes.

spam emails tend to exhibit higher frequencies for certain marketing-related words and special characters, particularly the exclamation mark (!) and dollar sign ($). Non-spam emails display more conservative feature values with fewer outliers.

We conducted correlation analysis among the first 20 features, which revealed moderate positive correlations between certain word frequency features, suggesting that some spam-related words tend to co-occur. However, our team did not detect severe multicollinearity issues that would necessitate feature elimination.

Our box plot analysis confirms the presence of numerous outliers in spam emails, particularly for character frequency features. We interpreted these outliers as representing aggressive spam tactics employing excessive use of special characters and capital letters to attract attention.

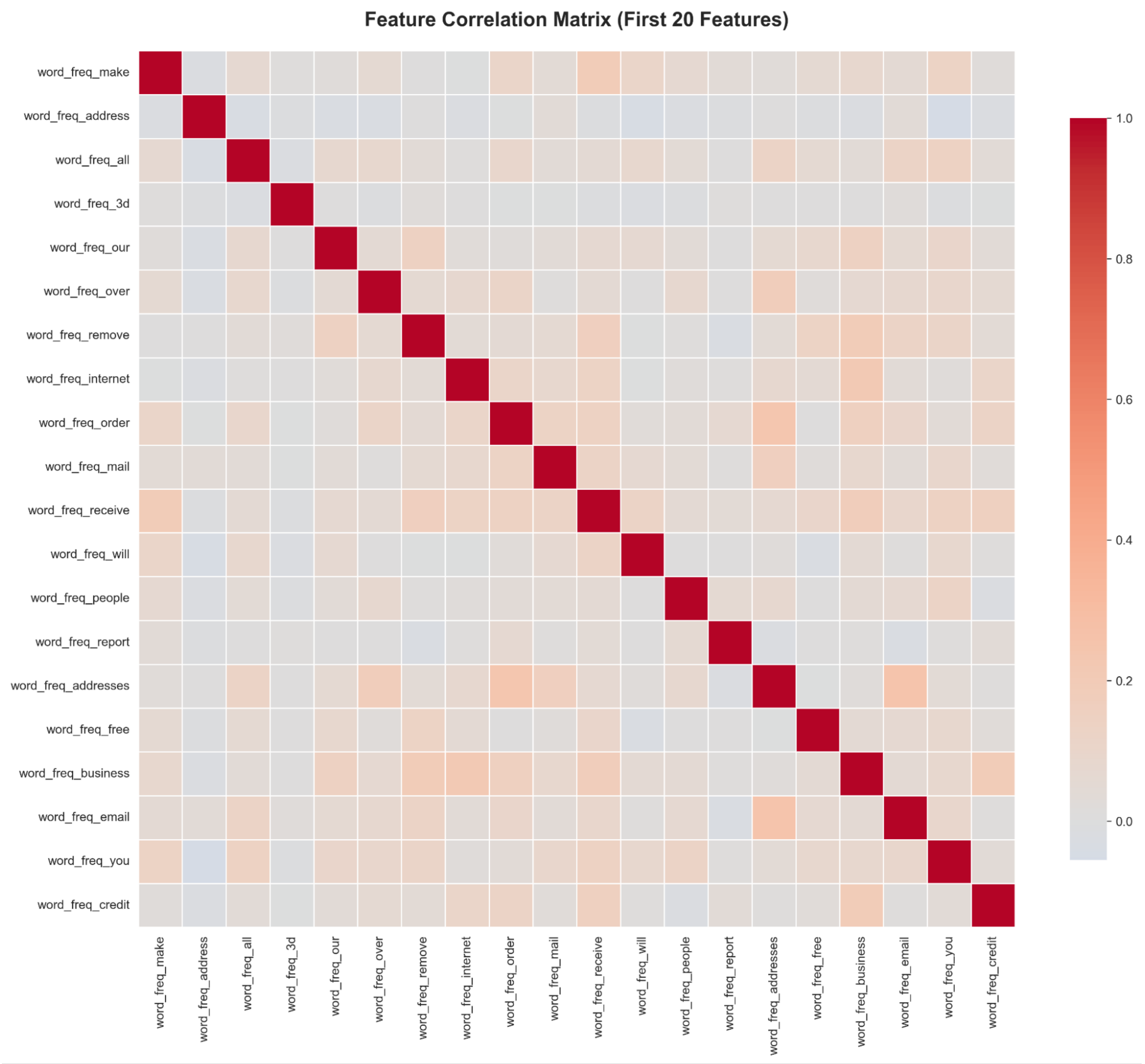

Figure 18. Correlation matrix of the first 20 features in the Spambase dataset.

3.1.3 Preprocessing Pipeline. Our team implemented the following preprocessing pipeline to ensure consistent and robust data preparation before model training.

Data Splitting. We partitioned the dataset into training (70%) and test (30%) sets using stratified sampling to maintain the original class distribution. This resulted in:

- **Training set**: 3,220 samples (39.41% spam)
- **Test set**: 1,381 samples (39.39% spam)

Feature Scaling. We applied `StandardScaler` to normalize feature distributions for Naive Bayes and Logistic Regression models. This transformation standardizes features to have zero mean and unit variance, which is crucial for distance-based algorithms and probabilistic models. Random Forest models were trained on unscaled data, as tree-based methods are inherently invariant to feature scaling.

Duplicate Handling. Our team made the strategic decision to retain the 394 duplicate rows because they represent genuine repetition patterns in email communication. Removing them could artificially alter the data distribution.

We chose not to apply additional feature engineering or dimensionality reduction techniques. All 57 original features were retained for model training to preserve the maximum amount of information.

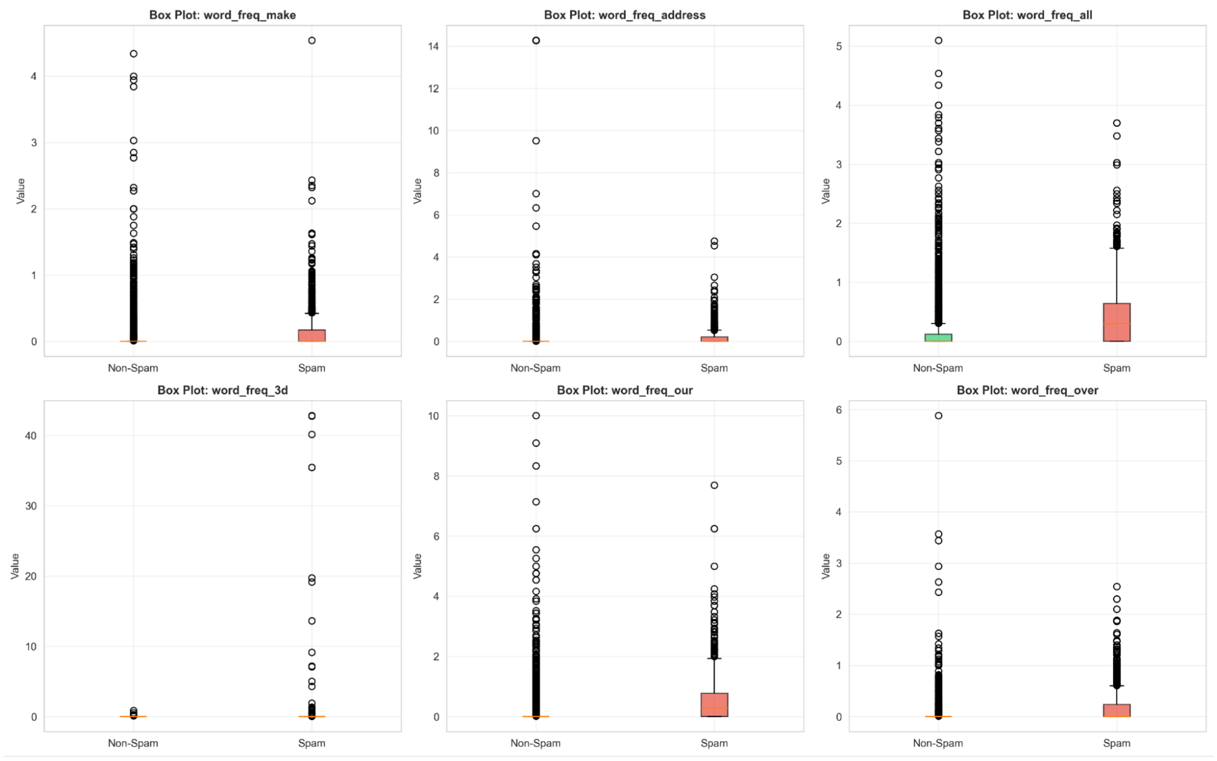

Figure 19. Box plots of selected word frequency features by spam vs non-spam classes.

3.2 Models and Methodology

3.2.1 Model Selection Rationale. Our team selected three traditional machine learning algorithms for this binary classification task, each representing different learning paradigms:

- **Naive Bayes (Gaussian)**: A probabilistic classifier based on Bayes' theorem with strong independence assumptions. We selected this for its computational efficiency and effectiveness on text-based features.
- **Logistic Regression**: A linear model using the logistic function to estimate class probabilities. We chose this for its interpretability, fast training, and strong baseline performance on binary classification tasks.
- **Random Forest**: An ensemble method combining multiple decision trees through bagging. We selected this for its ability to capture non-linear relationships and feature interactions without extensive feature engineering.

3.2.2 Hyperparameter Tuning Strategy. Our team performed hyperparameter optimization using `GridSearchCV` with 5-fold cross-validation to identify optimal configurations for each model. We selected the F1-score as the optimization metric to balance precision and recall, which is critical in spam detection where both false positives (legitimate emails marked as spam) and false negatives (spam reaching the inbox) carry costs.

Naive Bayes Hyperparameter Grid.

- **var_smoothing**: [1e-9, 1e-8, 1e-7, 1e-6]
- **Best parameters**: `var_smoothing = 1e-6`
- **Best CV F1-score**: 0.7998

Logistic Regression Hyperparameter Grid.

- **C** (inverse regularization strength): [0.01, 0.1, 1, 10, 100]
- **penalty**: [`l2`]
- **solver**: [`lbfgs`, `liblinear`]
- **Best parameters**: `C = 10`, `penalty = l2`, `solver = lbfgs`
- **Best CV F1-score**: 0.9046

Random Forest Hyperparameter Grid.

- **n_estimators**: [50, 100, 200]
- **max_depth**: [10, 20, 30, None]
- **min_samples_split**: [2, 5, 10]
- **min_samples_leaf**: [1, 2, 4]
- **Best parameters**: `n_estimators = 100`, `max_depth = 30`, `min_samples_split = 2`, `min_samples_leaf = 1`
- **Best CV F1-score**: 0.9385

Our hyperparameter search explored 4 configurations for Naive Bayes, 10 configurations for Logistic Regression, and 108 configurations for Random Forest, ensuring comprehensive exploration of the parameter space.

3.2.3 Implementation Details.

We implemented all models using `scikit-learn` version 1.3.0 in Python 3.10. Model training was conducted on the preprocessed training set with stratified 5-fold cross-validation for hyperparameter selection, followed by final model training on the entire training set using the best parameters.

We observed that model training times varied significantly:

- **Naive Bayes**: 0.0019 seconds (fastest)
- **Logistic Regression**: 0.0353 seconds (moderate)
- **Random Forest**: 0.3616 seconds (slowest, ~190× slower than Naive Bayes)

We note that the computational efficiency of Naive Bayes makes it particularly attractive for real-time spam filtering applications, while Random Forest's longer training time is offset by superior predictive performance.

3.3 Results

3.3.1 Model Performance Comparison.

We evaluated all three models on the held-out test set (1,381 samples) using multiple classification metrics. Table 3 summarizes the performance comparison across Accuracy, Precision, Recall, F1-Score, and ROC-AUC.

Key Performance Insights from Our Analysis:

- **Random Forest** achieved the best overall performance across most metrics, with an accuracy of 95.37%, precision of 95.28%, and ROC-AUC of 0.9867. The model correctly classified 1,317 out of 1,381 emails, with only 64 misclassifications (4.63% error rate).
- **Logistic Regression** demonstrated strong balanced performance with 92.76% accuracy and an excellent balance between precision (92.05%) and recall (89.34%), which our team recognized as a reliable choice for production environments requiring interpretability.
- **Naive Bayes** exhibited the highest recall (94.85%), successfully identifying 516 out of 544 spam emails. However, we observed this came at the cost of lower precision (70.01%), resulting in 221 false positives where legitimate emails were incorrectly flagged as spam.

3.3.2 Cross-Validation Consistency.

Our comparison between cross-validation and test set performance reveals excellent model generalization. Table 4 presents the F1-scores from cross-validation and test evaluation for each model.

We observed that all models showed minimal difference between cross-validation and test performance (<0.6%), indicating no overfitting and robust generalization to unseen data. Random Forest demonstrated the most consistent performance with only 0.19% variation.

3.3.3 Confusion Matrix Analysis.

Our detailed confusion matrix analysis reveals distinct error patterns for each model:

- **Naive Bayes**
  True Negatives: 616 **False Positives:** 221
  False Negatives: 28 **True Positives:** 516
  Error Pattern Observed: High false positive rate (26.4% of non-spam misclassified).
  Trade-off Identified: Aggressive spam detection catches 94.85% of spam but flags many legitimate emails.
- **Logistic Regression**
  True Negatives: 795 **False Positives:** 42
  False Negatives: 58 **True Positives:** 486
  Error Pattern Observed: Balanced errors with slightly more false negatives (58) than false positives (42).
  Trade-off Identified: More conservative spam detection with 89.34% spam catch rate.
- **Random Forest**
  True Negatives: 812 **False Positives:** 25
  False Negatives: 39 **True Positives:** 505
  Error Pattern Observed: Minimal errors in both categories.
  Trade-off Identified: Excellent balance with 92.83% spam detection and only 2.99% false positive rate.

3.3.4 ROC Curve Analysis.

Our ROC curve analysis demonstrates superior discriminative ability across all models:

- **Random Forest (AUC = 0.9867)**: We observed near-perfect separation between classes, with the curve closely hugging the top-left corner. The model achieves very high true positive rates even at extremely low false positive rates.
- **Logistic Regression (AUC = 0.9704)**: We found excellent discrimination with smooth probability estimates, making it well-suited for applications requiring adjustable probability thresholds.
- **Naive Bayes (AUC = 0.9338)**: Our analysis shows good discrimination despite lower precision, indicating that the probabilistic scores are well-calibrated even though the default threshold produces many false positives.

All three models significantly outperform random classification (AUC = 0.5), confirming their effectiveness for spam detection.

3.3.5 Feature Importance Analysis.

Our Random Forest feature importance analysis reveals the top 10 most discriminative features, as shown in Table 5.

We discovered that the exclamation mark (`!`) frequency is the single most important feature (11.80% importance), followed by the dollar sign ($) frequency (9.63%). These special characters are strong spam indicators, as spammers frequently use them to create urgency and highlight monetary offers. The word *"remove"* ranks third, likely appearing in spam unsubscribe instructions. Our team also noted that capital letter sequences play a significant role, as spam often employs excessive capitalization for emphasis.

Table 3. Performance comparison of different models on the test set (N = 1,381).

Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC
Naive Bayes	0.8197	0.7001	**0.9485**	0.8056	0.9338
Logistic Regression	0.9276	0.9205	0.8934	0.9067	0.9704
Random Forest	**0.9537**	**0.9528**	0.9283	**0.9404**	**0.9867**

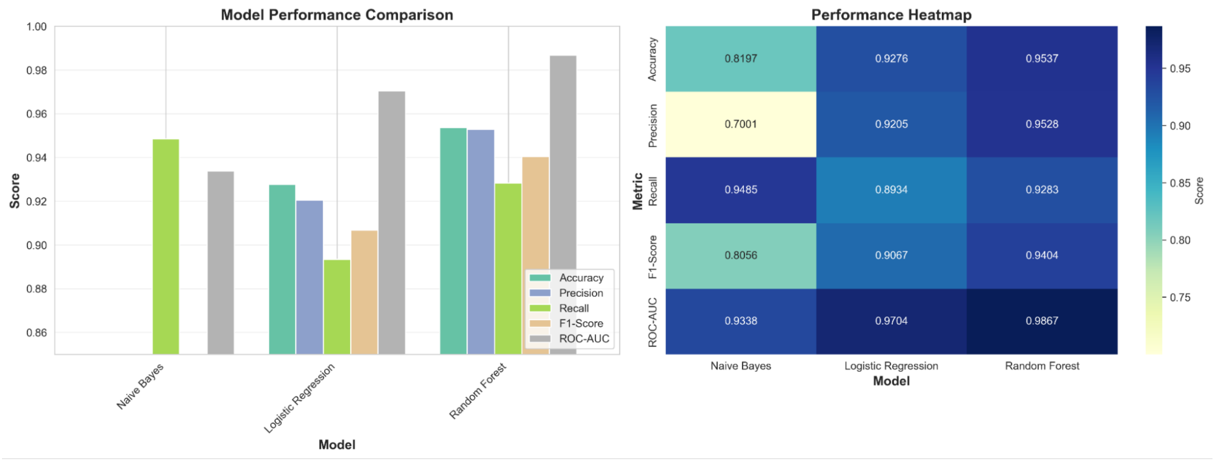

Figure 20. Comparison of model performance on Spambase using multiple evaluation metrics.

Table 4. Cross-validation vs. test performance (F1-score).

Model	CV F1-Score	Test F1-Score	Difference
Naive Bayes	0.7998	0.8056	+0.0058
Logistic Regression	0.9046	0.9067	+0.0021
Random Forest	0.9385	0.9404	+0.0019

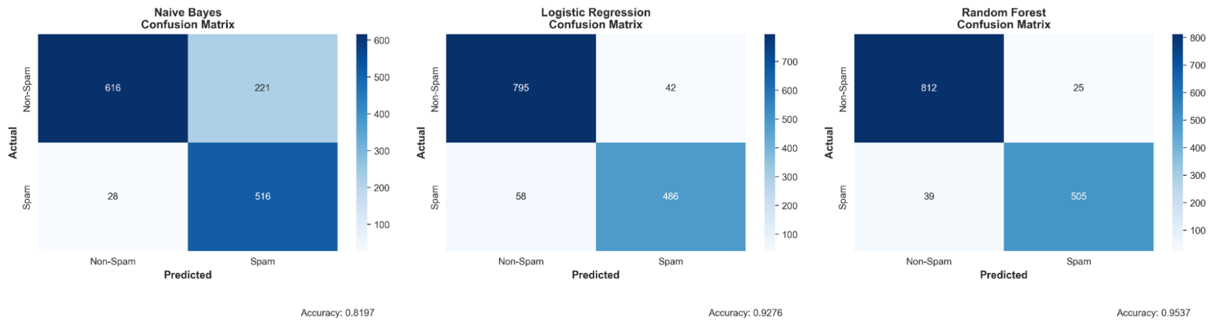

Figure 21. Confusion matrices for Naive Bayes, Logistic Regression, and Random Forest models on the Spam dataset.

3.4 Discussion

3.4.1 Comparative Model Analysis.

Our performance analysis clearly establishes **Random Forest** as the superior model for spam detection on this dataset, achieving 95.37% accuracy and 94.04% F1-score. We identified several factors that explain this dominance:

- **Non-linear Decision Boundaries.** Random Forest's ensemble of decision trees naturally captures complex non-linear relationships and feature interactions. We observed that spam detection involves intricate patterns where multiple features jointly determine spam likelihood (e.g., high $ frequency + high ! frequency + excessive capital letters = spam). Linear models such

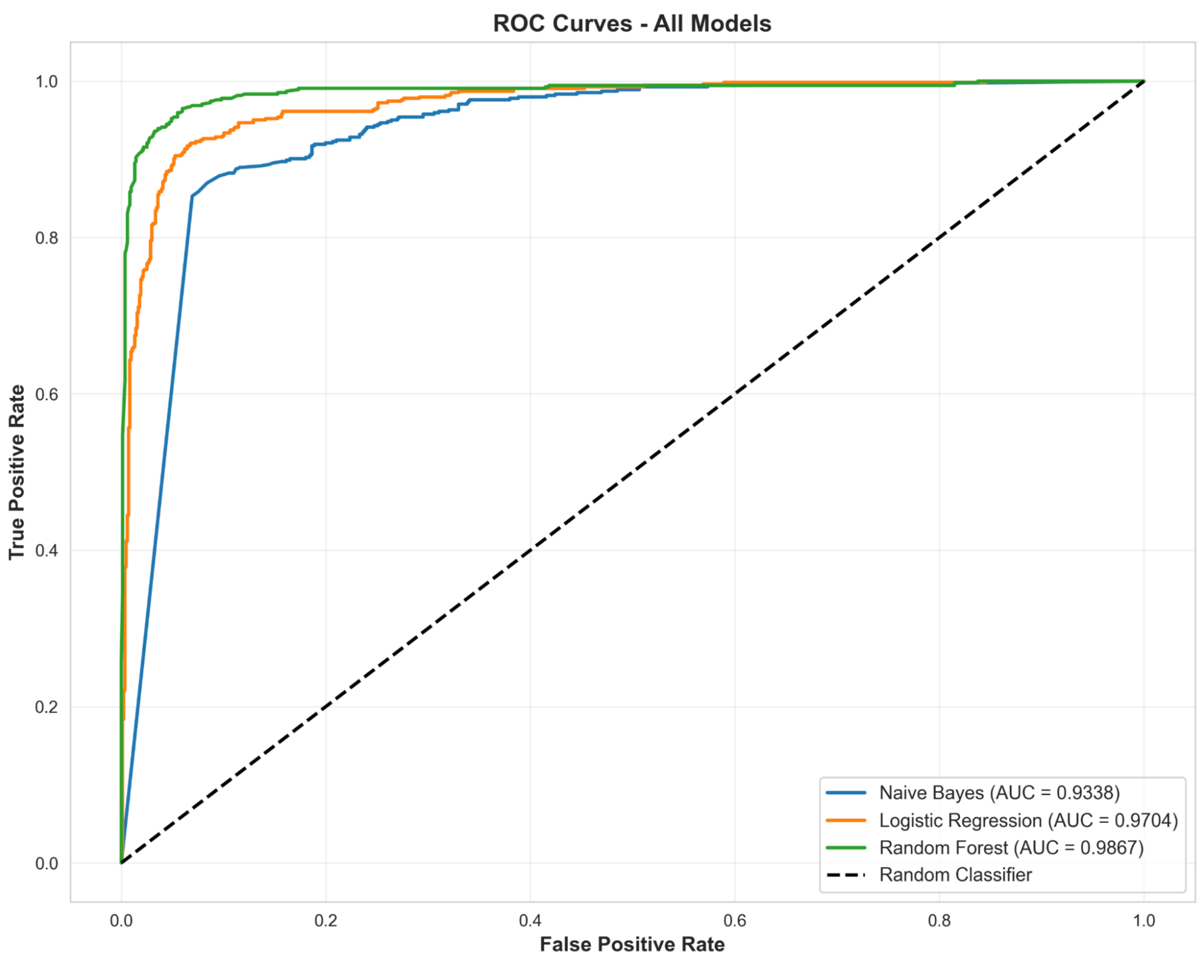

Figure 22. ROC curves of Naive Bayes, Logistic Regression, and Random Forest models, showing model discrimination performance.

Table 5. Top 10 most important features based on Random Forest.

Rank	Feature	Importance
1	char_freq_!	0.1180
2	char_freq_$	0.0963
3	word_freq_remove	0.0819
4	word_freq_free	0.0600
5	capital_run_length_total	0.0580
6	capital_run_length_average	0.0568
7	capital_run_length_longest	0.0550
8	word_freq_your	0.0484
9	word_freq_hp	0.0396
10	word_freq_you	0.0309

as Logistic Regression struggle with such interactions without explicit feature engineering.

- **Robustness to Outliers.** Our analysis revealed numerous outliers in spam feature distributions. Random Forest's tree-based structure is inherently robust to outliers, as splits are based on relative thresholds rather

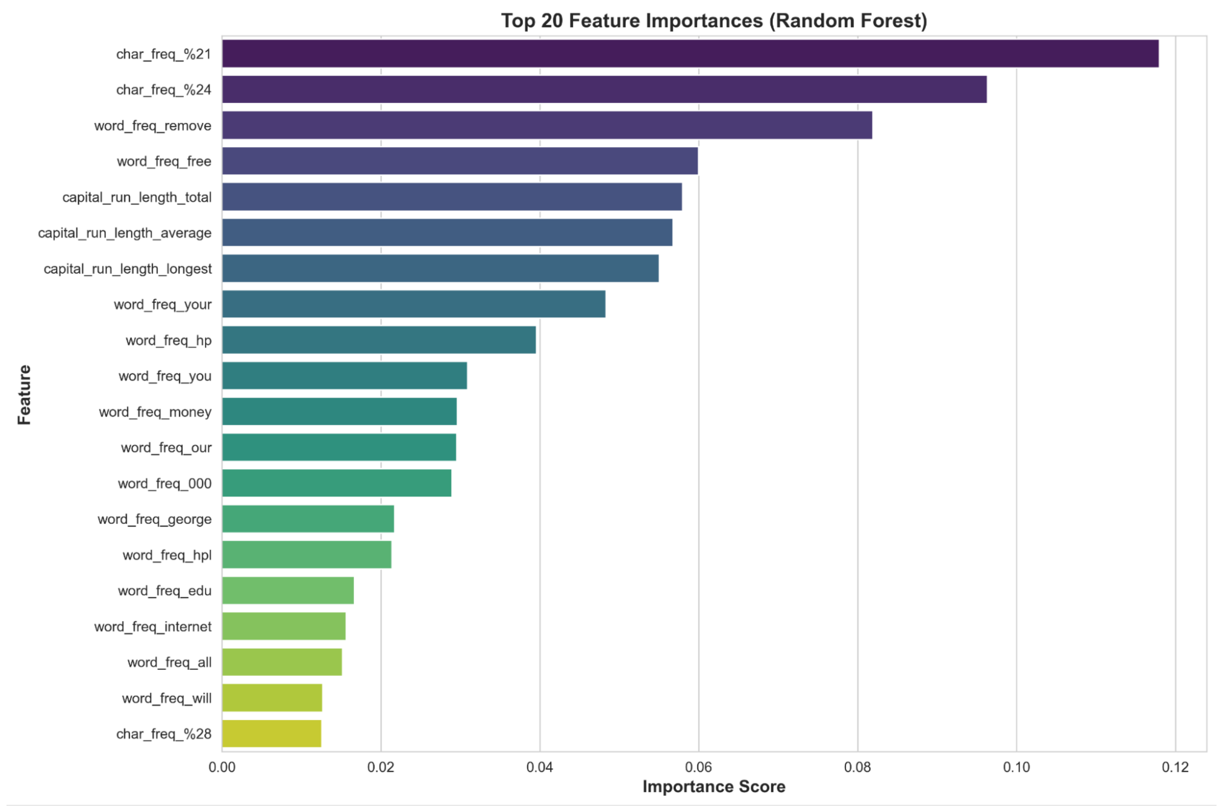

Figure 23. Top 20 most important features determined by the Random Forest model, ranked by importance scores.

than absolute distances. In contrast, Naive Bayes assumes Gaussian distributions and is therefore more sensitive to extreme values.

- **Feature Importance and Selection.** Random Forest implicitly performs feature selection through its splitting criteria, effectively down-weighting less informative features. This is particularly valuable given the 57-dimensional feature space, where not all features contribute equally.
- **Ensemble Diversity.** With 100 trees trained on bootstrapped samples, Random Forest averages out individual tree errors, leading to more stable and accurate predictions.

Logistic Regression's competitive performance (92.76% accuracy) demonstrates that spam detection contains substantial linear separability. We identified the following advantages of this model:

- **Interpretability:** Coefficient inspection reveals feature importance and direction of influence.
- **Speed:** 10× faster training than Random Forest (0.035s vs 0.362s).
- **Probability Calibration:** Produces well-calibrated probabilities useful for threshold tuning.
- **Simplicity:** Easier to deploy and maintain in production systems.

Naive Bayes's high recall (94.85%) makes it valuable for specific use cases despite lower overall accuracy. The model excels at catching spam but generates many false positives (221 legitimate emails flagged). We attribute this behavior to the following factors:

- **Feature Independence Assumption:** Naive Bayes assumes features are conditionally independent given the class. This assumption is violated in our dataset, where spam-related words frequently co-occur.
- **Gaussian Distribution Assumption:** The highly skewed feature distributions (many zeros, occasional extreme values) poorly match the Gaussian assumption used by the model.
- **Conservative Probability Estimates:** The model tends to be overconfident in spam predictions when multiple spam indicators are present simultaneously, leading to a high false positive rate.

3.4.2 Impact of Data Quality and Preprocessing.

Our team identified several dataset characteristics that significantly influenced model performance:

- **Class Balance Benefits.** The 60/40 non-spam/spam distribution proved ideal, requiring no specialized imbalance handling techniques. All models trained effectively without class weighting or sampling adjustments. We anticipate that more severe imbalance (e.g., 95/5) would likely degrade performance, particularly for Naive Bayes and Logistic Regression.
- **Feature Scaling Effects.** StandardScaler normalization was crucial for Naive Bayes and Logistic Regression, as these models are sensitive to feature magnitude. The wide range of scales (0 to 42.81) would otherwise cause features with larger ranges to dominate. Random Forest's invariance to monotonic transformations eliminated the scaling requirement.
- **Duplicate Retention.** We decided to retain 394 duplicate emails to preserve realistic email patterns. In practice, users frequently receive multiple copies of the same spam campaign, making duplicates informative rather than problematic.
- **Missing Value Absence.** The complete dataset (zero missing values) simplified preprocessing and ensured all 4,601 samples contributed to model training. We recognize that real-world email datasets often contain missing values requiring imputation strategies.

3.4.3 Error Pattern Analysis.

Our misclassification analysis reveals distinct error profiles for each model:

False Positive Analysis (Legitimate emails marked as spam).

- Naive Bayes: 221 false positives (26.4% of non-spam)
- Logistic Regression: 42 false positives (5.0% of non-spam)
- Random Forest: 25 false positives (3.0% of non-spam)

False positives are particularly problematic in spam filtering as they cause legitimate emails to be hidden or deleted. Random Forest's low 3% false positive rate makes it suitable for production deployment where user trust is critical.

False Negative Analysis (Spam reaching inbox).

- Naive Bayes: 28 false negatives (5.1% of spam)
- Logistic Regression: 58 false negatives (10.7% of spam)
- Random Forest: 39 false negatives (7.2% of spam)

Naive Bayes minimizes false negatives, catching 94.85% of spam. This aggressive filtering is appropriate for contexts where spam exposure is more costly than occasional false positives (e.g., corporate environments with IT support to retrieve misclassified emails).

3.4.4 Feature Importance Insights.

Our feature importance analysis provides actionable insights into spam characteristics:

- **Character-Based Indicators Dominate.** The top two features are special characters (! and $), collectively contributing 21.43% of model decisions. This suggests character-level features are more discriminative than word-level features for this dataset.
- **Marketing Language Patterns.** Words such as *"remove"*, *"free"*, and *"your"* appear in the top features, reflecting common spam tactics (unsubscribe instructions, free offers, personalized addressing).
- **Capital Letter Emphasis.** All three capital letter sequence features rank in the top seven, with a combined importance of 16.98%. Spammers' use of ALL CAPS for emphasis is a strong and easily detectable signal.
- **Context-Specific Features.** The presence of *"hp"* (Hewlett-Packard) in the top 10 indicates this dataset's origin from an HP email server, where company-specific terms help distinguish internal communications from external spam.

3.4.5 Computational Efficiency Considerations.

Our training time analysis reveals significant performance trade-offs:

- **Naive Bayes (0.0019s):** Suitable for real-time retraining on streaming email data.
- **Logistic Regression (0.0353s):** Acceptable for frequent model updates (e.g., daily retraining).
- **Random Forest (0.3616s):** 190× slower than Naive Bayes but still practical for batch retraining.

Our team recognizes that for large-scale email services processing millions of messages daily, inference time also matters. Logistic Regression offers the fastest prediction with simple linear computation, while Random Forest requires traversing 100 trees. However, modern computing infrastructure makes this difference negligible for most applications.

3.4.6 Limitations and Future Improvements.

Our team identified several limitations that warrant discussion:

- **Static Feature Set.** The 57 features are fixed and may not capture evolving spam tactics (e.g., image-based spam, unicode obfuscation). Incorporating additional features such as email metadata (e.g., sender reputation, temporal patterns) could further improve model performance.
- **Temporal Drift.** Spam characteristics evolve over time. Models trained on this historical dataset may degrade when deployed on current email traffic, necessitating periodic retraining to maintain performance.
- **Binary Classification Only.** The current models distinguish only between spam and non-spam, missing opportunities for finer-grained categories (e.g., marketing, phishing, malware) that would enable more nuanced filtering strategies.
- **No Ensemble Combination.** While Random Forest alone performs well, combining predictions from all three models through stacking or voting could potentially achieve even higher accuracy and robustness.

- **Threshold Optimization.** All models used the default 0.5 probability threshold. Application-specific threshold tuning (e.g., lowering the threshold to catch more spam at the cost of false positives) could better align with user or organizational preferences.

3.4.7 Practical Recommendations.

Based on our comparative analysis, we provide the following recommendations for practitioners:

For Maximum Accuracy (Production Systems).

- Deploy **Random Forest** with 100 estimators and `max_depth=30`.
- Expected performance: 95.4% accuracy, 95.3% precision, 92.8% recall.
- Training time remains acceptable for daily or weekly model updates.
- Suitable for email services prioritizing user experience and low false positive rates.

For High Recall (Security-Critical Environments).

- Deploy **Naive Bayes** with `var_smoothing=1e-6`.
- Expected performance: 94.9% recall with 26.4% false positive rate.
- Fastest training and inference, ideal for streaming scenarios.
- Suitable for corporate environments with IT support to handle false positives.
- Implement user feedback mechanisms to iteratively refine false positive handling.

For Balanced and Interpretable Filtering.

- Deploy **Logistic Regression** with `C=10` and L2 regularization.
- Expected performance: 92.8% accuracy with balanced precision and recall.
- Interpretable coefficients enable explaining filtering decisions to users.
- Fast enough for real-time retraining; suitable for user-facing applications requiring transparency.

For Hybrid Approach (Our Team's Innovative Recommendation).

- Use Naive Bayes for initial aggressive filtering (catching 94.9% of spam).
- Apply Random Forest to Naive Bayes's spam predictions to reduce false positives.
- Use Logistic Regression for borderline cases requiring user review.
- This cascading approach combines the strengths of all three models.

Feature Engineering Priorities.

- Monitor character frequencies (`!`, `$`) as primary spam indicators.
- Track capital letter sequence statistics.
- Incorporate sender reputation and email metadata.
- Consider temporal features (e.g., time of day, day of week) for future iterations.

3.4.8 Dataset-Specific Behavior.

Through our analysis, we observed that the UCI Spambase dataset exhibits characteristics that favor ensemble methods:

- High dimensionality (57 features) benefits Random Forest's ability to explore diverse feature subspaces.
- Feature interactions between character frequencies and word frequencies are naturally captured by tree splits.
- Non-linear boundaries between spam and non-spam regions favor tree-based models over linear classifiers.
- Balanced classes enable all models to train effectively without specialized techniques.

We recognize that this dataset represents an ideal scenario for supervised learning: clean data, balanced classes, informative features, and sufficient samples. Real-world spam filtering faces additional challenges including adversarial adaptation (spammers evolving to evade filters), multilingual content, and imbalanced data streams. Despite these differences, we believe the models and methodologies demonstrated here provide a strong foundation transferable to production spam filtering systems.

3.5 Conclusion

Our comprehensive comparative analysis of three traditional machine learning algorithms on the UCI Spambase dataset demonstrates that **Random Forest** achieves superior spam detection performance, with 95.37% accuracy, 94.04% F1-score, and 98.67% ROC-AUC. The model's ability to capture non-linear feature interactions and handle outliers makes it well-suited for the complex patterns inherent in spam detection.

However, model selection should consider application-specific requirements. **Naive Bayes** excels in recall-critical scenarios where missing spam is costly, while **Logistic Regression** provides an optimal balance of performance, interpretability, and computational efficiency for production deployments requiring transparency.

Key findings from our analysis include:

- Character frequency features (`!`, `$`) are the strongest spam indicators.
- Capital letter sequences contribute significantly to spam detection.
- All models generalize well with minimal overfitting ($<$0.6% CV–test gap).
- The relatively balanced class distribution enables effective training without specialized techniques.
- Feature scaling is essential for probabilistic and linear models but unnecessary for tree-based methods.

For practitioners implementing spam filtering systems, we recommend starting with **Random Forest** for maximum accuracy, maintaining **Logistic Regression** as an interpretable baseline, and considering **Naive Bayes** for recall-sensitive applications. Continuous monitoring and periodic retraining remain essential as spam tactics evolve over time to ensure sustained model effectiveness.

4 Conclusion and Future Work

4.1 Conclusion

In this project, our team conducted a systematic comparative analysis of traditional machine learning models across two distinct datasets: the **UCI Heart Disease** dataset (small, mixed-type, tabular) and the **UCI Spambase** dataset (large, high-dimensional, text frequency-based).

For the **Heart Disease** dataset, **Random Forest** and **SVM** achieved the highest accuracy, with Random Forest excelling in recall—making it well-suited for clinical applications where missing positive cases is costly. **KNN** achieved the highest ROC-AUC but exhibited greater sensitivity to parameter choices, indicating the need for careful tuning.

For the **Spambase** dataset, **Random Forest** delivered the strongest overall performance across Accuracy, F1-score, and ROC-AUC. **Logistic Regression** provided a fast and interpretable baseline, while **Naive Bayes** achieved the highest recall, making it valuable in security-critical contexts where detecting spam is prioritized over avoiding false positives.

Our results demonstrate that **data characteristics fundamentally shape preprocessing requirements, model performance, and evaluation priorities**. The same algorithm can behave very differently across datasets, underscoring the importance of *tailored workflows* rather than one-size-fits-all approaches.

4.2 Future Work

In future work, we plan to explore more advanced ensemble methods, such as boosting algorithms (e.g., **XGBoost**, **LightGBM**), to further improve model performance. We will also develop **feature engineering strategies** tailored to each dataset.

For the Heart Disease dataset, this may involve creating interaction features or transforming existing variables to better capture clinical relationships. For the Spambase dataset, we aim to enrich the feature set with additional text-related features or apply feature selection methods to identify the most informative signals.

Additionally, we intend to **adjust classification thresholds based on real-world error costs**. For example, in medical prediction, missing a positive case is more serious than a false alarm, while in spam detection, marking legitimate emails as spam is typically more costly. By tuning thresholds accordingly, the models can better align with the requirements of different application domains.

Finally, we plan to **evaluate the robustness of our models to temporal data drift** and apply **explainability methods** to make model decisions more transparent and trustworthy, especially in sensitive domains such as healthcare.

5 Code Availability

The full source code supporting this study is included in the `code/` folder submitted together with this paper. It contains preprocessing, model training, and evaluation scripts to ensure full reproducibility.

References

[1] Dua, D. and Graff, C. 2019. *UCI Machine Learning Repository*. University of California, Irvine. Retrieved from https://archive.ics.uci.edu.

[2] Shreekant, G. *Heart-Disease-Prediction-using-Machine-Learning* [GitHub repository]. Retrieved from https://github.com/g-shreekant/Heart-Disease-Prediction-using-Machine-Learning.

[3] Hopkins, M., Reeber, E., Forman, G., & Suermondt, J. (1999). Spambase [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C53G6X.

[4] Singh, A. (n.d.). Binary-Classification-using-ML: Binary Classification using Machine Learning. Retrieved from https://github.com/abhigyan2311/Binary-Classification-using-ML.