Data Ming Individual Project Report

Spam Classification Analysis and Optimization Using Machine Learning Models

Zhiwen Tian (June)[†]

[†]*Section: 1002*
[†]*Student ID: 2230033036*

Abstract

This report discusses the training and evaluation of various machine learning classification models based on Spambase data sets, aiming at conducting an efficient spam classifier. I choose to use K-nearest neighbor (KNN), custom perceptron, logistic regression, support vector machine (SVM), decision tree (DT), random forest, and gradient lifting models. Each model was hypertuned through grid search, and the performance of the model was evaluated using 5-fold cross-validation. The evaluation measures include Accuracy, Precision, Recall, F-Score, and AUC Score, and the results are visually analyzed for more intuitive evaluation and improvement of the model.

The experimental results show that the gradient lifting model has excellent performance in all indexes, with an accuracy of 0.9400 and an AUC of 0.9783. In addition, random forests and support vector machines also perform very well, especially in terms of AUC and accuracy similar to gradient lift models. In contrast, KNN and custom perceptrons perform slightly less well when dealing with nonlinear problems. Overall, the gradient lifting model is chosen as the best classifier, which is very suitable for high-precision spam detection tasks.

This experiment verifies the advantages of the ensemble learning method in the classification task, and provides a useful reference for further optimization of model performance in the future.

Keywords: *Spam Classification, Machine Learning, Hyperparameter Tuning, Ensemble Learning, Model Evaluation*

1. Introduction

1.1. Background

In the digital era, detecting and preventing spam has become a critical aspect of information security. This project leverages the SpamBase dataset from the UCI Machine Learning Repository, employing a range of classification models to analyze and classify email data with the goal of accurately distinguishing spam from legitimate emails. By evaluating and comparing model performance, we aim to identify the most effective classification method for this dataset, followed by an in-depth analysis and discussion of the results.

1.2. Objective

The objective of this project is to apply machine learning techniques to classify the SpamBase dataset effectively and to develop a robust classifier for spam prediction. Multiple models were trained, tuned, and evaluated to ensure high accuracy and stability of the classifier. Ultimately, the performance of each model is compared, and the best-performing model is selected as the optimal solution for this task.

2. Workflow

2.1. Diagram

Include a workflow diagram to visualize the process, showing data input, preprocessing, model training, evaluation, and analysis steps. The diagram of wolkflow is Figure 1.

2.2. Data Preprocessing

2.2.1. Data Loading

The Spambase dataset was downloaded from the UCI Machine Learning Repository using the ucimlrepo library. This dataset contains 57 continuous features with no missing values. The target variable y is binary, indicating whether an email is spam.

2.2.2. Data Standardization

Z-score standardization was applied to eliminate feature scaling differences, ensuring each feature has a mean of 0 and a standard deviation of 1.

2.2.3. Outlier Detection and Removal

Outliers were identified using the Z-score method, with a threshold of 3. Only data points within this threshold for all features were retained for model training.

2.2.4. Data Alignment

After removing outliers, the indices of the target variable y were aligned with those of the feature data X to prevent mismatches.

2.3. Model Selection

In order to realize the effective classification of spam, the following seven commonly used machine learning models are selected in this experiment:

- **K-Nearest Neighbor (KNN)**
  Classification of new data points based on distance measures, by calculating the distance from the test point to the training point. Commonly used distance measure:

$$d(x_i, x_j) = \sqrt{\sum_{k=1}^{n} (x_{ik} - x_{jk})^2}$$

  where $d(x_i, x_j)$ is the Euclidean distance between points x_i and x_j in an n-dimensional space.

- **Perceptron**
  Linear classification model with updated weights using gradient descent. Given features x and weights w, the model's prediction is:

$$y = \text{sign}(w \cdot x + b)$$

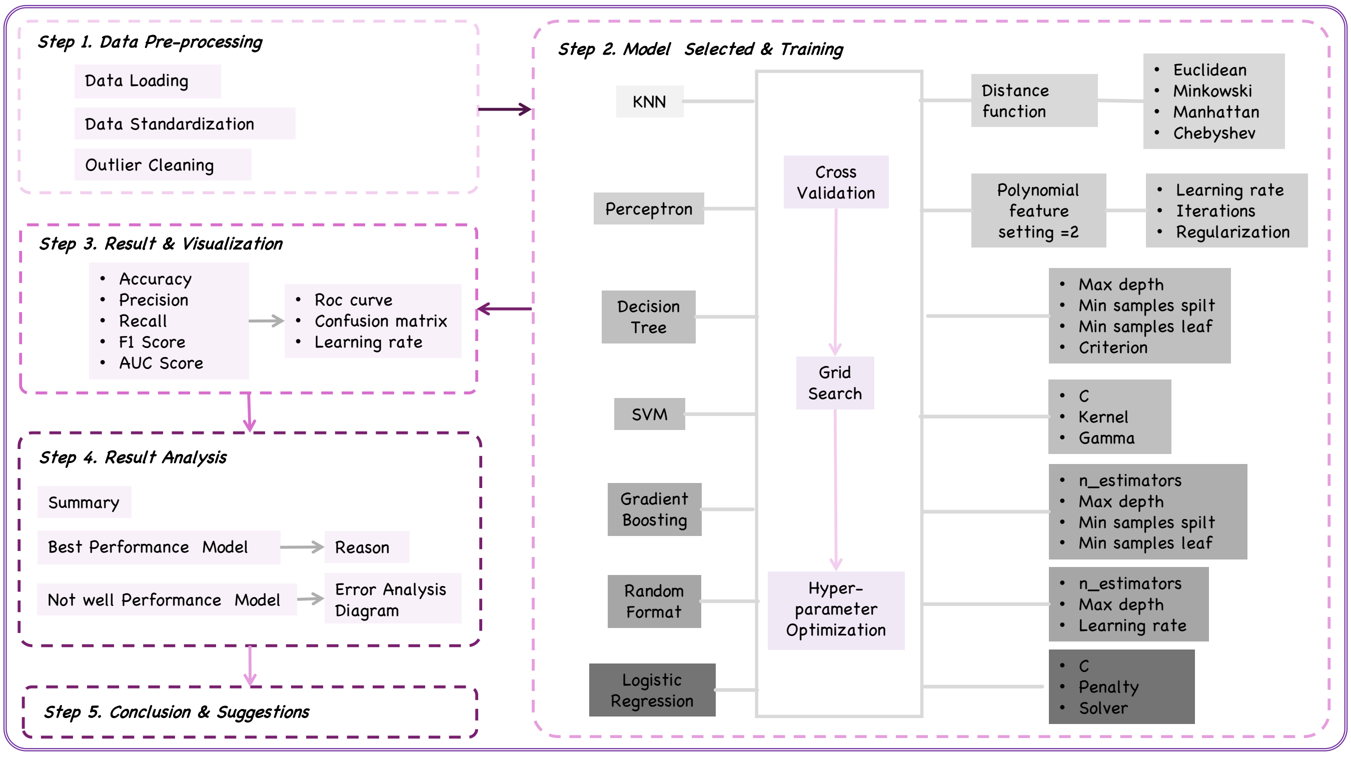

Figure 1. Workflow

where b is the bias term. Weights are updated using:

$$w \leftarrow w + \eta(y_{\text{true}} - y_{\text{pred}})x$$

where η is the learning rate.

- **Decision Tree**
  A tree-like model that selects split points by maximizing information gain (or minimizing the Gini impurity). For Gini impurity:

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

  where p_i is the probability of a sample belonging to class i, and C is the number of classes.

- **Logistic Regression**
  A linear classification model that predicts probabilities using a logistic function. The probability that an instance x belongs to class 1 is:

$$P(y = 1|x) = \frac{1}{1 + e^{-(w\cdot x+b)}}$$

  where w and b are the model parameters.

- **Gradient Boosting**
  An ensemble model that minimizes the error by combining multiple weak learners. For each iteration m, the model updates by adding a new tree $h_m(x)$ to fit the residuals:

$$F_m(x) = F_{m-1}(x) + \eta h_m(x)$$

  where η is the learning rate and $F_m(x)$ is the current model.

- **Random Forest**
  An ensemble of decision trees where each tree is trained on a random subset of features and samples. The prediction is obtained by majority voting for classification:

$$\hat{y} = \text{mode}\{T_1(x), T_2(x), \dots, T_k(x)\}$$

  where $T_i(x)$ represents the prediction of the i-th tree.

- **Support Vector Machine (SVM)**
  A model that finds the hyperplane maximizing the margin between classes. For linear SVM, the decision function is:

$$f(x) = w \cdot x + b$$

  The objective is to maximize $\frac{1}{\|w\|}$ while ensuring all points are correctly classified, often with a slack variable ξ for non-separable cases:

$$\min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \xi_i$$

  where C is the regularization parameter.

2.4. Model training and evaluation

2.4.1. Model initialization

Define and initialize the relevant hyperparameters for each model. For example, KNN sets k-value ranges and distance measures, decision trees set depth and splitting criteria, logistic regression sets regularization types, and so on.

2.4.2. Model training and evaluation

Train and test each model using 5-fold cross-validation ($'KFold'$, $k = 5$) to ensure the robustness of the evaluation results. Cross validation divides the data set into training set and verification set, thus reducing the interference of training errors on the results. The following indicators were used to evaluate the model performance:

- **Accuracy**
  The proportion of correctly classified instances over the total instances:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

  where:
  - TP = True Positives
  - TN = True Negatives

- FP = False Positives
- FN = False Negatives

- **Precision**
  The proportion of predicted positive samples that are actually positive:
$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$
- **Recall**
  The proportion of actual positive samples that are correctly predicted:
$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$
- **F-Score**
  The harmonic mean of Precision and Recall, which balances the two:
$$\text{F-Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$
- **AUC Score**
  The area under the ROC curve, reflecting the model's ability to distinguish between classes. It is generally calculated using the trapezoidal rule on the ROC curve.

2.4.3. Result recording and visualization

Record the average accuracy, accuracy, recall, F1 score, and AUC value of each model in cross-validation, and draw visual charts including confusion matrix, ROC curve, and accuracy variation chart to visually show the classification effect of the model.

2.5. Result Analysis

2.5.1. Best model selection

Comprehensively analyze the performance of each model to determine the best model on the data set. The gradient lifting model performs well in many indexes, especially in AUC and accuracy, showing strong classification ability.

2.5.2. Error analysis

Error analysis is performed on models with weak performance. By analyzing the error sources of these models, we can provide ideas for further improvement.

2.6. Conclusion and Suggestion

After comparative analysis, it is concluded that the gradient lifting model is the best model for this spam classification, with high accuracy and AUC value, which is suitable for high-precision spam detection. At the same time, random forest and support vector machines also have strong classification performance and can be used as alternative solutions. Future research could try:

1. More complex integration methods: Explore combinations of different integration models to further improve classification.
2. Feature Engineering: Find more features that help with classification and improve the generalization ability of the model.
3. Adjust the balance of the data set: If there is a class imbalance problem, oversampling, undersampling and other methods can be used to improve the performance of the model on a few classes.

3. Data Preprocessing

Data preprocessing is essential for ensuring model performance and stability. This project's preprocessing includes data standardization, outlier removal, and index alignment. The steps are detailed as follows:

3.1. Dataset Acquisition

The Spambase dataset was obtained from the UCI repository using the `ucimlrepo` library. The data was split into features X and target y, where y represents binary classification labels (spam or non-spam).

3.2. Data Standardization

To remove scale differences between features, X was standardized, ensuring each feature has a mean of zero and unit variance, calculated as:
$$X_{\text{standardized}} = \frac{X - \mu_X}{\sigma_X} \tag{1}$$
where μ_X is the mean of feature X, and σ_X is the standard deviation. Standardization improves model convergence and performance.

3.3. Outlier Detection and Removal

Outliers were removed using the Z-score method with a threshold of 3. For each standardized feature, Z-scores were calculated as:
$$Z_i = \frac{X_i - \mu_X}{\sigma_X} \tag{2}$$
where Z_i denotes the Z-score for the i-th data point. Points with $|Z| > 3$ were excluded, reducing the influence of extreme values.

3.4. Index Alignment and Target Cleaning

After outlier removal, y was aligned with the cleaned X indices to maintain consistency. Non-numeric entries in y were filtered, and y was cast to integers, ensuring compatibility with the model.

3.5. Post-Processing Data Quality

Following preprocessing, X and y meet the following criteria:

- **No Outliers**: Outliers were removed via Z-score filtering.
- **Standardized Features**: Each feature has zero mean and unit variance.
- **Index Consistency**: Indices of X and y are aligned, ensuring data integrity.

These preprocessing steps provide a robust foundation for training and evaluating classification models.

4. Models Adopted

Custom-Implemented Models

4.1. K-Nearest Neighbors (KNN)

4.1.1. Model Design

In this project, the K-Nearest Neighbors (KNN) algorithm was selected as the primary classification model. KNN is an instance-based learning algorithm that classifies samples by calculating the distance between the sample to be predicted and the training samples. The key aspect of model design lies in choosing appropriate parameters, including the number of neighbors k and the distance metric. To optimize KNN's performance, I implemented grid search, systematically exploring combinations of different distance metrics (Euclidean, Manhattan, Chebyshev, and Minkowski) and multiple k values to find the optimal classification parameters.

4.1.2. Training Process

To enhance generalization and prevent overfitting, we applied **5-fold cross-validation** during model training. The steps are as follows:

1. Data Splitting
   The data was randomly divided into five subsets. For each run, four subsets were used as the training set and one as the validation set, repeated five times.
2. Model Training
   For each fold, KNN classification was performed using the selected k value and distance metric on the training data.
3. Model Validation
   Predictions were made on the validation set, and accuracy was calculated. The average accuracy across the five runs was used as the performance metric for each parameter combination.

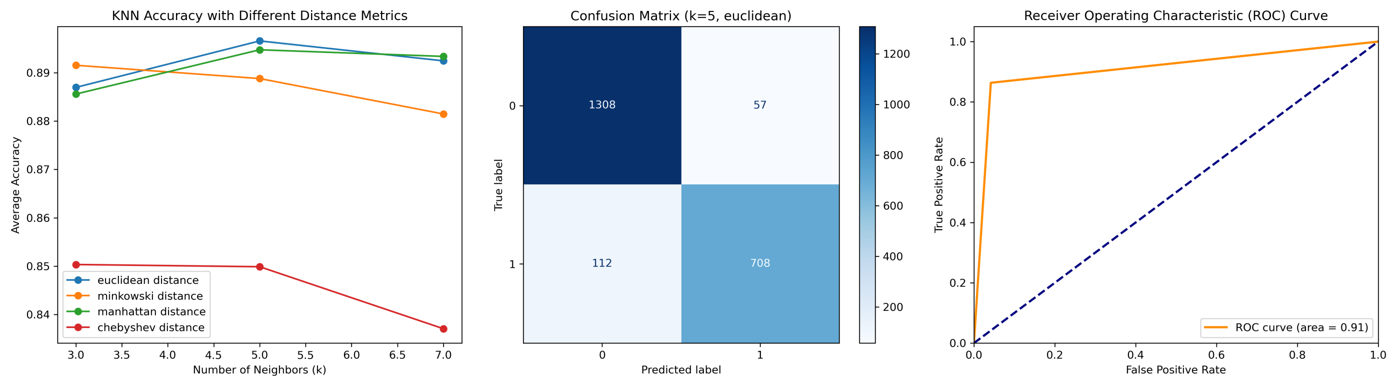

Figure 2. Results of KNN analysis

4. Result Recording
   The average accuracy for each k-distance metric combination was recorded for subsequent parameter selection.

4.1.3. Parameter Selection

To optimize the KNN classifier, we focused on tuning two main parameters:

1. **Number of Neighbors (k)**
   I evaluated different values for k (3, 5, and 7) to determine the optimal choice .
2. **Distance Metric**
   I tested four distance metrics, including Euclidean, Manhattan, Chebyshev, and Minkowski distances.
   - **Euclidean Distance**
     Measures straight-line distance, suitable for general similarity calculations.

$$d_{\text{Euclidean}}(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

   - **Manhattan Distance**
     Computes the sum of absolute differences across dimensions, effective for data with strong axis-oriented features

$$d_{\text{Manhattan}}(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

   - **Chebyshev Distance**
     Measures the maximum absolute difference along any coordinate dimension, useful when constraints on directions are present

$$d_{\text{Chebyshev}}(x, y) = \max_{i=1}^{n} |x_i - y_i|$$

   - **Minkowski Distance**
     A generalized distance metric parameterized by p, which includes Euclidean and Manhattan as special cases.

$$d_{\text{Minkowski}}(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}}$$

4.1.4. Optimal Parameter Combination

A grid search with **5-fold cross-validation** was conducted to assess the accuracy of each k-distance metric combination. The combination of $k = 5$ and **Euclidean distance** produced the best performance across all tests, yielding the highest average accuracy.

4.1.5. Results and Visualization

Evaluation Results
- **Accuracy**: 0.9227
- **Precision**: 0.9255
- **Recall**: 0.8634
- **F1 Score**: 0.8934
- **AUC Score**: 0.9108

Visualization Analysis: Figure 2. Results of KNN Analysis

4.2. Perceptron

4.2.1. Model Design

In this project, a Weighted Perceptron model was used, incorporating polynomial feature expansion (degree = 2) to enhance the model's nonlinear fitting capacity. This design includes regularization and positive sample weighting to optimize classification performance. During training, momentum and learning rate decay strategies were employed to accelerate convergence and prevent overfitting.

4.2.2. Training Process

The model was trained using 5-fold cross-validation to ensure effective parameter selection and generalization. The specific steps are as follows:

1. Data Splitting
   The dataset was randomly divided into five subsets, with each fold using four subsets for training and one subset for validation, repeated five times to assess model stability across different datasets.
2. Model Training
   During training, weights and bias were adjusted based on misclassified samples, with the following strategies:
   - **Momentum**: Accelerated convergence and reduced training time.
   - **Learning Rate Decay**: Gradually decreased the learning rate over iterations to prevent large adjustments in later stages.
   - **Positive Sample Weighting**: Higher weights were assigned to positive samples, enhancing performance on imbalanced data.
3. Performance Evaluation
   Accuracy was calculated on the validation set, and the average across five folds was used to evaluate each parameter combination's performance.

4.2.3. Parameter Selection

Grid search was used to optimize the learning rate, number of iterations, and regularization parameter, with the following candidates:

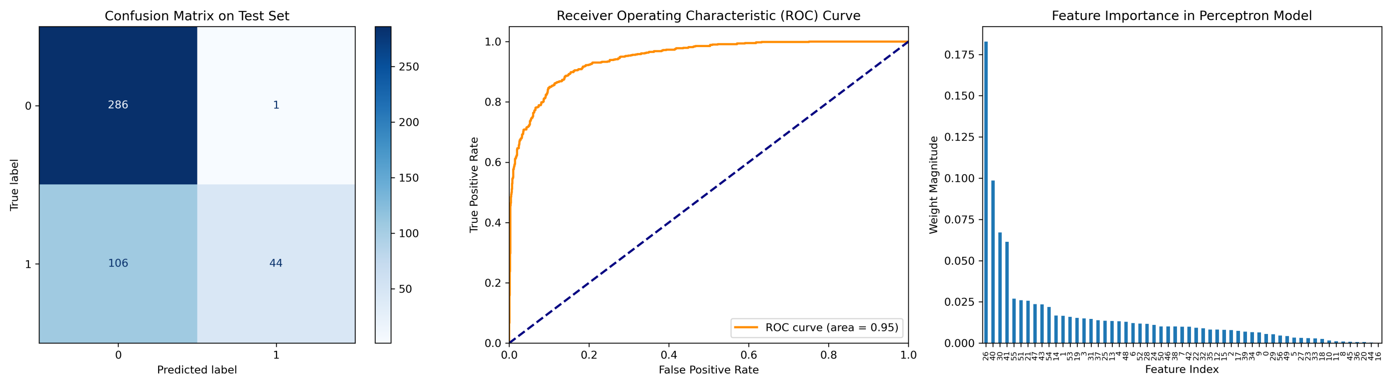

Figure 3. Results of Preceptron analysis.

- Learning Rate: 0.001, 0.01, and 0.1
- Number of Iterations: 500, 1000, and 2000
- Regularization Parameter: 0.001, 0.01, and 0.1

The optimal parameters were found to be:

- Learning Rate: 0.001
- Number of Iterations: 1000
- Regularization Parameter: 0.01

This combination achieved an average cross-validation accuracy of 0.9034.

4.2.4. Results and Visualization

Evaluation Results

- **Accuracy**: 0.8517
- **Precision**: 0.7455
- **Recall**: 0.9183
- **F1 Score**: 0.8230
- **AUC Score**: 0.9462

Visualization Analysis: Figure 3. Results of Preceptron Analysis

These metrics show high recall for positive samples, though precision is slightly lower, indicating some false positives in positive sample recognition.

Library-Implemented Models

4.3. Decision Tree

4.3.1. Model Design

In this project, I used a Decision Tree model for classification. Decision trees split the dataset into subsets based on feature conditions, creating interpretable decision paths. Hyperparameters were optimized via grid search to enhance accuracy and generalization performance.

4.3.2. Parameter Selection

Through grid search, I optimized the following hyperparameters:

- **Maximum Depth (max_depth)**: Limits the maximum depth of the tree, with candidate values of None, 10, 20, and 30.
- **Minimum Samples per Leaf (min_samples_leaf)**: Controls the minimum number of samples required at each leaf node, with candidate values of 1, 2, and 4.
- **Minimum Samples for Split (min_samples_split)**: Sets the minimum number of samples required to split an internal node, with candidate values of 2, 5, and 10.
- **Split Criterion (criterion)**: The split criterion, including “gini” and “entropy”.

The optimal hyperparameters were found to be:

- **criterion**: gini
- **max_depth**: 10
- **min_samples_leaf**: 1
- **min_samples_split**: 2

This combination achieved the best accuracy in cross-validation, providing stable classification performance.

4.3.3. Training Process

The Decision Tree model was trained using 5-fold cross-validation to ensure generalization. The training steps are as follows:

1. **Data Splitting**: The dataset was randomly divided into five subsets. For each run, four subsets were used for training and one for validation, repeated five times.
2. **Model Training**: The Decision Tree model was trained on each fold using the optimal hyperparameters.

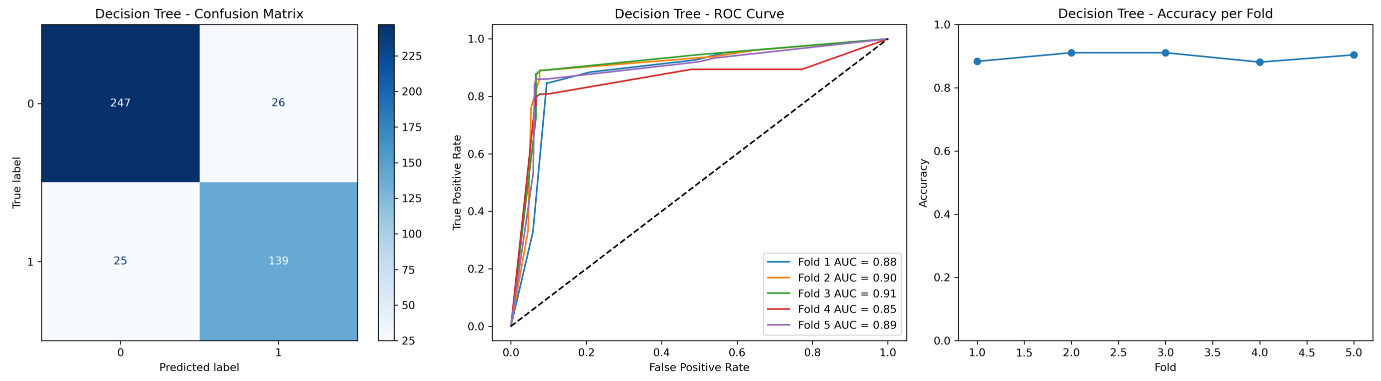

Figure 4. Results of Decision Tree analysis (source: Decision Tree results).

3. **Performance Evaluation**: For each validation set, accuracy, precision, recall, F1 score, and AUC were calculated. The average across the five folds was recorded as the model's final performance.

4.3.4. Results and Visualization

Evaluation Results

- **Accuracy**: 0.8979
- **Precision**: 0.8736
- **Recall**: 0.8477
- **F1 Score**: 0.8601
- **AUC**: 0.8862

Visualization Analysis: Figure 4. Results of Decision Tree Analysis

These metrics indicate a high accuracy and balanced performance in the spam classification task, though the AUC is slightly lower compared to other models.

4.4. SVM

4.4.1. Model Design

In this project, I used a Support Vector Machine (SVM) as the classification model, optimizing hyperparameters through grid search. The SVM model effectively separates samples in high-dimensional space and, by using a non-linear kernel function (such as the RBF kernel), establishes complex decision boundaries to improve classification performance.

4.4.2. Parameter Selection

To optimize the SVM model, I performed grid search on the following parameters:

- **Regularization Parameter (C)**: Controls model complexity, with candidate values of 0.1, 1, 10, and 100.
- **Kernel Function (kernel)**: Linear (linear), RBF (rbf), and polynomial (poly) kernels were tested.
- **Kernel Coefficient (gamma)**: Controls the influence range for the RBF and polynomial kernels, with candidates "scale" and "auto".

The optimal parameters were:

- **C**: 1
- **gamma**: scale
- **kernel**: rbf

This parameter combination achieved the highest accuracy in cross-validation, ensuring optimal classification performance.

4.4.3. Training Process

The SVM model was trained using 5-fold cross-validation, recording performance on each fold. The steps are as follows:

1. **Data Splitting**: The dataset was randomly divided into five subsets. For each run, four subsets were used for training and one for validation to assess model performance on different data splits.
2. **Model Training**: The SVM model with the optimal parameters was fitted on each training set.
3. **Performance Evaluation**: Accuracy, precision, recall, F1 score, and AUC were calculated on each validation set, with the average across five folds recorded as the model's final performance.

4.4.4. Results and Visualization

Evaluation Results

- **Accuracy**: 0.9240
- **Precision**: 0.9356
- **Recall**: 0.8559
- **F1 Score**: 0.8939
- **AUC**: 0.9644

Visualization Analysis: Figure 5. Results of SVM

These metrics indicate that the SVM model achieved high accuracy and AUC on the spam classification task, confirming its strong performance.

4.5. Random Forest

4.5.1. Model Design

In this project, I used a Random Forest model for classification. Random Forest is an ensemble method that constructs multiple decision trees and combines their predictions through voting, improving accuracy and stability. This model is advantageous for handling high-dimensional data and has strong generalization capability.

4.5.2. Parameter Selection

I optimized the following hyperparameters through grid search to find the best model configuration:

- **Number of Trees (n_estimators)**: Controls the number of trees in the forest, with candidate values of 50, 100, and 200.
- **Maximum Depth (max_depth)**: Limits the maximum depth of each tree, with candidate values of None, 10, 20, and 30.
- **Minimum Samples for Split (min_samples_split)**: Sets the minimum number of samples required to split a node, with candidate values of 2, 5, and 10.
- **Minimum Samples per Leaf (min_samples_leaf)**: Controls the minimum number of samples at each leaf node, with candidate values of 1, 2, and 4.

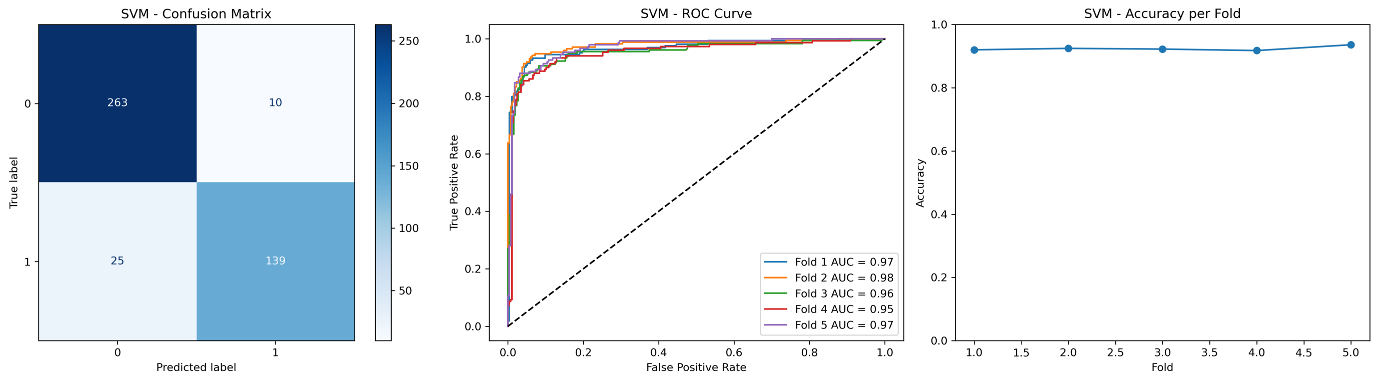

Figure 5. Results of SVM analysis

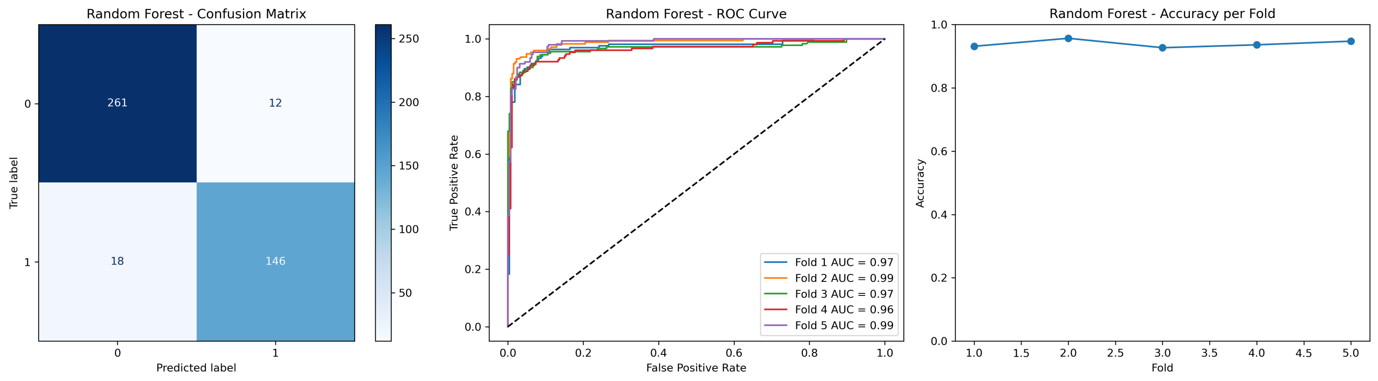

Figure 6. Results of Random Forest analysis

The optimal hyperparameters were found to be:

- **n_estimators**: 100
- **max_depth**: 10
- **min_samples_split**: 2
- **min_samples_leaf**: 1

4.5.3. Training Process

The Random Forest model was trained using 5-fold cross-validation to ensure generalization performance. The specific steps are as follows:

1. **Data Splitting**: The dataset was randomly divided into five subsets. For each run, four subsets were used for training, and one subset for validation, repeated five times.
2. **Model Training**: The Random Forest model was trained on each fold using the optimal hyperparameters.
3. **Performance Evaluation**: For each validation set, accuracy, precision, recall, F1 score, and AUC were calculated. The average across five folds was recorded as the model's final performance.

4.5.4. Results and Visualization

Evaluation Results

- **Accuracy** : 0.9365
- **Precision**: 0.9438
- **Recall**: 0.8989
- **F- Score**: 0.9208
- **AUC Score**: 0.9632

Visualization Analysis: Figure 6. Results of Random Forest

The Random Forest model achieved high accuracy and stability in the spam classification task. Through optimal parameter selection and cross-validation, the model effectively prevents overfitting while maintaining accuracy.

4.6. Gradient Boosting

4.6.1. Model Design

In this project, I used a Gradient Boosting model for classification. Gradient Boosting is an ensemble learning method that iteratively builds a series of weak learners (typically decision trees) and reduces residuals to create a strong classifier. This model was selected due to its strong performance on high-dimensional and imbalanced data.

4.6.2. Parameter Selection

I optimized the following hyperparameters using grid search:

- **Learning Rate (learning_rate)**: Controls each tree's contribution to the overall model, with candidate values of 0.01, 0.1, and 0.2.
- **Number of Trees (n_estimators)**: The number of weak learners, with candidate values of 50, 100, and 200.
- **Maximum Depth (max_depth)**: The maximum depth of each tree, with candidate values of 3, 5, and 7.

The optimal parameters were:

- **learning_rate**: 0.1
- **n_estimators**: 100
- **max_depth**: 3

4.6.3. Training Process

I trained the model using 5-fold cross-validation to ensure generalization. The specific steps are as follows:

1. **Data Splitting**: The dataset was randomly divided into five subsets, with four subsets used for training and one for validation, repeated five times.

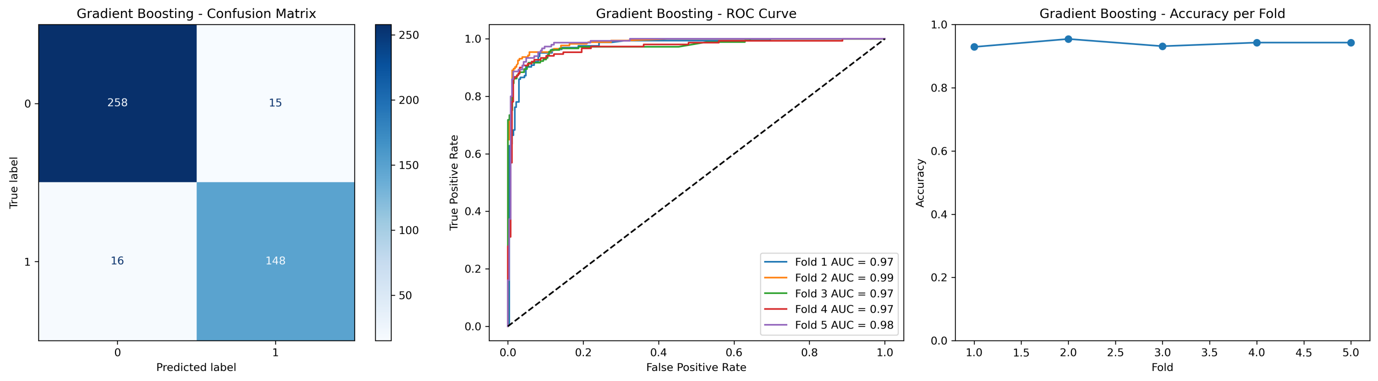

Figure 7. Results of Gradient Boosting analysis

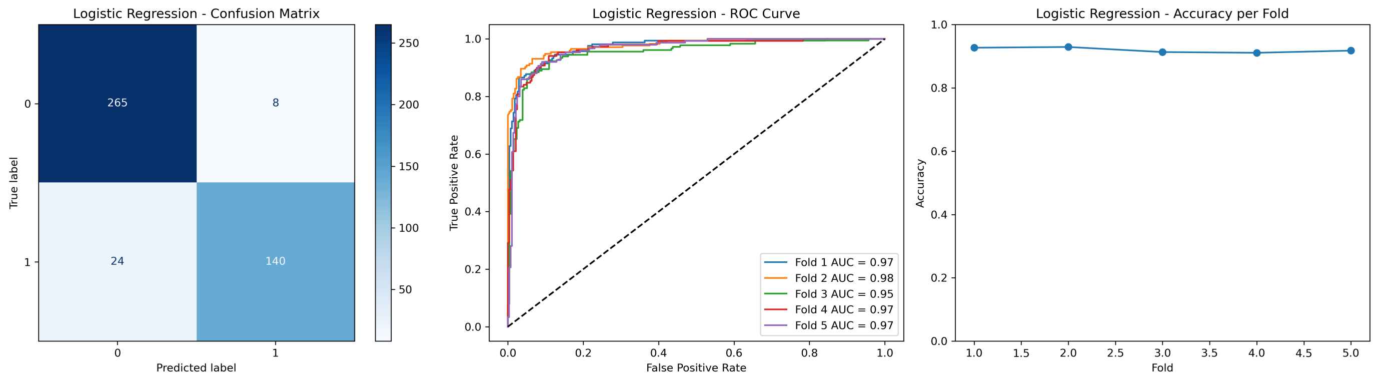

Figure 8. Results of Logistic Regression analysis

2. **Model Training**: The Gradient Boosting model was trained on each fold using the optimal parameters.
3. **Performance Evaluation**: Accuracy, precision, recall, F1 score, and AUC were calculated for each validation set, and the average across five folds was recorded as the final performance.

4.6.4. Results and Visualization

Evaluation Results

- **Accuracy**: 0.9400
- **Precision**: 0.9328
- **Recall**: 0.9051
- **F1 Score**: 0.9186
- **AUC**: 0.9783

Visualization Analysis: Figure 7. Results of Gradient Boosting

In summary, the Gradient Boosting model exhibited excellent classification ability and strong generalization in the spam classification task, with its performance further enhanced through optimal parameter selection.

4.7. Logistic Regression

4.7.1. Model Design

In this project, I used a Logistic Regression model for classification. Logistic Regression is a linear model widely applied in binary classification tasks, learning the relationship between sample features and classes to predict the category of new samples. The model's strengths lie in its interpretability and computational efficiency.

4.7.2. Parameter Selection

I optimized the following hyperparameters through grid search:

- **Regularization Strength (C)**: Controls the weight of regularization, with candidate values of 0.01, 0.1, 1, 10, and 100.
- **Penalty Type (penalty)**: Choice between L1 or L2 regularization.
- **Solver (solver)**: "liblinear" was chosen to support both L1 and L2 regularization.

The optimal hyperparameters were:

- **C**: 1
- **penalty**: L2
- **solver**: liblinear

4.7.3. Training Process

I trained the model using 5-fold cross-validation to ensure generalization. The specific steps are as follows:

1. **Data Splitting**: The dataset was randomly divided into five subsets, with four subsets used for training and one subset for validation, repeated five times.
2. **Model Training**: The Logistic Regression model was trained on each fold using the optimal parameters.
3. **Performance Evaluation**: For each validation set, accuracy, precision, recall, F1 score, and AUC were calculated, with the average across five folds recorded as the model's final performance.

4.7.4. Results and Visualization

Evaluation Results

- **Accuracy**: 0.9195
- **Precision**: 0.9116
- **Recall**: 0.8693
- **F1 Score**: 0.8896
- **AUC**: 0.9665

Visualization Analysis: Figure 8. Results of Logistic Regression

In summary, the Logistic Regression model demonstrated efficient and stable performance in the spam classification task. Through optimal parameter selection and cross-validation, the model achieved high classification accuracy.

5. Result Analysis

5.1. Model Performance Summary

The table 1 Summary for Model Performance

5.2. Model Evaluation

5.2.1. Best-Performing Model

The **Gradient Boosting** model demonstrated the best performance on this dataset, with its advantages highlighted in the following aspects:

1. **High Accuracy and AUC**
   Gradient Boosting achieved the highest accuracy and AUC scores among all models evaluated, indicating superior performance in class separation and reduction of misclassifications. This model effectively balances precision and recall, making it a robust choice for classification.
2. **Capturing Nonlinear Relationships**
   Compared to other models, Gradient Boosting is more adept at handling complex nonlinear relationships within the data. By progressively optimizing weak learners, the model refines its decision boundary accuracy, effectively managing intricate feature interactions.

Table 1. Model Evaluation Results

Index	Model	Accuracy	Precision	Recall	F1 Score	AUC
1	KNN	0.9227	0.9255	0.8634	0.8934	0.9108
2	Preceptron	0.8517	0.7455	0.9183	0.8230	0.9462
3	Decision Tree	0.8938	0.8600	0.8537	0.8567	0.8860
4	SVM	0.9240	0.9356	0.8559	0.8939	0.9644
5	Random Forest	0.9400	0.9447	0.8918	0.9173	0.9765
6	Gradient Boosting	0.9400	0.9328	0.9051	0.9186	0.9783
7	Logistic Regression	0.9195	0.9115	0.8693	0.8895	0.9667

Note: The results above show the evaluation metrics for different models used in the classification task.

3. **Fewer Misclassified Samples**
   Gradient Boosting exhibited fewer misclassified samples, especially in distinguishing challenging cases. This ability to accurately identify difficult samples is essential for reducing model errors and enhancing overall predictive performance.

In summary, Gradient Boosting's advantages in capturing complex data structures, balancing evaluation metrics, and improving predictive accuracy make it outperform other models on this dataset.

5.2.2. Error Analysisl

The performance of the **Weighted Perceptron** and **Decision Tree** models was suboptimal. Key factors contributing to their lower performance are analyzed below:

1. Perceptron

- **Feature Overlap:** Misclassifications were predominantly linked to certain features, such as the high frequency of "technology" and "pm," which exhibit significant overlap between classes.
- **Boundary Issues:** PCA visualization shows that misclassified samples are scattered within the correctly classified clusters, indicating blurred boundaries between classes, which impairs the model's ability to correctly identify positive samples.

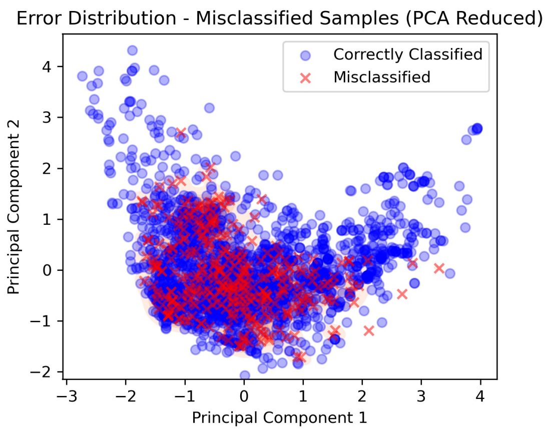

Figure 9. Perceptron Error Distribution

2. Decision Tree

- **Shallow Depth Underfitting:** The number of misclassified samples decreases as tree depth increases, indicating potential underfitting due to a shallow tree structure.
- **Feature Split Limitations:** The model's decision boundary is limited by feature splits, suggesting that increasing tree depth or performing more feature engineering could enhance its discriminative capability.

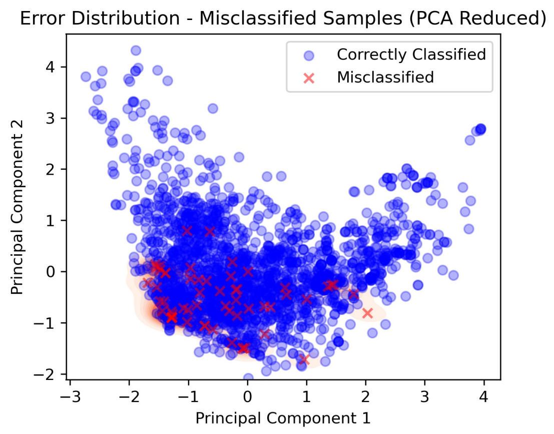

Figure 10. Decision Tree Error Distribution

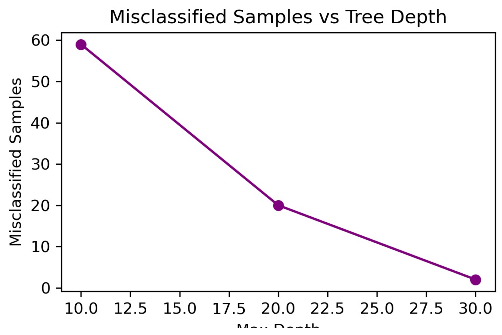

Figure 11. Decision Tree misclassified samples

5.2.3. Recommendations for Improvement

- **Enhanced Feature Engineering:** Explore additional feature transformations, such as polynomial features, and consider regularization to better capture class distinctions.
- **Model Tuning:** Increase the positive class weight for the Weighted Perceptron and consider deeper or ensemble tree-based methods for the Decision Tree.

These adjustments are anticipated to reduce the misclassification rate and improve the overall model performance.

6. Conclusion and Suggestions

6.1. Conclusion

This experiment evaluated the performance of various machine learning models on the Spambase dataset for spam classification. The **Gradient Boosting** model excelled with an accuracy of 0.9400 and an AUC of 0.9783, making it the optimal choice for high-precision spam detection tasks. The **Random Forest** and **SVM** models also performed well, closely matching Gradient Boosting in terms of accuracy and AUC. In contrast, the **Perceptron** struggled with the dataset's nonlinear features, indicating their unsuitability for complex classification tasks. These results suggest that ensemble methods offer significant advantages in handling intricate feature interactions and enhancing classification performance.

6.2. Suggestions

1. Prioritize the Gradient Boosting Model
   Given its high accuracy and robustness, the Gradient Boosting model should be the primary choice for spam detection tasks, especially in applications requiring high precision.
2. Further Hyperparameter Optimization
   For the Gradient Boosting model, consider more refined hyperparameter tuning techniques, such as Bayesian optimization, to further improve accuracy and generalization.
3. Explore Ensemble or Stacked Models
   Combining Gradient Boosting with other models in a stacked ensemble may enhance stability and generalization.
4. Enhance Feature Engineering for Linear Models
   For models requiring simplicity and interpretability, consider feature transformations or adding nonlinear features for KNN and the Custom Perceptron to better capture complex patterns in the data.

These recommendations aim to provide directions for further model optimization to meet the demands of spam detection tasks more effectively.