Assignment 3 – AIML at Scale

Team VAP
Members: Aadyanth Masthipurum. Vishal, Rui Pan

Exploration of Classification Techniques for Spam Detection

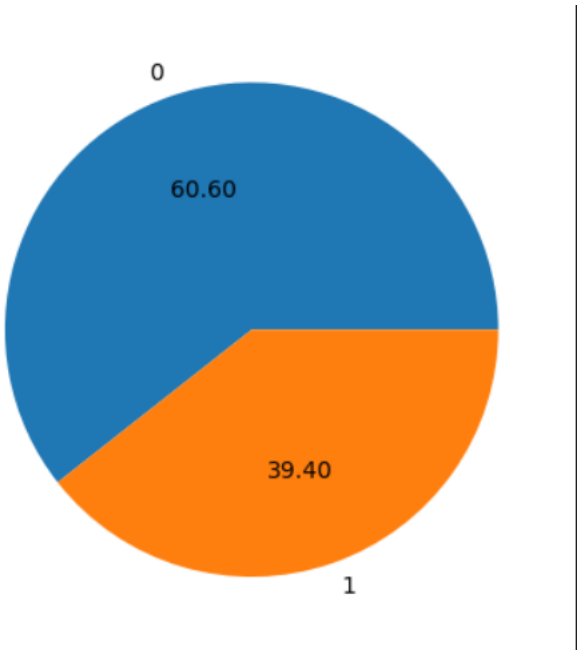
```python
# Load the dataset
df = pd.read_csv(r"C:\Users\aadya\Downloads\spambase\spambase.data", header=None)

# Load column names from the spambase.names file
names_file_path = r"C:\Users\aadya\Downloads\spambase\spambase.names"
with open(names_file_path, 'r') as f:
    lines = f.readlines()
    # Extract column names from lines starting with "word"
    column_names = [line.split(":")[0].strip() for line in lines if line.startswith("word")]

# Manually specify additional column names for numerical features
additional_columns = [f"feature_{i}" for i in range(len(column_names), 58)]

# Combine all column names
all_column_names = column_names + additional_columns

# Assign column names to the DataFrame
df.columns = all_column_names
df.head()
```

✓ 0.0s Python

iternet	word_freq_order	word_freq_mail	...	feature_48	feature_49	feature_50	feature_51	feature_52	feature_53	feature_54	feature_55	feature_56	feature_57
0.00	0.00	0.00	...	0.00	0.000	0.0	0.778	0.000	0.000	3.756	61	278	1
0.07	0.00	0.94	...	0.00	0.132	0.0	0.372	0.180	0.048	5.114	101	1028	1
0.12	0.64	0.25	...	0.01	0.143	0.0	0.276	0.184	0.010	9.821	485	2259	1
0.63	0.31	0.63	...	0.00	0.137	0.0	0.137	0.000	0.000	3.537	40	191	1
0.63	0.31	0.63	...	0.00	0.135	0.0	0.135	0.000	0.000	3.537	40	191	1

The **spambase** data contains **4601** records. The aim was to predict the spam and ham on the behalf of the perimeter given in the data.

Exploring the target variable:



Depiction of the distribution of spam and ham emails in the dataset. Here, 0 depicts ham and 1 depicts

spam emails.

Data Pre-processing and Normalization

StandardScaler for feature scaling, which standardizes features by removing the mean and scaling to unit variance. This method standardizes the features by removing the mean and scaling each feature to unit variance. This choice was motivated by our aim to mitigate the influence of outliers, thereby enhancing the models' ability to generalize from the training data to unseen data effectively. Standard scaling is particularly beneficial in our context, given the diverse range of features in the Spambase dataset, including word frequencies and punctuation marks.

Feature Selection:

```
# SelectKBest feature selection
selector = SelectKBest(score_func=f_classif, k=20)

xtrain_selected = selector.fit_transform(xtrain, ytrain)
xtest_selected = selector.transform(xtest)
```

There are a total of 57 independent features in the dataset. We decided it would be ideal to use select 20 features. We selected the best 20 features to use using ‘SelectKBest’ automated feature selection.

Data Transformation:

```
scaler = StandardScaler()
xtrain = scaler.fit_transform(xtrain)
xtest = scaler.transform(xtest)
```

Hyperparameter Tuning:

```
# Define classifiers and parameter grids
classifiers = {
    "Naive Bayes": GaussianNB(),
    "Random Forest": RandomForestClassifier(),
    "KNN": KNeighborsClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "Gradient Boosting": GradientBoostingClassifier()
}

param_grid = {
    "Naive Bayes": {},
    "Random Forest": {'n_estimators': [10,20, 30, 40, 50, 60, 70, 80, 90, 100], 'max_depth': [None, 5, 10, 15, 20, 25, 30, 30 , 35, 40, 45, 50 ]},
    "KNN": {'n_neighbors': [3, 5, 10, 15, 20, 25]},
    "Decision Tree": {'max_depth': [None, 5, 10, 15, 20 , 25, 30, 35, 40, 45, 50]},
    "Gradient Boosting": {'n_estimators': [50, 100, 200], 'learning_rate': [0.01, 0.1, 1]}
}
```

```
# Perform grid search and evaluation for each classifier
for name, classifier in classifiers.items():
    # Perform grid search
    grid_search = GridSearchCV(classifier, param_grid[name], scoring='accuracy', cv=5)
    grid_search.fit(xtrain_selected, ytrain)
    best_model = grid_search.best_estimator_
```

We estimated the best hyperparameters for each model by using GridSearchCV and by using accuracy as a scoring parameter. Below are the models and the type of hyperparameters estimated.

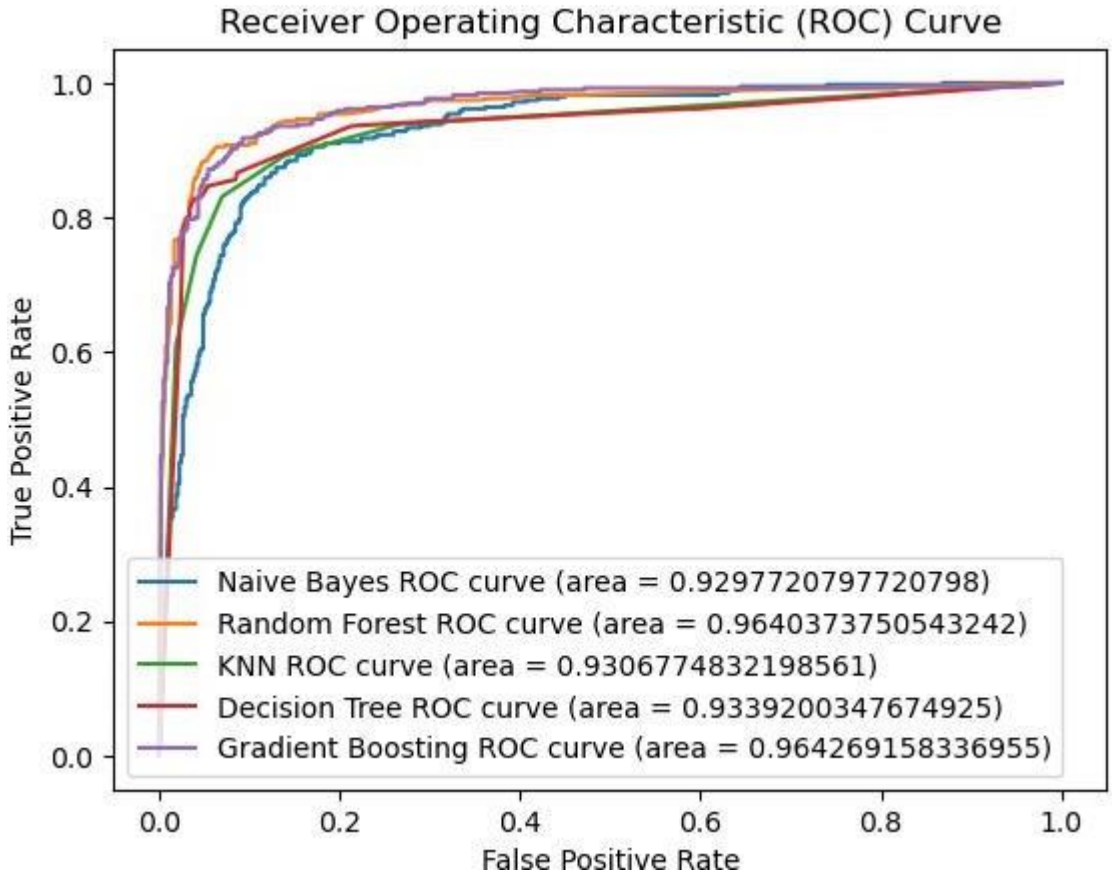
- Random Forest Classifier: Using GridSearchCV we estimated the best combination of the number of trees and the depth of each tree.
- KNN: Estimated the best K for nearest neighbours
- Decision Tree: Identified the best depth of the decision tree.
- Gradient Boost: Identified the best combination of the number of estimators and learning rate.

Performance of model:

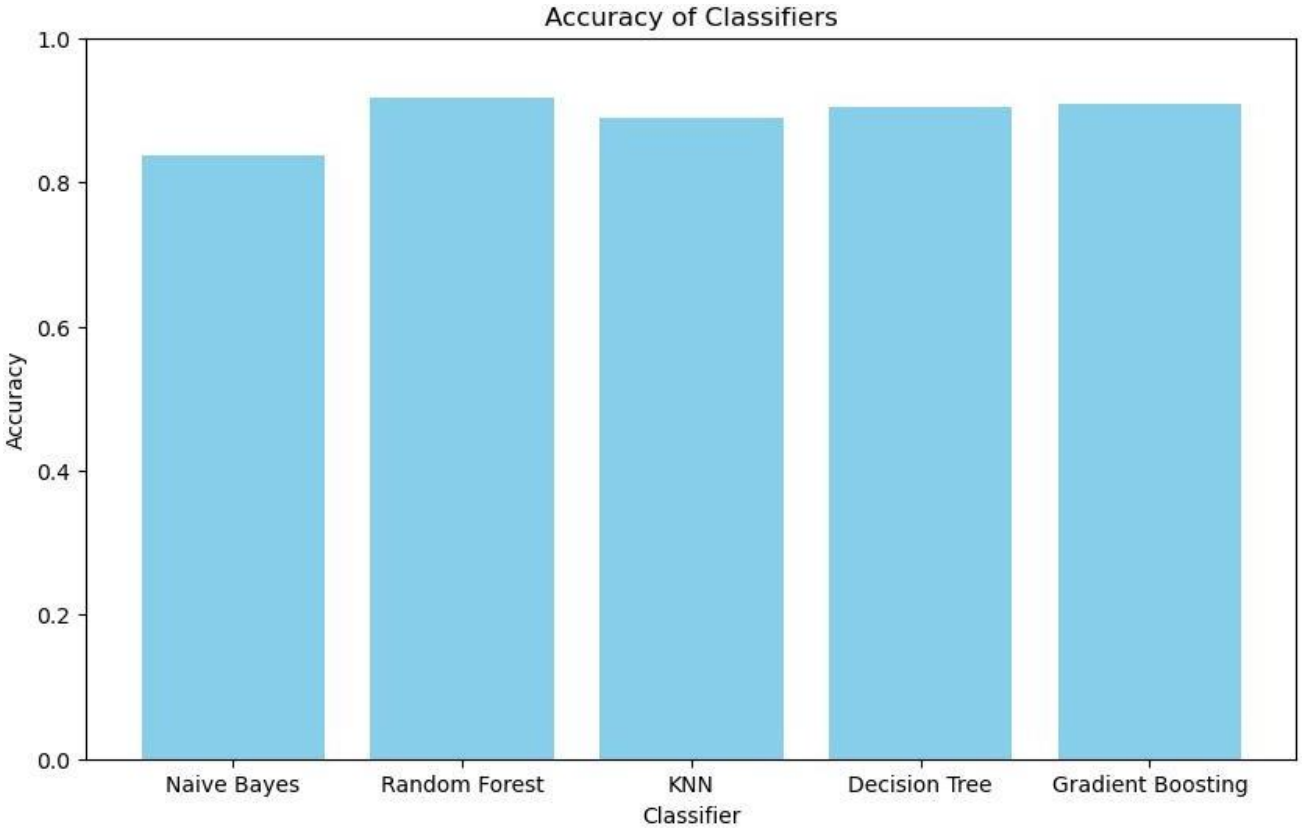
We have built 5 models: Decision Tree, KNN, Random Forest, Naive Bayes and Gradient Boosting algorithms.
Results of models:

```
Naive Bayes Model Evaluation:
Accuracy: 0.8360477741585234
Confusion Matrix:
[[501  30]
 [121 269]]
Precision: 0.8996655518394648, Recall: 0.6897435897435897, F1-score: 0.7808417997097242
AUC: 0.9297720797720798
Random Forest Model Evaluation:
Accuracy: 0.9142236699239956
Confusion Matrix:
[[506  25]
 [ 54 336]]
Precision: 0.9307479224376731, Recall: 0.8615384615384616, F1-score: 0.8948069241011984
AUC: 0.9639987445072191
KNN Model Evaluation:
Accuracy: 0.8838219326818675
Confusion Matrix:
[[502  29]
 [ 78 312]]
Precision: 0.9149560117302052, Recall: 0.8, F1-score: 0.853625170998632
AUC: 0.9398691390216813
Decision Tree Model Evaluation:
Accuracy: 0.9055374592833876
Confusion Matrix:
[[511  20]
...
[[505  26]
 [ 58 332]]
Precision: 0.9273743016759777, Recall: 0.8512820512820513, F1-score: 0.8877005347593584
AUC: 0.9642305277898499
```

Graphical Representation of model scores:



Accuracies Graphs:



The image above explains the performance results of all the models we built. **Random Forest and Gradient Boosting** are the best performing models when the overall predictive accuracy is taken into account.

Random Forest Classifier has resulted in an accuracy of **0.914** and the Gradient Boosting model has yielded an accuracy of **0.9087**

Meta-Modeling Techniques
Our exploration of classification techniques included a focus on meta-modeling techniques, particularly ensemble methods like Gradient Boosting and Random Forest classifiers. These methods combine the predictions of several base estimators to improve generalizability and robustness over a single estimator. The rationale behind employing these techniques was twofold:

Gradient Boosting: This technique builds models sequentially, with each new model correcting errors made by previously trained models. It proved especially effective in handling the complex patterns in the Spambase dataset, enhancing both accuracy and reducing misclassification costs.

Random Forest: By integrating multiple decision trees to make more accurate predictions and estimate the misclassification costs, the Random Forest classifier offered a balanced approach to spam detection. Its performance was notably superior in terms of both predictive accuracy and managing the cost implications of misclassifications.

These meta-modeling techniques underscore our commitment to leveraging advanced methodologies to address the challenges of spam detection, optimizing for both accuracy and the practical implications of misclassifications in real-world settings.

Computing Misclassification Costs:

```python
#calcualting costs

def calculate_normalized_misclassification_cost(conf_matrix, y_test):
    FP = conf_matrix[0][1]
    FN = conf_matrix[1][0]
    TN = conf_matrix[0][0]
    TP = conf_matrix[1][1]

    # Costs
    C_FN = 10  # Cost for false negatives
    C_FP = 1   # Cost for false positives

    # Actual number of positives and potential FPs (all negatives)
    N_FN_actual = FN + TP
    N_FP_potential = FP + TN

    # Misclassification cost
    misclassification_cost = (C_FN * FN) + (C_FP * FP)

    # Maximum possible cost
    max_possible_cost = (C_FN * N_FN_actual) + (C_FP * N_FP_potential)

    # Normalized cost
    normalized_cost = misclassification_cost / max_possible_cost
    return normalized_cost
```

In terms of predictive accuracy, Random Forest classifier performs the best. We then calculate the misclassification costs associated with the model in the ratio of 1:10.

```
print(f"Random Forest Normalized Misclassification Cost: {rf_normalized_cost}")
✓ 0.0s
Random Forest Normalized Misclassification Cost: 0.17264725795531483
```

We find that the misclassification cost is 0.172.

Cost-sensitive Classification Model

Given the nature of our task—spam detection—the misclassification errors carry different weights. Classifying a non-spam (ham) email as spam is far costlier than missing a spam email. This discrepancy is because the former can lead to the loss of important information if legitimate emails are incorrectly filtered out. Therefore, we adopted a 10:1 cost ratio, emphasizing the higher cost of falsely classifying non-spam as spam.

To address this, we adjusted our model evaluation to focus on minimizing this costlier error. Specifically, for our cost-sensitive classification model, we prioritized reducing false positives (non-spam classified as spam) over reducing false negatives (spam classified as non-spam). This approach influenced both our model selection and hyperparameter tuning, guiding us towards models that offer a better balance, particularly the Random Forest classifier, which demonstrated an exceptional ability to distinguish between spam and non-spam emails while minimizing the costlier misclassification errors.

*Some assistance taken from Chatgpt