Project Report

# GitHub URL

<https://github.com/Crowmium1/UCDPA-L.J._Fitzgerald.git>

# Introduction

This paper presents a comprehensive evaluation of various machine learning models in their ability to classify emails as 'spam' or 'not spam'. The study evaluates multiple models and aims to determine which model provides the most reliable and accurate email classification based on a series of metrics, such as accuracy, precision, recall and F1-scores, among others.

Email classification is a pivotal task to filter out unwanted messages and prioritize genuine communication. With the continuous growth of electronic mails, automated systems need to be adept at correctly classifying 'spam' and 'not spam' emails.

The machine learning models being evaluated are as follows:

* **K-Nearest Neighbors (KNN)**: A non-parametric method that classifies based on the majority label of its nearest data points.
* **Support Vector Machine (SVM)**: A linear classifier that determines an optimal hyperplane to segregate classes.
* **Random Forest**: An ensemble of decision trees that considers random subsets of features.
* **Gradient Boosting (GB)**: An ensemble method that builds trees sequentially to correct the errors of its predecessor.
* **Bernoulli Naive Bayes (BNB)**: A probabilistic classifier tailored for binary/boolean features.
* **Gaussian Naive Bayes (GNB)**: A probabilistic classifier that assumes features follow a normal distribution.
* **Ada Boost (AB)**: An ensemble method that adjusts weights iteratively.
* **Stacked Classifier**: An ensemble approach combining multiple models.

# Dataset

For our study, we sourced a dataset containing labelled emails. Non-spam email origins are from a person’s work and personal sources. There are 58 feature labelled columns with 4601 rows. 1 column is the ‘spam/ham’ target variable, where 1 is spam and 0 is not spam. The other 57 columns consist of word and character count frequency information. This information is taken from the ‘Spambase’ dataset, found on the UCI repository (see references). There are missing values in this dataset.

# Implementation

**Data Import and Preparation**:

1. Modules imported and data read from the UCI repository into numpy arrays.
2. Conducted descriptive statistics and visualized the spam/ham class distribution.
3. Data structures, duplicates and missing values were managed.
4. Explored distribution of duplicated rows relative to the target variable.
5. Visualized data correlation through heatmaps and identified high-correlation features.

**Feature Engineering**:

1. Employed principal component analysis (PCA) on the dataset for dimensionality reduction.
2. Developed a dataframe for correlated feature labels. Utilized Random Forest and Gradient Boosting classifiers to derive feature importance values, followed by conversions to dataframe format.
3. Visualized performance metrics of the model against the number of features used, along with ROC curves and reduced dimensionality accordingly.
4. Conducted descriptive statistics on each feature engineered dataframe.

**Model Evaluation and Hyper tuning**:

1. Evaluated base model performance using default parameters, ensuring SVM and KNN models were scaled appropriately.
2. Split the data into training and test subsets, applying cross-validation.
3. Implemented loops to train, fit, and predict data, documenting accuracy scores.
4. Generated ROC curves on base data and calculated confidence intervals for accuracy.
5. Iterated the process with hyper tuned parameters and stored the best models for ensemble training and stacking.
6. Evaluated performance of KNN base and hyper tuned models using hypothesis testing.

**Ensemble Methods**:

1. Re-split the data and evaluated models on validation data, storing accuracies for hypothesis testing.
2. Printed confusion matrices and classification reports for each model trained on optimal parameters.
3. Visualized model performances and reported validation set scores.
4. Conducted bootstrap confidence intervals on data and formed three null hypotheses concerning model performances.
5. Undertook paired T-tests to determine significant performance differences between models.

**Stacking**:

1. Formed a null hypothesis regarding the stacked model's performance relative to individual ensemble models.
2. Used ensemble models as estimators and instantiated a stacking classifier.
3. Trained and evaluated data on the validation set.
4. Performed a paired T-test comparing the stacked model with the best-performing ensemble method.

**Analysis Plots**:

1. Diagnosed model performance, identifying underfitting, overfitting, or potential data needs with learning curve plots.
2. Employed calibration curves to inspect prediction probability alignment.

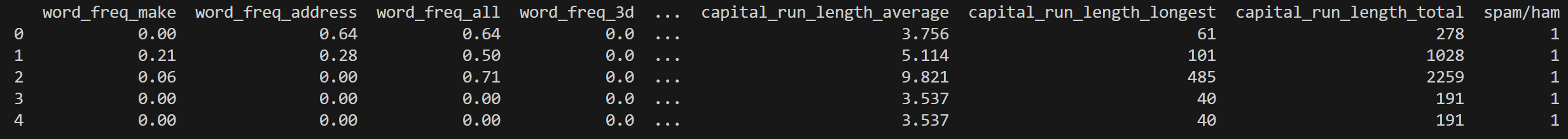
**Image Extraction and File Handling**:

1. Archived the trained model for future access.
2. Transformed the script into a HTML format for improved accessibility.
3. Extracted embedded script images for independent storage.

# Results

**A screenshot of a computer program

Description automatically generated****Fig.1: Spambase descriptive statistics**

**  
Fig.2: Spambase DataframeA graph of ham and spam emails

Description automatically generated  
Fig. 3: Distribution of target class variable in original dataframe.**

**A white sheet with black text

Description automatically generated  
Fig. 4: Distribution of duplicate values vs. target class**

**A blue and red graph

Description automatically generated  
Fig. 5: Correlated feature labels**

**A screenshot of a graph

Description automatically generatedFig.6: Feature Importances  
A graph of a number of different components

Description automatically generated  
Fig. 7: PCA on original dataset  
A graph of a graph

Description automatically generated with medium confidenceA graph of a graph

Description automatically generated with medium confidenceA graph of a graph of a graph

Description automatically generated with medium confidenceA graph of a graph

Description automatically generated with medium confidence  
A group of graphs showing different colors

Description automatically generated  
Fig. 8: Performance vs. number of features and ROC curves for feature variables.  
  
A close up of text

Description automatically generated  
Fig. 9: Feature Variables  
A screenshot of a computer

Description automatically generated  
Fig.10: Descriptive Statistics for feature variables.  
A document with numbers and text

Description automatically generated with medium confidence  
Fig. 11: Inferential Statistics for Base Models**

**A screenshot of a document

Description automatically generated  
Fig. 12: Inferential Statistics for hyper tuned ModelsA graph of a receiver operating characteristic curve

Description automatically generated  
A graph of a receiver operating characteristic curve

Description automatically generated  
Fig. 13: ROC curves for base vs. hyper tuned models  
A screenshot of a graph

Description automatically generatedA screenshot of a graph

Description automatically generated  
Fig. 14: Confusion matrices**

**A screenshot of a computer program

Description automatically generated  
Fig. 15: Hypothesis 1  
A graph with a number of squares

Description automatically generated with medium confidence  
Fig. 16: Ensemble model performance**

**A graph of a curve

Description automatically generated  
Fig. 17: ROC curves for best classifier vs. stacking classifier.  
A graph of a graph

Description automatically generated with medium confidence  
Fig.18: Ensemble models learning curves   
A graph of a graph

Description automatically generated with medium confidence  
Fig. 19: Ensemble models’ calibration curves**

**A screenshot of a computer program

Description automatically generated  
Fig.20: Hypothesis 2**

**A graph of a training curve

Description automatically generated with medium confidenceA graph of a graph of a graph

Description automatically generated with medium confidence  
Fig. 21: Stacking vs. Gradient Boosting Curves**

**A screenshot of a computer

Description automatically generated  
Fig.22: Hypothesis 3**

# Analysis and Discussion

**The descriptive statistics:** The original Dataframe gave an indication for the size, shape and frequency of the data, and the distribution of the target class from figures 1-3. Duplicates' distribution seemed to match the true distribution based on figure 4; however, duplicates were not used as a feature variable. For dimensionality reduction, a correlation matrix and principal component analysis were applied to the Spambase dataset. Highly correlated features with the target variable were deemed valuable. From figure 6, 49 of the original dataset's feature labels comprised 95% of the explained variance.

The random forest showed the best performance on ROC AUC for the validation set, with 30 features selected for further analysis. Comparisons between random forest and gradient boosting importance values revealed similarities, with random forest being more stable. Four base models were evaluated, and the SVM model outperformed in terms of accuracy, precision, recall, and F1-score.

**The inferential statistics:** The 4 base models being evaluated for performance are shown on figure 10. The confidence intervals indicate results falling in that range 95% of the time.

**False Positive (FP)** or type 1 error or sensitivity: A genuine (not spam) email is classified as spam. This means an important email might get missed or deleted.

**False Negative (FN)** or type 2 error or specificity: A spam email is classified as genuine. This means the user gets unwanted emails in their inbox.

The cost of a FP might be higher if a critical email is missed, especially in a business context, while the cost of an FN might be lower since it would mean a minor inconvenience of deleting or marking the email as spam manually.

Therefore, in the terms of balancing sensitivity and specificity, we can aim for a higher specificity (true negative rate). The cost of missing a genuine email could be higher than the inconvenience of manually sorting out a spam email that lands in the inbox.

Notably, both optimized KNN and SVM models have slightly improved in correctly identifying 'spam' emails (higher TPR) and not misclassifying 'not spam' emails as 'spam' (lower FPR).

After hyper tuning, SVM and KNN showed slight accuracy improvements. The obtained p-value comparing hyper tuned KNN to the base KNN was 0.6124, failing to show a significant improvement. In this case, we **fail to reject the null hypothesis** and say that there is no statistically significant improvement in performance from base model to hyper tuned model. The t-statistic of -0.5069 and effect size of 0.0119 corroborate this.

In the context of email classification, high specificity (true negative rate) is crucial to avoid mistakenly labelling genuine emails as spam. Hyper-Tuned SVM exhibited high specificity without compromising sensitivity. GNB and BNB saw no changes after hyper tuning, due to limited tuning of hyperparameters. All models were likely quite well-tuned already, as the baseline performance was already very high.

**A group of numbers on a white background

Description automatically generated Fig.23: TPR,FPR,TNR,FNR**

**All ensemble models** had an accuracy of about 93%. Confusion matrices revealed more false negatives than false positives for all models, with Ada boost underperforming in precision. The f1-scores indicated a balanced performance between precision and recall for all models. Paired t-tests comparing model pairs showed no significant performance differences.

Observing the confusion matrices, all models have more false negatives than false positives, meaning they are more likely to miss 'spam' emails than to misclassify 'not spam' emails as 'spam'.

The random forest classifier has more false negatives and less false positives than AB and GB, meaning it misses a higher proportion of actual spam. This is less preferable.

All models demonstrate a high degree of confidence, with overlapping intervals, indicating comparable performance. This suggests that there isn’t overfitting, as the data is generalizing well to new data.

The p-values from paired t-tests reveal no significant difference in performance between any of the model pairs (AB vs RF, AB vs GB, or RF vs GB). The high p-values (>0.05) lead to failing to reject the null hypotheses.

The effect sizes, which quantify the magnitude of the difference between models, are near zero, reiterating the lack of any substantial difference in performances.

**For Stacking** an accuracy of 0.9403 was obtained, which is higher than the RF that stands at 0.9349. A second hypothesis test was formed to test the performance difference between the stacked model and the best performing model.

A paired t-test obtained p-value was 0.3534, which is greater than the common significance level of 0.05. The consequence of this result was the failure to reject the null hypothesis (H0). This means that there's no statistically significant evidence to suggest differing performance between the stacked model and the RF. The t-statistic of -0.9284 and effect size of 0.0224 backs this decision.

In conclusion, while the stacked classifier shows a slight improvement in raw accuracy compared to the best-performing individual ensemble model (RF), this improvement was not statistically significant. Thus, selecting between these models may depend on considerations beyond accuracy, like model complexity, interpretability, and computational resources.

**Analysis Plots**: Learning curves are plotted for the entire dataset, while calibration and ROC curves target the validation set. These curves diagnose model bias and variance, providing insights into potential overfitting or underfitting.

In all observed instances, the test curves fail to converge with the training curves. This divergence indicates slight overfitting due to model complexity and suggests the model doesn't generalize well to unseen data. Simply adding more data is unlikely to enhance the model's performance. Instead, addressing the model's complexity or introducing new features may offer improvement.

The ROC curve delineates minimal differences between the stacking and gradient boosting classifiers. Calibration curves, essential for gauging actual probability scores, demonstrate that the gradient boosting classifier is optimally calibrated among ensemble models. The more aligned a model's forecast is with the diagonal running from the bottom left to the top right, the better its calibration. Among all ensemble models, the gradient boosting classifier aligns best with this diagonal.

# Conclusions

Hyper tuning for KNN did not yield a statistically significantly improvement.

GB performs the best on average with the least variation. RF, though close to GB in average performance, has more variability, while AB lags in performance but has variability between GB and RF.

Statistical tests show no significant performance differences between the ensemble models.

The stacked model slightly outperforms RF in accuracy, but this difference isn't statistically significant. Therefore, choosing between the stacked model and RF should be based on factors other than just accuracy, such as computational efficiency or interpretability.

All in all, the optimized SVM model performs the best in terms of highest TPR and TNR and lowest FPR and FNR, making it the model of choice, particularly since avoiding misclassification is a priority.

Though the Stacked Classifier showed better accuracy, it wasn't significantly better than the top individual ensemble model, GB.

# Extra File

Another python file was created to show some other techniques I learned in this course. Generators, list comprehensions, regular expressions and web scraping is showcased in this file.

Titles taken from the RTE and BBC news webpages were scraped using the beautiful soup and requests libraries. A function utilizing \*\*kwargs argument was used for this scenario, as after inspecting the html elements, RTE and BBC have different naming conventions for their titles.

The generator is commented out, and a list comprehension is used instead because the memory savings from using a generator in this situation would be minimal. The simplicity of using a list comprehension instead is better, although if you want to scrape thousands of titles, then a generator would be preferable.

# References

UCI Dataset Link - [Spambase - UCI Machine Learning Repository](https://archive.ics.uci.edu/dataset/94/spambase)