Project Report

# GitHub URL

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

# Abstract

This project will focus on the binary classification of a dataset into the two categories ‘spam’ and ‘not spam’, using word and character frequencies. Assessing the effectiveness of 5 different machine learning models on this dataset and running diagnostics throughout will lead to a more robust and reliable methodology from the built-in models for answering this question.

# Introduction

I chose this project because it was a simply binary classification problem to start off as my first machine learning project. The dataset was linked from Kaggle to the UCI repository as the spambase repository.

Having an efficient program for filtering through 1000’s of emails at once to assess whether something is spam or not is a very useful tool for anyone with an email address. One thing I can imagine doing if I continue this project, is deploy it as an application or a web extension. Then with a statistical significance of p-value < 0.05, put filtered emails into junk as spam.

Statistics and visualization will be used throughout to generate insights and ideas for further querying the data. Running diagnostics throughout will lead to a robust and reliable methodology for answering this question. Cleaning, preprocessing, initializing models, parameter estimation, ensembling classifiers, stacking and evaluating model performance and significance will all be done in this project.

# Dataset

The dataset I located on the UCI repository. It contains one text file for the features labels, and a data file, including the emails and their index numbers.

Feature engineering techniques were already done on this dataset to put it into numerical format as word and character frequencies. These may include Bag-of-words, TF-IDF tokenization, lemmatization, removing stop words, n-grams.

This dataset is about classifying emails as spam or non-spam. Non-spam emails are from work and personal sources. There are missing values in this dataset.

* On the last column of the dataset, ‘spam/ham’ is the binary class, where 0 denotes not spam and 1 denotes spam.
* Columns 1 to 48 represent a percentage of characters in the e-mail that match that word.
* Columns 49 to 55 represent a percentage of characters in the e-mail that match that character (.i.e $%&).
* Column 55, ‘capital\_run\_length\_average’, represents the average length of uninterrupted sequences of capital letters.
* Column 56, ‘capital\_run\_length\_average’, represents the length of longest uninterrupted sequence of capital letters.
* Column 57, ‘capital\_run\_length\_total’, represents total number of capital letters in the e-mail.
* Column 58 is the binary class column.

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

# Implementation Process

## Importing

The first thing I did was import any immediate modules I knew I would need. Then I started looking into how to import and read these two files I downloaded UCI repository. Since the files were being read from a different location to the pycharm file, I used an absolute path.

Next, I needed to extract the text I needed from the text file and insert them as the feature labels into the dataframe. I used a for loop while opening the file, split the text from the 33th line onwards, segmented the code I needed. I removed leading and whitespaces. These substrings were assigned to the variable feature\_names. The .data file was loaded, read and turned into a dataframe. The feature\_names variable was assigned as the feature labels to this dataframe.

I printed the dataframe and noticed something was wrong. I printed the list of column names, and they were missing one heading in the dataframe. I manually adjusted the text file and ran an error check on whether the column names and data were lining up. This ran positively.

## Preparing and cleaning data

Firstly, I checked the descriptive statistics for the numpy array. In order to visualize whether there was an imbalance in the binary class, I converted this column into percentages for 0 and 1. I plotted these results using the seaborn library.

I converted the numpy array to a dataframe so that anything being referenced into the future won’t have naming conflicts. I checked for missing values, duplicate columns and rows and removed any. I made a feature variable for duplicate rows in case I needed it.

Since there is no class imbalance on the spam/ham column, no random under-sampling or over-sampling techniques will be needed.

## Feature Engineering

The main concern in this kind of reduction is to make sure that the selected features provide a representative and informative view of the data, especially with respect to the binary classification task at hand.

I made a heatmap of the correlation matrix using the numpy and seaborn libraries and plotted. The np.ones\_like() method is used to create a numpy array mask with the same shape as the dataframe to cover the area that was extracted by np.triu.

I defined a function which applied the correlation matrix to the dataframe using the corr() method and looped over each column, including in the output only values which were above a correlation value of 0.3. Using this function, a dictionary called high\_corr\_counts was made. This dictionary was converted to a dataframe and printed. Correlation values range from 0 to 1.

A function called ‘display\_feature\_importance’ was defined. This accepts ‘importance values’ from a machine learning model, sorts their indices and outputs a bar chart showing them relatively. RandomForest and GradientBoosting importance values are converted into dataframe format and printed. Importance values range from 0 to 1.

PCA analysis was done on all three newly created feature variables. First a dataframe was created combining these variables. These were then sub plotted onto a 2x2 grid using a for loop to scale the data, apply PCA while retaining 95% variance results. Another for loop was used to blank the 4th subplot. Feature labels are removed for PCA analysis.

Descriptive Statistics for each feature engineered variable combined was made by looping the same descriptive statistics that was run on the original dataset.

## ROC

The ROC curves for the original dataset were created. First the dataset was split into test and training data. Machine learning models were instantiated with default values. Using a for loop, the models were fitted to the training data respectively and predicted on the test data. The ROC curve and area were computed and plotted.

## Evaluating base model performances

The models were instantiated again, dropping the multi-nomial learner this time. A function for evaluating a model’s performance was created. This function got the mean and standard deviation cross validation scores for each model using a for loop. A num\_folds variable was created to manage the number of times this operation is iterated for each model for more accurate results. A pipeline was used to combine the standard scaler method with the modelling methods. The append method is used to transfer the output to the return values, which are empty lists.

The evaluate\_models function is ran on scaled and non-scaled data. This information is plotted on box plots using another function.

## Hyperparameter Tuning

The classifiers, kfold variable and best\_models dictionary are instantiated. A for loop is used to do a grid search for hyperparameter tuning using cross validation. The results are added to the dictionary as means, standard deviations and params values for later use. Results are printed.

## Train and Predict

Here I trained the original dataset using the best\_models dictionary contained the hypertuned parameters. With the plot\_confusion\_matrix\_and\_report function, I plotted the confusion matrix for these models and printed out the respective classification reports.

## Ensembling

An ensemble of the classifiers were evaluate with non-scaled data and these were plotted.

## Stacking

Estimators were created using the best\_models dictionary for the stacking classifier. The stacking classifier was instantiated. The training data was fitted to it. The test data was predicted on the stacking classifier. ROC area and curve were attained for these models.

## Learning and Calibration Plots

Learning curve plot and calibration curve plot functions were defined using if statements, the learning curve module and plotting.

The learning curve plot function takes the estimators, features for training/testing, the target variable, and cross validation as parameters. The function uses learning\_curve from scikit-learn to generate training and test scores for different sizes of the training set. Average training and cross-validation across multiple runs to give a mean and standard deviation for scores and an idea of uncertainty in the scores. These are plotted in the function with the ‘plt’ object as a return value. Whether the model is underfitting, overfitting or it would benefit from more data will be realized from these plots.

The calibration curve plot function takes the classifier, and the training/test data as parameters. A calibrated version of the classifier is created which adjusts the probabilities produced by the original classifier, so that they are scaled using the sigmoid method with a cross validation of 5. The function provides a plot of how well the predicted probabilities of a classifier (probs) align with the actual outcomes. The closer a curve is to the perfectly calibrated line, the better the calibration of the classifier.

The calibration curve loops over the classifiers to generate the plot for the calibration curves.

## Importing to file and using regex to extract images to file

The trained stack\_clf model is saved to memory as a .plk file in the project repository folder.

Using the os, re and base64, the script was read to a html file and saved in the project repository folder. Using regex and the .findall() method all the images with 64 bit encoding were extracted and put into the extracted images folder in the project repository folder for further use.

# Analysis and Results

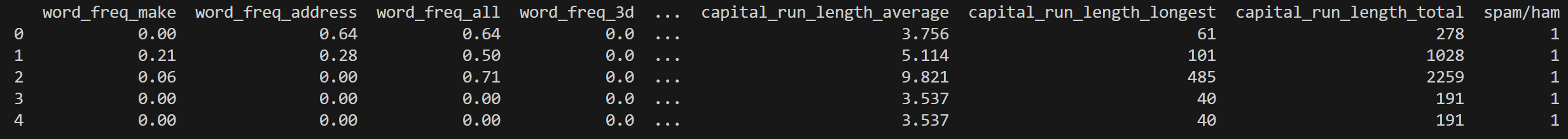
There were no duplicate columns, but there were 391 duplicate rows. This may indicate that these emails are promotional, spam or simply similar in nature. These are non-unique feature combinations, were made into a variable for duplicate rows.

I plotted a histogram of the spam/ham column with percentages displayed. It is evident that there is no class imbalance in the target variable ‘spam/ham’. This is good, as failing to address a class imbalance here can lead to misleadingly high accuracy for the machine learning models. For this reason, techniques such as resampling and assigning weight losses won’t be necessary for this project.

A graph of ham and spam emails

Description automatically generated  
The percentage difference in the binary class is shown in fig. 1.

The first five rows for the original dataset dataframe in fig. 2 for the most part shows the relative percentage a word appears in a particular email or row. The descriptive statistics for this dataframe in fig. 3, shows that there are 4601 emails in the entire dataset.

  
Fig. 2: First 5 rows for the original dataset dataframe.

A screenshot of a computer program

Description automatically generated

Fig. 3: Descriptive statistics for the original dataset dataframe.

A correlation matrix is a table showing correlation coefficients between variables, where the ability is given to see which pairs of data are closely related to each other. Each cell in the table shows the correlation between two variables. It is used as input data into more advanced diagnostic analyses. The heatmap displayed in fig. 4, gives this visual representation clearly.

A blue and red graph

Description automatically generated

Fig. 4: Correlation Matrix heatmap for the all the feature labels.

To scope out the importance of dimension reduction on a dataset, feature variables were created. The high correlation count dataframe represents values for the number of feature labels which are correlated with other feature labels with a set threshold of 0.3. This means that word\_freq\_857 here is correlated with 11 other columns in the original dataset. The top 40 samples were selected.

The Random Forest feature variable was created, and the top 40 samples were selected from the result. It constructs multiple decision trees and creates residuals from splitting the data. Averaging the decrease in these residuals across folds for each feature gives feature importance. Character frequencies ‘!’ and ‘$’ have very high importance values, meaning they predominately belong to the spam class.

The Gradient Boosting classifier was made into a variable and the top 40 samples with the highest importance values were chosen for this feature variable. Features that are frequently used at the primary splits or that best improve the purity of the target variable carry more importance. Just like for a Random Forest, Gradient boosting considers more importance as being highly associated with the outcome.

A computer screen shot of a computer code

Description automatically generated

Fig. 5: High Correlation Count Dataframe

A screenshot of a computer program

Description automatically generated

The dataframes for feature variables 2 and 3 are displayed in fig. 6.

A graph of a number of blue squares

Description automatically generated with medium confidenceA screen shot of a graph

Description automatically generated

The relative importance values for these feature variables across all feature labels are displayed in fig. 7.

A screenshot of a computer

Description automatically generated

Fig. 3: Descriptive statistics for the three feature variables

The PCA results are shown in figures 1 and 2.

retaining 95% variance results. A graph with a line and a line

Description automatically generated with medium confidence

A screenshot of a graph

Description automatically generated

The PCA results for these 3 feature variables are shown in fig. 8.

A graph of a function

Description automatically generated with medium confidence

The ROC curve for the 5 machine learning models being investigated with the original dataset without hypertuned parameters is shown in fig. 9.

A computer screen shot of a black screen

Description automatically generated

The effect of scaled and non-scaled data and the respective boxplots are shown for the original dataset in fig. 10.

A screen shot of a computer

Description automatically generated

The results for hypertuning each model being investigated are shown in fig. 11. These are saved into the best\_models dictionary for further use.

A screenshot of a computer screen

Description automatically generated

The results of the classification reports for each model are shown in fig.12.

A screenshot of a graph

Description automatically generated

The Confusion Matrixes for each model with hypertuned parameters in fig.13.

A diagram of a graph

Description automatically generatedA diagram of a graph

Description automatically generated

Boxplot results of the mean and standard deviation for the ensembled classifiers (AdaBoostClassifier, RandomForestClassifier and GradientBoostingClassifier) are shown in fig. 13.

A graph of a receiver operating characteristic comparison

Description automatically generated

The ROC curve for each model with hypertuned parameters is shown in fig. 14.

A group of graphs showing different types of lines

Description automatically generated with medium confidence

Learning plots are plotted for the best models on the original dataset in fig. 15.

A graph with blue lines and white text

Description automatically generatedA graph with a line and a dotted line

Description automatically generatedA graph with blue lines and white text

Description automatically generated

Calibration curves are plotted for the best models on the original dataset in fig. 16.

# Model Training

## Hyperparameter Tuning

Cross validation is used to select the best parameters for machine learning algorithms. A list of the parameters of interest for his project below are explained.

1. K-nearest neighbors

Euclidean distance is used, as the each feature label represent the same type of values.

1. Support Vector Machines

For an SVM, the most crucial hyperparameters are ‘C’, ‘kernel’ and ‘gamma’. ‘C’ tells the SVM optimization how much you want to avoid misclassifying each training example. ‘Kernel’ specifies the type of algorithm. ‘Gamma’ defines how far the influence of each data point and determines the shape and spread of the decision boundary.

1. Gaussian Naïve Bayes

Default values were used for the best models dictionary.

1. Bernouli Naïve Bayes

This model estimates based on the frequencies of values in the test data in comparison to the training data. If there is a zero frequency occurrence, this can cause problems in the model. To counteract this, a smoothing parameter is introduced known as ‘alpha’.

This defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. This can be seen as the inverse of the radius of influence of samples selected by the model as support vectors.

**Interpretability**:

* **Logistic Regression**: Being a linear model, it offers better interpretability. You can clearly see the weights (coefficients) assigned to each base model's output.

**Training Time**:

* **Logistic Regression**: Typically faster to train than Gradient Boosting.

**Complexity & Flexibility**:

* **Logistic Regression**: It's a simpler model and can often provide good results if the base models' outputs are roughly linearly separable. It's also faster to train compared to gradient boosting.

**Meta-model**:

* **StackingClassifier with Logistic Regression**: In the first scenario, you are using a logistic regression as the meta-model in your stacking ensemble. Logistic regression is a linear model for classification. When using logistic regression as a meta-model, you're essentially learning how best to combine the predictions of your base models in a linear manner.
* (The learning curve shows how error changes as the training set size increases. This curve can help diagnose bias and variance issues in the model and the complexity of the model. The calibration curve is used to assess the quality of predicted probabilities by a classification model. A well-calibrated curve is where the actual observed frequencies of the positive class (1), match the prediction.)

# Insights

(Point out at least 5 insights in bullet points)

* Spam emails often contain words with high frequency counts, such as "free," "money," and "urgent."
* Email length and the presence of specific characters might contribute to spam classification.
* The model's precision, recall, and F1-score metrics will indicate its performance.

# Extra File

Another python file was created in order to show some of the other techniques I have learned during this course.

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. This helps with defining their if statements for identification and its cleaner code for future adjustments.

The titles are converted to dataframes respectively and then merged on the index using pandas. It is more efficient to concatenate here, because if the lengths between columns differ, then nan values will appear, since it needs to be merged on the index for this scenario.

Using a for loop, some useful applications of regular expressions were done on the dataframe to clean it and check for particular information.

A list comprehension was used here instead of a generator 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)