Final Report

CM3203 – One Semester Individual Project

Machine Learning Scanner to Detect Phishing Emails that Contain Malware

Author: Uthman Shaikh

Supervisor: Amir Javed

Moderator: Hiroyuki Kido

School of Computer Science and Informatics

Cardiff University

Final Year Project Submitted for the Degree:

BSc Computer Science with Security and Forensics (with year in Industry)

2021

Abstract

Phishing attacks via email are attempted every year and more people are left vulnerable and helpless against them. Emails have become an essential requirement for electronic communication, setting up accounts, or purchasing any goods online. So, ensuring the security and protection of our email is highly important. One of the main technologies used to detect phishing emails are Machine Learning Algorithms. The purpose of this project is to develop a scanner that uses machine learning to detect phishing emails and attachments that contain malware. Seven different machine learning classification models will be compared and researched to evaluate which is the most effective. These models are Linear Support Vectorisation, Logistic Regression, Naive Bayes Bernoulli, Decision Tree, Random Forest, Neural Network, and K-Nearest Neighbor. This project will also include the functionality to detect malicious intent from malware files.

Acknowledgments

I would like to take this opportunity to thank and express my gratitude towards everyone that helped and supported me throughout this project. I would also like to use this opportunity to express my gratitude towards my supervisor Amir Javed who supported me and encouraged me throughout this entire semester and kept me on track when I needed help. I would also like to thank my moderator Hiroyuki Kido who has given up their time to review my work.

Finally, I would like to thank my parents, and all my siblings who motivated me and cared for me whilst studying at Cardiff University. Without everyones help I would have never attained the knowledge or experience I have today.

Contents

List of Tables

List of Figures

Chapter 1: Introduction

Preface

Since the 1970s emails have played a major role in electronic communication in all around the world. During its rapid growth attackers have taken advantage of this vital tool and turned it against organisations and everyday users by sending malicious emails with harmful files or phishing scams. Phishing attacks have been on the rise all around the world and according to research from the organisation Proofpoint [1], 75% of organisation across the globe have experienced some form of phishing attack. Some companies are experiencing an average of 1,185 attacks per month [2]. Nowadays it is common for organisations to provide some form of phishing awareness training to all their employees, but statistics show that only 3% of employees report phishing emails to their management. This explains why 74% of attacks targeting US businesses were successful in 2020 because 97% of uses cannot identify a sophisticated phishing email. A single spear phishing attack costs and average of $1.6 million [3] which is why we need a better way to protect ourselves and organisations. This is a very interesting topic because most people are aware of this form of attack but still so many people get affected every year. Anyone with an email could be subjected to this, organisations are not their only target even some children get affected. Statistics have shown that 71% of sextortion victims were under the age of 18, which is why we should all try to protect ourselves from phishing attacks. Phishing attacks can be delivered through many different mediums, but research has shown that 96% of social engineering attacks were delivered by email [4] which is why I am focusing my project to detect phishing emails.

If we can create a software that can detect phishing email attacks, then it should be distributed to the whole public, so they have a better chance at being protected from theses atrocities. It will not protect against all attacks, but it will help reduce the number of attacks affecting people. The novelty about my approach is that I will be researching effective techniques used within existing solutions and expanding on them within my own design. One of the differences that my solution has over other similar solutions it that I will be utilising multiple sources within each email. One of the sources I will be using is the data within the header which the user does not usually see when receiving an email.

Project Aims and Scope

The main target for this project is to research and display a proof of concept for detecting phishing emails using Machine Learning. The second important objective of this project is to have the capability to detect any malicious files within the attachments of emails. Different machine learning algorithms will be compared and evaluated to conclude which is the most efficient at detecting phishing emails.

This projects scope is to be able to classify the difference between a phishing email and a benign email by evaluating different Machine Learning models. This system will also be able to evaluate the performance between the different machine learning algorithms.

Target Audience

The intended audience and beneficiaries for this project are individuals who studying or researching the field of security applications and are interested in designing and building Machine Learning algorithms for classification. This paper can also be utilised by researchers who are evaluating the performance and efficiency of different Machine Learning algorithms.

Report Structure

The structure of this project will be divided into 8 chapters.

Chapter 1: Introduction to the issue and the target and goals for this project.

Chapter 2: The research and information conducted to support design of this project.

Chapter 3: The design structure and plan for developing this software.

Chapter 4: Details regarding the implementation of the system and the tools and APIs used.

Chapter 5: Evaluation an Analysis of the results produced by the system.

Chapter 6: Aspects of the system that can be improved within future work.

Chapter 7: Summarisation of the conclude result of the developed system.

Chapter 8: Personal reflection on self-development whilst working on this project.

Chapter 2: Background Research

Machine Learning Models

When I took on this project, I had no prior experience with machine learning. My only understanding of this technology was that some form of data is fed into an algorithm and the Machine Learning part creates a model that classifies the data into one or many different objects. The outcome I intended was to learn and understand about the different models that can be used to classify emails into phishing or benign.

SVM (Support Vector Machine)

The Support Vector Machine algorithm is a very strong but simple model to understand which can help solve linear and non-linear problems. Essentially this model takes an input of data and plots it on a graph, it then attempts to classify the data by drawing a line to separate the data into different possible classes. This line is known as the Hyperplane [5]. The hyperplane can be distinguished in a linear or non-linear way, as it depends upon how the data is distributed within its space. The SVM model also applies varying weighting to each of the features based upon their importance. This helps classify the data accurately so two disconnected sections can be formed.

Optimal hyperplane is distinguished by finding the points that are closest to the line from both classes [5]. The points in the 2D space are known as the support vectors and the space between the lines are known as the margin. The optimal line is drawn as the mid-line within the margin so that it is evenly distanced from the different support vector classes. We can define the hyperplane as a n-dimensional Euclidean space where n-1 dimensional subset of the space can divide the space into two disconnected sections [5]. This figure shows a representation of a linear regression.

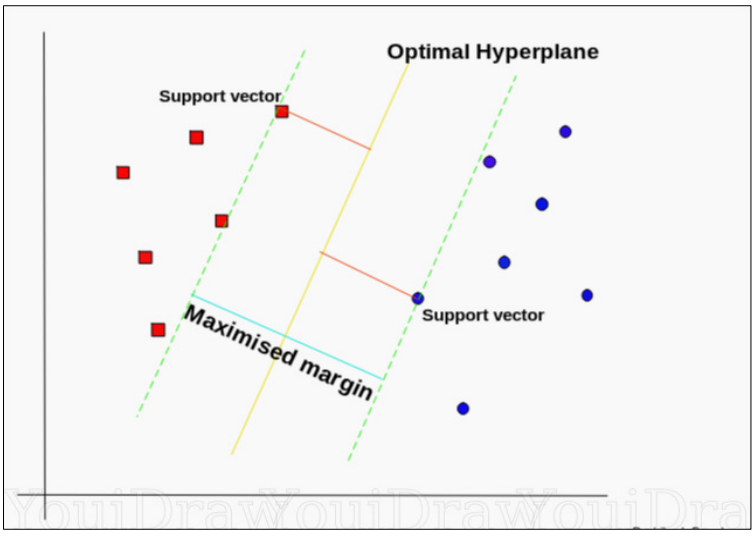


Figure 1 Optimal Hyperplane using SVM algorithm [5]

Logistic Regression

The Logistic Regression is another basic algorithm used to solve classification problems. Its underlying methods and techniques are very similar to the linear regression model but instead of using a line to distinguish the data it uses a more complex function known as Sigmoid which can be represented as an S-shaped curve. The Sigmoid function is used to map predicted values to probabilities by mapping any real value to another value between 0 and 1 [6] as they represent the minimum and maximum values. As this model uses a supervised learning algorithm it requires a pre-labelled outcome variable as a binary value such as 'yes' and 'no' or 1 and 0. The logistic regression also applies weightings to the features in the data, but they do not influence the probability linearly. Their weighted sum is transformed into a probability by the logistic function and that is used to derive a predicting value [7].

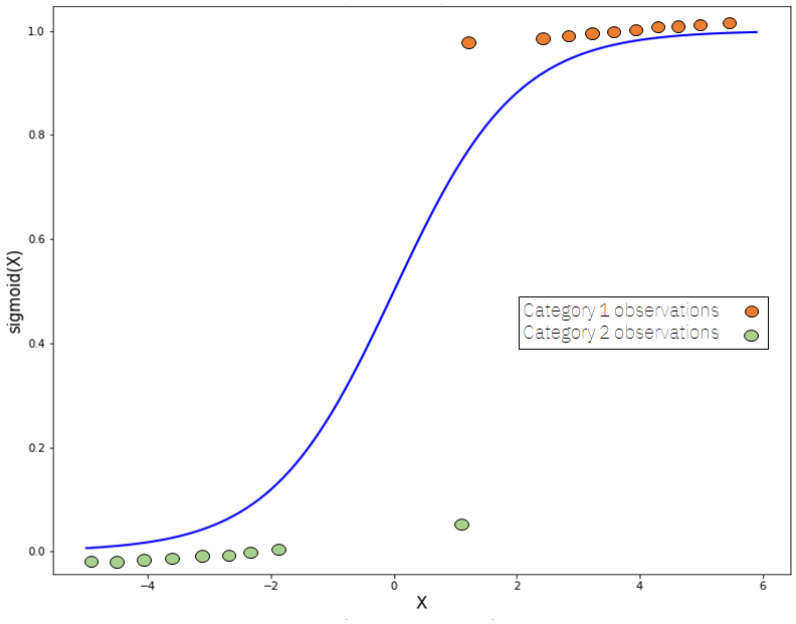


Figure 2 Logistic Regression Example with Sigmoid Curve [8]

Naive Bayes

Naive Bayes is known to be a simple machine learning algorithm which uses a probabilistic classifier. In this theorem there are different classifiers that can be used but as this program will only be classifying the data into binary values Bernoulli Naive Bayes was the most appropriate as it uses Boolean variables as predictors. Bernoulli makes its classification using the Maximum A Posteriori (MAP) estimation rule found in Bayesian statistics [9]. This estimation rule computes the conditional probability of one outcome when given another outcome, we call this the posterior probability or P(Y|X). For each value in Y, we need to compute its expression by calculating the probability that the value Y will take on any given value, when given new attribute values for X from the distributions of P(Y) and P(X|Y) estimated in the training data [10]. In short, the most probable or maximum values for P(X|Y) are taken, hence why it is also known as 'maximizing a posteriori'. The usefulness for this model is that it only requires a small number of training data to creates its necessary parameters for its classification.

Equation 1 Naive Bayes Classifier [11]

$$\text{Posterior} = \frac{\text{Likelihood} * \text{Prior}}{\text{Evidence}}$$

$$P(target|\ predictor) = \frac{P(predictor\ |\ target)\ .\ P(target)}{P(predictor)}$$

Decision Tree

The Decision Tree classifier model is used in Supervised Machine Learning. It uses a sequential structure of questions and answers to narrow down the result to a specific class by creating a tree like structure that uses “if this than that” conditions [12]. The depth of the structure is important as it represents the number of questions that must be answered to produce a predictive result. This model is visually simple to understand and to interpret, it can use both categorical and numerical forms of data for its classification. In the structure each condition is represented as a leaf (node) and the different classifying outcomes are branches (edges) [13]. For this program, each node would represent a feature and each branch would represent one of the two classes ‘phishing’ or ‘benign’. The figure bellow is a representation of what a structure would look like for a decision tree.

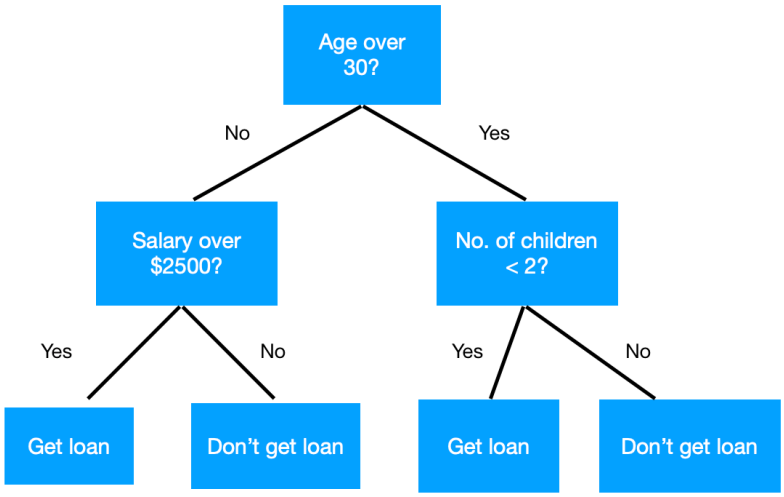
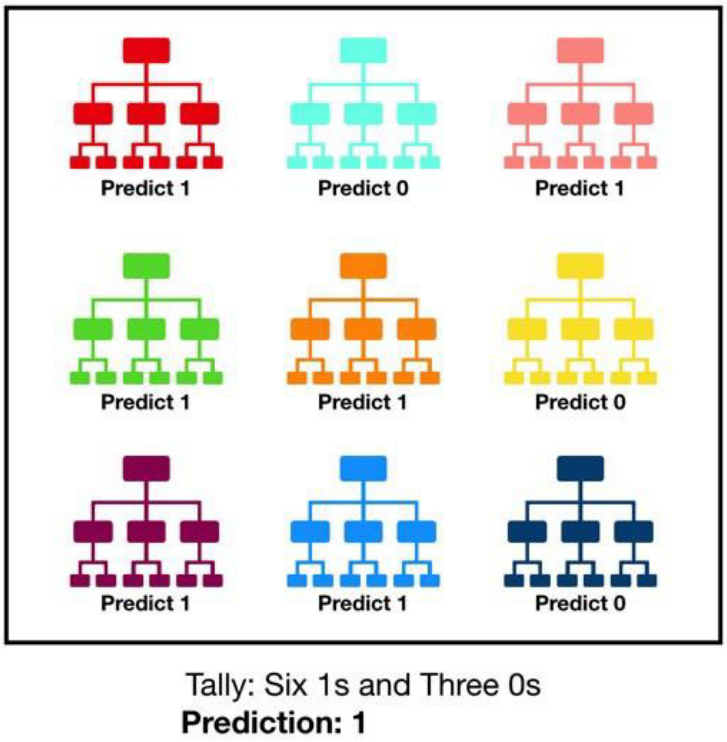


Figure 3 Decision Tree Structure [14]

Random Forest Classifier

The Random Forest Classifier is fundamentally the same as decision trees, but the only difference is that it consists up of multiple decision trees hence the name ‘Random Forest’. This model creates many of uncorrelated decision trees to classify an object and then the most common predicted result from each tree is used as the predicting classifier. So theoretically it is more accurate than the decision tree, but it does takes longer to train and test the model. The wonderful effect from this classifier with using multiple decision trees is that it helps eliminate the individual errors that could appear within each decision tree [15]. This models’ basic principle uses the knowledge of many to decide on a predictive outcome. The figure bellow provides a representation of the random forest classifier.

Figure 4 Random Forest Classifier [15]



Neural Network

Neural Networks use complex mathematics to derive a classifying result. A neural network takes its inspiration from the same learning process that is used in a human brain. The network it creates consists of functions, called parameter which are also known as neurons. The neurons allow the computer to fine itself by learning and analysing new data after receiving multiple inputs [16]. There are many layers in the network and the outputting results from each neuron in the first layers is then used as an input of data to the neurons in the second layer and this process continues until every layer of neurons have been factored and the final neurons have all received their inputs. The end neurons in the network produce the terminal result in the model. The figure bellow provides a representaiton of a Neural Network, the 'x' representing the inputting data and the 'f' as the classifying output.

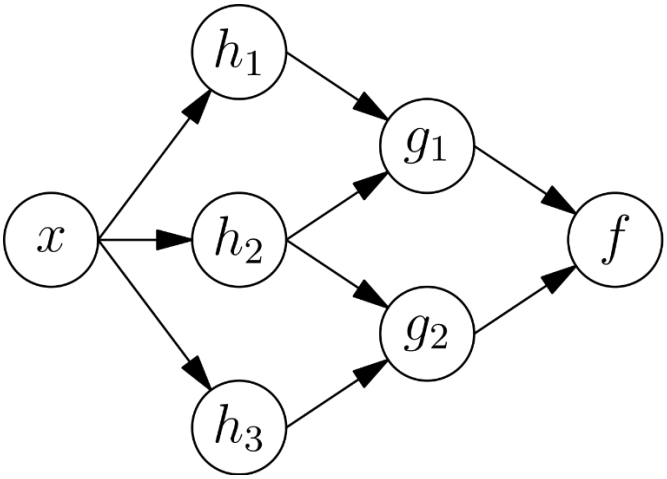


Figure 5 Neural Network [17]

K Nearest Neighbor

K Nearest Neighbor (KNN) is another simple algorithm that is easy to visually represent as the model has been designed to classify its data based upon how its Neighbor is classified. In this model, data is presented on a graph so when classifying a new input of data, the number of nearest Neighbor's are determined based upon the value parameter 'k'. K can be any value, for example if 5 is used then the 5 nearest Neighbor's to the predicting value are grouped into their classes and the largest class is used as the predicting result. This algorithm requires a lot of fine tuning to the parameters to decide on the most effective value to represent parameter K. Smaller values used for K can be quite noisy and will have a high effect on the result whereas a high value for K will have smoother decision boundaries meaning a lower variance, but it will have an increased bias [18].

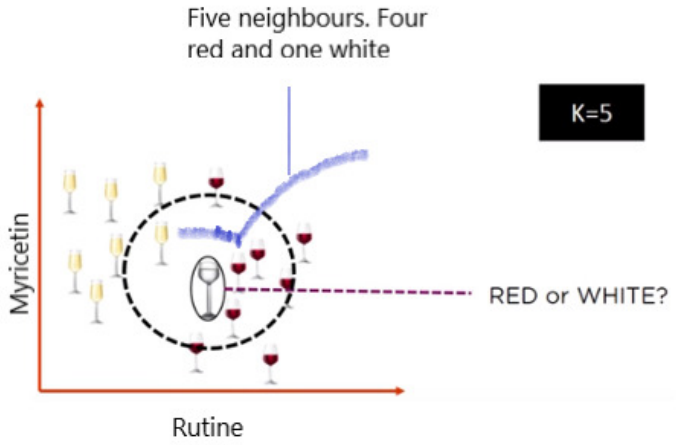


Figure 6 K-Nearest Neighbor Example [18]

Learning Evaluation

To evaluate the performance and results of the scanner many techniques will be used. The majority will be targeted towards evaluating the Machine Learning algorithms.

Confusion Matrix

A Confusion Matrix is a classification model used to assess the performance to see if each algorithm has positively or negatively predicted the correct classes. From the variable classes it produces other performance indicators can be evaluated. The size of the matrix can be measured as 'n x n' where n represents the number of possible classes. For this tool, the machine learning algorithms will only be classifying two possible classes so the expected matrix size will be '2 by 2' which are phishing or benign. Confusion matrix classes are [19]:

- TP represents **True Positive** which is when the model correctly predicts the positive class.
- FP represents **False Positive** which is when the model incorrectly predicts the positive class.
- TN represents **True Negative** which is when the model correctly predicts the negative class.
- FN represents **False Negative** which is when the model incorrectly predicts the negative class.

	Classified as phishing	**Classified as legitimate**
Phishing website	True Positive (TP)	False Negative (FN)
Legitimate website	False Positive (FP)	True Negative (TN)

Figure 7 Confusion Matrix [20]

Performance Indicators

From the resulting data found in the confusion matrix we can generate other evaluation metrics to assess the performance of the machine learning model. These metrics are Accuracy, Precision, Recall and F-Measure [21]. One other performance indicator that is important to gather is the elapsed amount of time it takes to train and test the model.

Accuracy: this metric represents a general accuracy of the model, but it may not be useful if the classes are not equally balanced. The datasets of emails required for this tool need to have balanced amount in benign emails to phishing emails.

Accuracy =

(True-Positive + True-Negative) / (True-Positive + True-Negative + False-Positive + False-Negative))

Precision: this value is calculated to distinguish the exact percentage of positive values that were truly positive. This produces the performance of positive terms for classification.

Precision = (True-Positive / (True-Positive + False- Positive))

Recall: this value has also been known as the "hit rate" which represents the proportion of the True Positive results the model was able to classify or the rate of the genuine Positive predictions.

Recall = (True-Positive / (True-Positive + False-Negative))

F-Measure: this can also be known as F-Score which can be derived from the harmonic mean of the results of precision and recall. The calculation is represented as:

F-Measure = (2 * Precision * Recall) / (Precision + Recall)

Time to Train and Test: when comparing machine learning algorithms, the resulting accuracy is without a doubt important but the time it takes to train and test the model plays a huge factor when finding the most suitable algorithm. If I intend on further developing my model with larger datasets, then it would be preferable to use a model that can train its data quickly preferably under a minute if the dataset has less than 10,000 emails.

Python ML Libraries and APIs

For this section I will be discussing the tools that were needed for the development of this project. Python was my language of choice because most libraries and APIs I intended on using had a lot of support. Regarding my own skills and knowledge, I am most confident and experienced with using this language, so it was a logical decision. Python is a well-known regarded language that has many open-source Machine Learning libraries, like Scikit Learn, Keras and Theano.

Scikit Learn is considered one of the most useful libraries to use in machine learning in Python. As it contains many tools with statistical modelling around classification, it also comes packed with lots of useful features like Supervised and Unsupervised learning algorithms, cross validation, feature extraction and measuring performance. It also has support for different machine learning algorithms which will provide me with useful data to find the most effective model. Keras was another python tool that I considered using as it provides a high-level, front-end specification and implementation for building Machine Learning models [22]. It is especially good with building Neural Networks as it has

lots of support with back end deep learning frameworks. I also considered using Theano as it has a low-level library which is great for scientific computing and deep learning tasks. It uses multi-dimensional arrays to provide many mathematical operations and expressions. For this task, my chosen machine learning tool to use was Scikit learn because of its wide variety in algorithms and features with building classification models.

For detecting Malicious Files, I needed to use an API, the two that I found and compared were VirusTotal API and Scanii. They both offer a wide range in security and detection with an easy to orientate REST API that can be used without any costs. Their free access is limited to a degree, with VirusTotal you are capped at fours detections per minute and with scanii they offer a free trial, but it only lasts for a limited time. For this project, I used VirusTotal for scanning and detecting malicious files as it can be used for an unlimited number of times and because of its wide range of features and detection like scanning hashes.

Datasets

One of the most important parts of this project was having datasets of emails that contained both phishing and benign content. As I intended to extract my own set of features, I had to narrow my search to find datasets of emails in a raw state where they have not been stripped of any information and were available publicly. I was only required to produce an MVP (minimum viable product) that shows this concept strategy is effective with detecting phishing data. One of the difficulties I discovered with finding the right dataset was that many datasets had been stripped of information. Some only contained the subject and email body and many others were already extracted of their features. In the end I decided on looking for two separate datasets one containing only phishing emails and another of only benign emails.

The dataset I chose to use for detecting phishing emails “Fraudulent E-mail Corpus” [23] [24] mainly consisted of Fraudulent Scammers which were used and have affected people financially. This dataset contains around 4000 emails and for each one all the headers, tags and information are presented. For the non-phishing (benign) emails the dataset that I decided to use was the SpamAssassin corpus [25] as it also contained a thorough number of headers and context for all emails. This dataset has also been used in many research papers regarding machine learning in spam and phishing detection [26] [27].

The final required dataset needed for the development of this project was a list of malicious files or the hashes of a malicious files. Either one of these could be used to test the connection with the API

to the file detector (VirusTotal). The dataset used to test the functionality for detecting malicious files was gathered from Kaggle, [28] it contains a list of 50 hashes from different malwares.

Existing Solutions

Machine Learning classifiers for phishing detection have existed for a long time but still many people every year are affected and only 3% of people report the attempted attacks. From researching varying methods and techniques used in existing solutions I can learn effective information that can help me build develop this tool. In 2018 Ethan, Richard, and Joshua, extensively researched efficient methods for detecting malicious files found in email attachments [29] like portable executables (PE). They did this by running mathematical algorithms and performance tests against VirusTotals API to assess how effective it is at detecting malicious files. One conclusion that came to be was that VirusTotal is effective with detecting malicious files, but it can over detect by labelling non-malicious files as malware; however, this was likely caused by suspicious vendors assuming that all attachments send in a phishing attack were malicious.

When detecting phishing emails there are many ways to tackle this issue one commonly used method is the bag of words tool which extracts textual data. The bag of words model uses a large two-dimensional matrix of the textual features used in each email [30] to classify between phishing and benign. The accuracy it produces is ok, however there is a lot of room for improvement but a benefit to this strategy is that it is simple to create and put together. From the report written about "Identifying Phishing Attacks" by Brandan Azad [31] his conclusion provided useful insight for using the bag of words strategy regarding its performance. This strategy can produce an adequate result, but it can take a long time to compute, and detection could be more accurate.

Another methodology used to detect phishing emails is extracting data about different features within email. In 2014 Prateek, Anand, and Ponnurangam from the Institute Indraprastha in Delhi used this technique to detect phishing attacks [32]. Some of the features they would extract from the subject are replies, the number of words in the subject header, the richness of the text (measured as a decimal value), if it is verified and the number of characters within the subject. Those features and more were also extracted from the body of the email, the combination of these features is referred to as Stylometric features. From this report they concluded that the Random Forest classifier is the most effective at detecting phishing emails which achieved a maximum accuracy of 97.04%.

For this project, I aim to explore different machine learning algorithms for detecting phishing attacks to find the most effective classifier. This project will also demonstrate what features are most effective with this detection. The main differentiating factor that makes my project different and unique

compared to other available solutions is that I will be extracting large quantities of features from each email to use as a dataset. This dataset will contain the measurements of emotion in the text, the quantities of punctuation, the sentiment analysis and various other features will be extracted from the headers of the email. From the research I have performed with looking at existing solutions most data and information is disregarded and not used when it comes to classification between phishing and benign emails.

One of the takeaways from this project that I will be researching is what features play the biggest role regarding this classification in emails. Some of the features I chose to extract from the emails were previously used in other projects and some other features were specific things I came up with from analysing the layout of a raw email. It is important to use a wide range of features so we can compare after the models have been trained and tested which features were set to high importance and which had little effect on the classification. For the features that had little effect we can draw a conclusion on whether to keep them or remove them.

Chapter 3: Design and Methodology

This chapter will explain the design and methodology of the implemented solution. The initial requirements will be discussed, and the system plan and design will be presented to show the intended strategy for this project. It is essential that these steps be carefully considered and planned as they will provide a baseline for the needs and desires of the solution.

Software Requirement Specification

The software requirement specification for this project can be broken into two sections functional and non-functional. These requirements help assist with guidance during the technical implementation and provide a criteria evaluation for the success of the final product.

Function Requirements

The important functional requirements are defined in the following statements:

- Raw email files must be processed as an input for feature extraction
    - If the emails are not in their raw form, then there will be missing sections of data such as the data found within the header.
- The features generated from each email must be a numerical value.
    - The machine learning models can only process numerical values as it uses mathematical equations to classify the data.
- Two separate datasets should be created one of Benign emails and another of Phishing emails after the extraction of the features from the data.
    - These two datasets will be combined and labelled if they are phishing or not. The combined dataset is needed for training and testing the machine learning classifier.
- A target of five separate machine learning algorithms should be used and compared to derive the classification results. The selected five models used are algorithms I have researched and learnt about.
    - The machine learning models that should be used are: Support Vector Classification, Logistic Regression, Naïve Bayes, Decision Tree and Random Forest Classifier. Having a wide range or models to use will help determine the most effective algorithm.
- 70% of emails must be classified into their correct class.
    - The main purpose of this tool is to detect phishing emails so if the majority cannot be classified then this tool would be ineffective.
- The tool must be able to identify malware within malicious files or from malicious hash files.

    - Malicious files can be sent as an attachment within an email which is why the need to detect these harmful files is so important.

The optional functional requirements are defined in the following statements:

- A target of two extra machine learning algorithms should be added and compared to the classification process. The selected extra two models used are algorithms I have researched and learnt about.
    - The other machine learning models that should be used are: Neural Network Classifier and K Nearest Neighbor. Comparing more machine learning models will assist with finding the most effective model.
- The software should use a balanced dataset of benign to phishing emails.
    - To determine the accuracy of the model a balanced dataset should be used containing 50% benign emails and 50% phishing emails. Although in real life the testing of a 50-50 split in phishing to benign emails does not happen, I need to have an even comparison so the performance evaluators can be analysed.
- 90% of emails must be classified into their correct class.
    - The main purpose of this tool is to detect phishing emails so if the majority cannot be classified then this tool would be ineffective.
- A summary evaluation of each classifying model should be logged.
    - The Accuracy, Precision, Recall, and F-Measure all need to be calculated so we can evaluate which model is the most efficient.
    - The amount of time to train and test each algorithm needs to be recorded too.

Non-Functional Requirements

The non-functional requirements are defined in the following:

- **Reliability** – the tool needs to be able to compute without any errors and be available all the time. It needs generate a dataset of features, classify phishing emails, and detect malicious files without any errors.

- **Usability** – when running the different capabilities of the system is should be easy to use and to navigate.

- **Speed** – the time it takes to process the datasets, detect the malicious files, and classify the results should be reasonably quick but justifiable if it extends for more time.

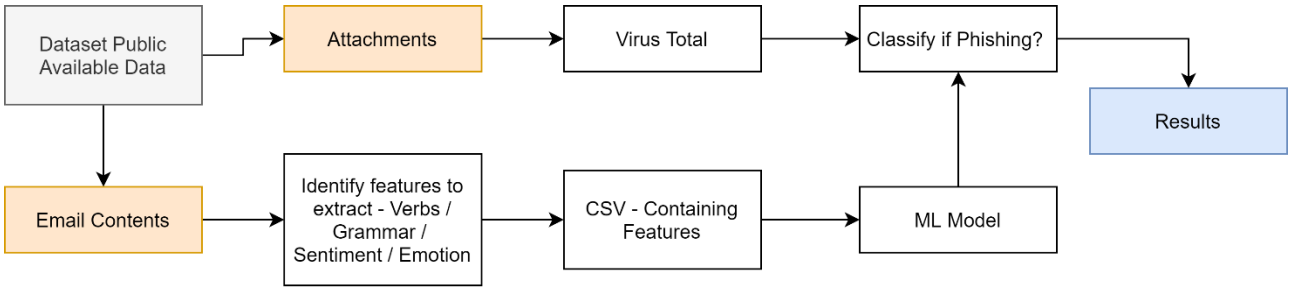
- **Size** – the size of the system will not exceed 1000 megabytes as a dataset of this size would take up more storage than required. This system only needs to demonstrate the functionality of the detection and classification.

- **Re-useability** – the implementation of the system needs to be broken down into separate modules so that they can be further used by other projects or systems.

Initial System Design

The initial plan created for this software had two separate paths that aided with the classification for detecting malicious emails. One path was for detecting malware within malicious files, and the other processed the classification for detecting phishing emails. The strategy I intended for detecting malware was first gathering a list of malicious files or the hashes of malicious files and scanning them against the VirusTotal API. The outcome would then be recorded and would be classified as malicious or not. For classifying emails there are a few more levels, the first stage was gathering datasets of phishing and benign emails, then a list of features would be extracted from each email and stored in a two-dimensional matrix. Phishing emails would be labelled 1 and the benign emails 0 so they could be distinguished for the machine learning models during testing and training. At the machine learning stage, the performance, and results for each classification algorithms are compared and evaluated.

Figure 8 Initial Design

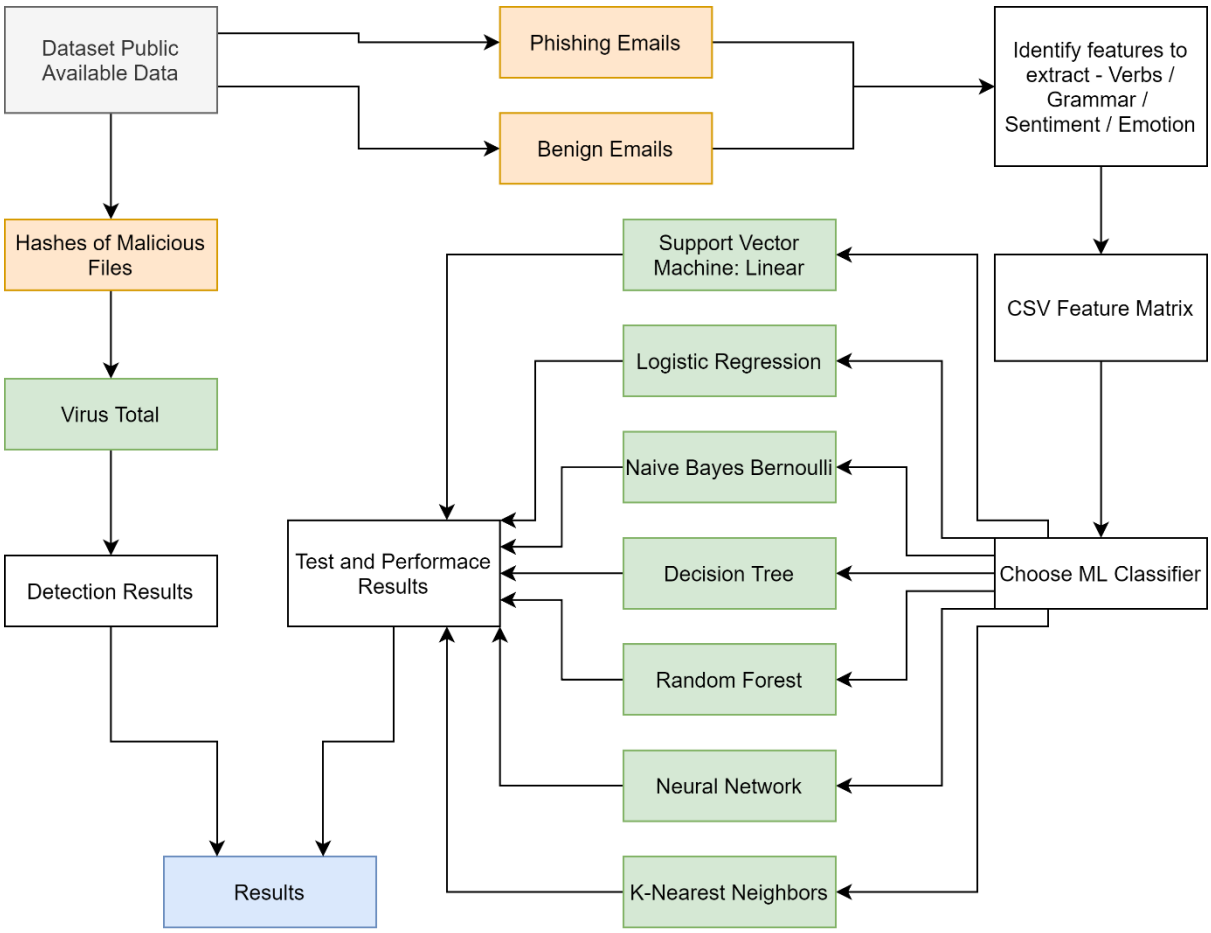


Final System Design

The final design of the system has been changed from the initial design but there are many similarities between them both (see figure 9). Regarding the datasets there are three different types, one is a list of malicious hash files that has been used in phishing attacks and the other two are separate datasets of phishing emails and benign emails. The VirusTotal API is used to detect if the malicious files contain any malware, if the files are correctly identified then the results are sent back to the user. For the email classification path at the first stage all the features are extracted from the datasets and merged to generate a new dataset. This is used to train the machine learning algorithms. At the next stage of

the design the user has the option to choose a specific model they want to test. Upon selecting a model, the system starts the training then the testing results and performance are recorded and presented to the user.

Figure 9 Final Design



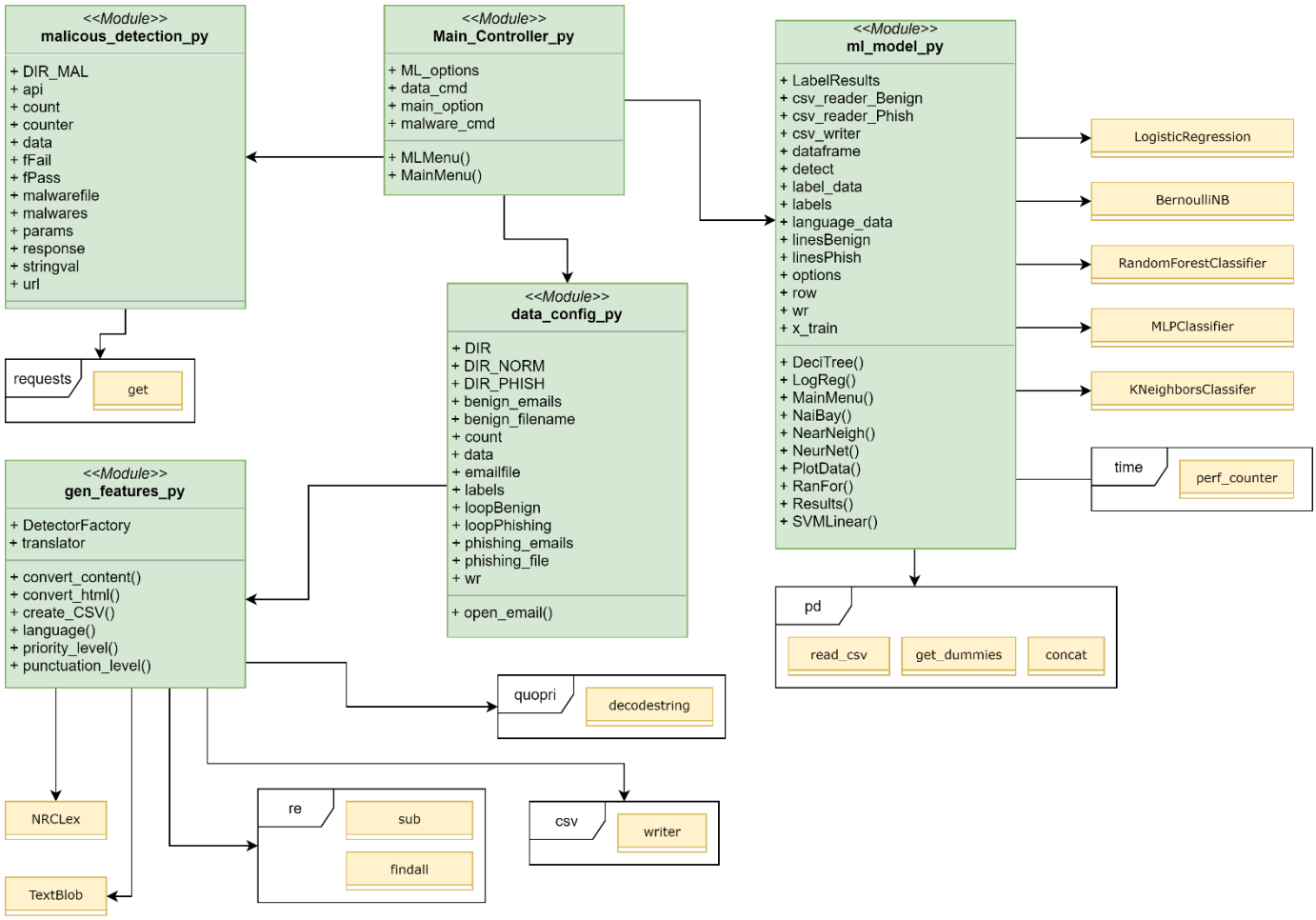
The UML figure below shows a representation of the five separate python scripts used in this system and how they flow together. There are different labels flowing from each script which represent some of the libraries and APIs used. Starting with the first script Main_Controller.py the purpose of this page is to provide a simple interface that can be used to execute the different functionalities of this tool. This is the only script that need to be executed to operate this tool, when doing so it creates a command line interface that give you simple menu option to select from.

The script malicious_dectection.py handles the detecting of malicious files, there is only one API used by this script which is VirusTotal [33]. The script sends a get request to the API to scan the hash of a malicious file. Before any features can be extracted from the email datasets the output file where the features will be stored need to be created and configured. This is done by the script data_config.py. The datasets are also broken up so each email can be individually distinguished and then sent to

another script gen_features.py. At this script, a single email is inputted, and all the features are extracted and then added to a new dataset. There are quite a few different libraries and APIs used by this script starting with the first NRCLex [34] this package is used to measure the quantitative amount of emotion within the text of the email. TextBlob [35] is used to quantify the sentiment polarity and subjectiveness of the text. The library quopri [36] is used to decode string that is encoded in different characters. The CSV writer is used to output the features to the two-dimensional matrix that is stored in a csv file. And finally, the regular expression operator is used to configure text to be displayed in a specific way.

The last remaining script is ml_model.py this is where all the machine learning classification takes place. The pandas [37] library is used to make slight configurations to the final dataset by removing certain redundant columns. The library time is used to measure the amount of time it takes to train and test each ML model. Finally, the remaining labels all represent the different ML models used by the scikit learn API [38].

Figure 10 UML System Design



Development Methodology

Due to the limited time and the structure of this project choosing a development methodology was necessary to guarantee an outcome for this project. Therefore, I chose the waterfall methodology as it was the most suitable based of the aims and objectives for this project. The process for this methodology is broken down into separate distinctive phases and each phase must be completed before moving onto the next phase. Each phase has specific requirements that help clearly define the criteria for its completion. Once each phase is completed an evaluation is done to determine if the project is running on schedule or not. I chose this methodology because of its simplistic design, that makes it easy to manage when meeting specific deliverables and reviewing each phase. The development of this project can be broken down into specific iterations (phases) as shown below:

Iteration 1: Understanding the requirements for this project by learning and researching email classification models. Creating an initial plan with my supervisor Amir Javed and clarifying the required functional requirements for this project. Gathering a dataset of emails and malicious files or hashes.

Iteration 2: Processing the phishing and benign email datasets and converting them into a readable format. Use detection API to scan malicious hashes for malware.

Iteration 3: Extract a list of compiled features from the email datasets and store in CSV matrix.

Iteration 4: Create a range of different machine learning models to classify phishing emails. Evaluate the performance of each model.

Chapter 4: Implementation

This chapter of the report will discuss the implementation of the code, and the tools used to develop them.

Project Structure

This project was implemented and designed using python 3 and there are five separate scripts that are required to run this tool. Each script serves a specific purpose, and some require a specific file to be executed, with these specific files the tool would be unresponsive. The interactive shell used to test and execute this tool was windows PowerShell, although it has not been tested all other interactive command lines should work.

Main_Controller.py – serves as the main interface to interact with this tool through.

malicious_detection.py – used to scan the malicious hashes and requires the file “dataset/ malware/Malware dataset.csv” as an input.

data_config.py – prepares the collection of datasets for feature extraction. This requires the file " dataset/phishing/fradulent_emails.txt" and the collection of files in the folder “dataset/ benign/”

gen_features.py – all features are extracted from the email variable passed into this script and the results are outputted to one of two files “feature_matrix/matrix_Phish.csv” or “feature_matrix/matrix_Benign.csv”.

ml_model.py – this file merges the file “matrix_Phish.csv” and “matrix_Benign.csv” and creates the file "feature_matrix/merged_data.csv". This merged dataset is then used to train, test, and evaluate different machine learning algorithms.

Data Preparation and Processing

Phishing and Benign Emails

Loading data

The datasets used for the phishing and benign emails were collected from two separate sources. The phishing dataset was collected from an email corpus on Kaggle [39] and is stored in a single txt file. This dataset contains 3,976 emails and is mostly comprised of Nigerian fraud scams. The method used to load this dataset is done by reading the text file and splitting it up into individual emails. Each email is parsed through an email parser and stored in a list; this gives the data a structured format.

```
#phishing dataset location
DIR_PHISH = os.path.join("dataset","phishing")
emailfile = os.path.join(DIR_PHISH,"fradulent_emails.txt")
#reads through single text file a divides string into seperate emails
with open(emailfile, 'r') as file:
    data = (file.read())
phishing_file = data.split("From r")
phishing_emails = [Parser(policy=default).parsestr(phish) for phish in phishing_file]
```

Figure 11 Loading Phishing Dataset

The benign email dataset was gathered from the spam assassin public corpus, but I only used certain collections to match the number of emails in the phishing dataset. The datasets used were 20030228_hard_ham [40] and both 20030228_easy_ham versions [41] [42] this created a total of 4,150 emails. The emails in the easy ham collections we all received from safe sources, but the hard ham collection was designed to be harder to detect as they contain HTML, unusual markup languages, coloured text, and spam like sounding phrases. The benign dataset is stored differently to the phishing emails, each benign email is stored in an individual file, so a different method was used to process the data. The processing method first locates all the individual files, then parses them through an email parser and then stores them in a list.

```
#benign dataset location
DIR = os.path.join("dataset")
DIR_NORM = os.path.join("dataset", "benign")
#converts data to a structured email
def open_email(phishing, filename, file_dir=DIR):
    directory = "phishing" if phishing else "benign"
    with open(os.path.join(file_dir, directory, filename), "rb") as file:
        return email.parser.BytesParser(policy=email.policy.default).parse(file)
#creates a list of structured emails
benign_filename = [name for name in sorted(os.listdir(DIR_NORM)) if len(name) > 10]
benign_emails = [open_email(phishing=False, filename=name) for name in benign_filename]
```

Figure 12 Loading Benign Dataset

In both figure 11 and 12 the email parser API is used to convert the unstructured email string into a structured email format. The purpose for using this parser is that it formats the email into a structure that makes it easier to manipulate and query. Emails contain multiple parts and subparts but changing the format allows the email to be indexed by the header names and any other sub-messages within the email. This helped with extracting certain features from the datasets. Both implementations for importing the separate dataset collections have been uniquely designed to support each dataset. If the file containing the phishing emails are changed then the scripts will have to be modified. If the folders containing the datasets are changed then they will have to be modified in the script too. This implementation method is only designed to be a minimum viable product, if more datasets are intended to be added then a new implementation method will need to be created to support varying

collections. Within its current state if any modification is made to the files or names then the tool will not function properly, it will likely throw error or become unresponsive.

```
import email
from email.parser import BytesParser, Parser
from email.policy import default
```

Figure 13 Email Parser Import

Feature extraction

The next stage in data preparation was extracting multiple features from the phishing and benign structured datasets. This data would then be used for the machine learning classification models. In total there are 76 individual columns in the final dataset of extracted features but there are 30 unique features measured. Some feature results had to be spread across multiple columns because it was the most effective solution for measuring them numerically. The following features are:

Receivers – this feature is a measurement of the number of recipients that received the email.

CC – the feature is also a measurement of the number of receipts that were CCd in the email.

Subject – this feature checks to see if a subject heading is present. The numerical value is stored as a Boolean value, 1 if present and 0 if not.

Forward – the value for this measurement is also a boolean value which checks to see if the email has been forwarded. This is done by checking the subject heading if it starts with “fw”.

Reply – the value for this measurement is also a boolean value too as it checks to see if the email is a reply. The check is done by searching the subject header for the string “re:”.

Length – this feature is the length of the email which is calculated by measuring the number of characters that are found in the body and sub-messages of the email.

Priority – this feature checks to see if a priority level was set when the email was sent. There are 5 different possible levels that can be used. 1 is the highest priority, 2 is high, 3 is the normal standard, 4 is low and 5 is the lowest. This value is assigned to one of three different headings, X-Priority, X-MSMail-Priority, and Importance.

```
def priority_level(email):
    priority_matrix = []
    priority_a = str(email["X-Priority"]).lower()
    priority_b = str(email["X-MSMail-Priority"]).lower()
    priority_c = str(email["Importance"]).lower()
    all_priority = ''.join([priority_a, priority_b, priority_c])

    if ("nonenonenone" in all_priority):
        priority_matrix.extend([3])
    elif ("1" in all_priority) or ("highest" in all_priority):
        priority_matrix.extend([1])
    elif ("2" in all_priority) or ("high" in all_priority):
        priority_matrix.extend([2])
    elif ('3' in all_priority) or ("normal" in all_priority):
        priority_matrix.extend([3])
    elif ("4" in all_priority) or ("low" in all_priority):
        priority_matrix.extend([4])
    elif ("5" in all_priority) or ("lowest" in all_priority):
        priority_matrix.extend([5])
```

Figure 14 Priority Level

HTML – another feature that is also checked in each email is if there is any HTML code in the content. This is recorded as a Boolean value so if any HTML content is detected the feature is labelled 1 and 0 if none is detected.

ContentParts – Within each email there is one main content type used which is "text/plain" however some emails contain multiparts which is when multiple different content types are used. This feature counts the number of content types used.

```
#check content type
contents = [part.get_content_type() for part in email.walk()]
matrix.extend([len(contents)])
```

Figure 15 Checks Content Type

ifMultipart – this feature is like the one above however this creates a Boolean value of 1 if more than one part is detected and 0 if only one part is used.

Attachment – this feature scans each part within the email and checks if any attachments were added. The numerical value is labelled 1 if there are attachments and 0 if there are none.

URL – phishing attacks occur commonly by sending malicious URLs to people, so detecting if any URLs were present is important. If any weblinks or URLs were detected, then this feature is labelled 1 and 0 is none are detected.

Emoji – this feature checks the content within the email to see if any emojis or emoticons are used. This detection is performed using two different methods, the first imports the emoji API and uses a

regex parser to detect if any string within the email matches and emoji. The second method used imports a library called emotlib [43]. Multiple for loops are used to check if there are emojis or emoticons within the content of the email. If something is detected the feature is labelled 1 and 0 if nothing is detected.

```
#check for Emoji's
content_emoji = (str(convert_content(email)))
check_emoji = bool(emoji.get_emoji_regexp().search(content_emoji))
emoji_count = 0

if check_emoji == True:
    emoji_count += 1
for emojiE in emotlib.EMOTICONS:
    if emojiE in content_emoji:
        if len(emojiE) >= 1:
            emoji_count += 1
for emojiE in emotlib.EMOJIS:
    if emojiE in content_emoji:
        emoji_count += 1
if emoji_count != 0:
    matrix.extend([1])
else: matrix.extend([0])
```

Figure 16 Checks for Emoji's

Phone Number – if any phone numbers are detected this feature is labelled 1 and 0 if none are detected. This feature extraction was done with regular expression operators using the function findall. As numbers all around the world are structured slightly differently a series of multiple patterns were used so most international phone numbers could be detected.

Advertisements – this feature checks if the email is an advertisement by searching the main body of text for any references to any subscriptions or mailing lists. The results are labelled as a Boolean value 1 if advertisements detected and 0 if not.

Images – if there are any images attached or within the email this feature will be labelled 1 and 0 if none are detected. This detection is done by scanning each subpart of the email for any images.

Punctuation – for this feature's extraction it took up 33 separate columns, because each column represents the number of times one specific punctuation is used. For example, one column counts the number of times an exclamation (!) mark is used. This was the only logical and unbiased method I could come up with to quantify this feature.

```
def punctuation_level(content):
    pun_matrix = []
    punctuation = "!\"#$£%&'()*+,-./:;<=>?@[\\]^_`{|}~"
    for i in punctuation:
        if (content.find(i)):
            pun_matrix.extend([content.count(i)])
        else: pun_matrix.extend([0])
    return(pun_matrix)
```

Figure 17 Punctuation Feature Extraction

Emotion – for detecting emotion within the message of the email I used a tool known as the NRC Word-Emotion Association Lexicon (often shortened to NRC Emotion Lexicon) [34]. This package is used to measure the emotional affect from a body of text by using a dictionary consisting of approximately 27,000 words associate certain words with emotions and sentiments. The eight emotions used are Anger, Fear, Trust, Surprise, Sadness, Disgust, Joy, Anticipation and the two sentiments are positive and negative. Hence why it takes up 10 columns. Within the framework of this package, it uses the NLTK library's WordNet synonym set to group the Nouns, Adjectives, Adverbs and Verbs. This package has been used in many research papers which proves its accuracy [44] with emotion detection. The eight emotions used in nrclex takes inspiration from Robert Plutchik's Wheel of emotions which is a psychoevolutionary classification methodology for classifying emotions into one of eight classes [45]. The numerical value given to each emotion is a valued score of intensity between 1 and 0.

```
#check for Emotions
text_object = NRCLex(translated.lower())
freq = text_object.affect_frequencies
li = text_object.affect_dict
try:matrix.extend([freq['anger']])
except:matrix.extend([0])
try:matrix.extend([freq['fear']])
except:matrix.extend([0])
try:matrix.extend([freq['trust']])
except:matrix.extend([0])
```

Figure 18 Snippet of NRCLEX emotions

Sentiment – for detection sentiment a second evaluation reading was measured using TextBlob. Text Blob is a python library for Natural Language Processing, that returns the polarity and subjectivity of a body of text [46]. The Polarity sentiment is a measurement intensity between the negative and positive where negative is -1 and positive is 1. The Subjectivity sentiment is a quantified amount that represents the amount of personal opinion to factual information within the text. The higher given

value within the text means more personal opinion is present to factual information. This measurement value lies between 0 and 1.

```
#check for Sentiment for body
matrix.extend([TextBlob(translated).sentiment.polarity])
matrix.extend([TextBlob(translated).sentiment.subjectivity])
```

Figure 19 TextBlob Sentiment

Languages – the final feature extracted is languages as some of the emails in both phishing and benign datasets contained different languages. To also note for all the previous feature extractions around text analysis the package langdetect [47] was used to detect if the content of the email was in another language. Langdetect was also tested, and it showed to have high accuracy with language detection [48]. If the text were not English, then it would be translated to English using the Google Translator API [49] because the other packages like NRCLex and TextBlob require the language input to be English.

Regarding the extracted feature because languages cannot be numerically measured or valued, I had to create a solution to make this possible. I did this by storing the detected language code (ISO 639-1) in the matrix until the full extraction had completed. After the extraction completed, new columns were created for each detected language within the dataset and Boolean values of 1 or 0 were assigned if the language was present.

```
#language detect
detect_languages = []
blob = TextBlob(check_content)

if (bool(check_content.strip())) == False:
    matrix.extend(['en'])
else:
    for sentence in blob.sentences:
        try:
            detect_languages.extend([detect(str(sentence))])
        except: continue
    if (email['Content-Transfer-Encoding'] == 'base64') and (len(detect_languages) == 1):
        matrix.extend(['en'])
    elif (email['Content-Transfer-Encoding'] == 'base64') and ('en' in detect_languages):
        matrix.extend(['en'])
    else: try:
            detected_lang = detect(check_content)
        except:
            detected_lang = 'en'
        matrix.extend([detected_lang])
```

Figure 20 Language Detection

Extracted Features

```
From Steve_Burt@cursor-system.com  Thu Aug 22 12:46:39 2002
Return-Path: <Steve_Burt@cursor-system.com>
Delivered-To: zzzz@localhost.netnoteinc.com
Received: from localhost (localhost [127.0.0.1])
    by phobos.labs.netnoteinc.com (Postfix) with ESMTP id BE12E43C34
    for <zzzz@localhost>; Thu, 22 Aug 2002 07:46:38 -0400 (EDT)
Received: from phobos [127.0.0.1]
    by localhost with IMAP (fetchmail-5.9.0)
    for zzzz@localhost (single-drop); Thu, 22 Aug 2002 12:46:38 +0100 (IST)
Received: from n20.grp.scd.yahoo.com (n20.grp.scd.yahoo.com
    [66.218.66.76]) by dogma.slashnull.org (8.11.6/8.11.6) with SMTP id
    g7MBkTZ05087 for <zzzz@spamassassin.taint.org>; Thu, 22 Aug 2002 12:46:29 +0100
X-Egroups-Return:
sentto-2242572-52726-1030016790-zzzz=spamassassin.taint.org@returns.groups.yahoo.com
Received: from [66.218.67.196] by n20.grp.scd.yahoo.com with NNFMP;
    22 Aug 2002 11:46:30 -0000
X-Sender: steve.burt@cursor-system.com
X-Apparently-To: zzzzteana@yahoogroups.com
Received: (EGP: mail-8_1_0_1); 22 Aug 2002 11:46:29 -0000
Received: (qmail 11764 invoked from network); 22 Aug 2002 11:46:29 -0000
Received: from unknown (66.218.66.217) by m3.grp.scd.yahoo.com with QMQP;
    22 Aug 2002 11:46:29 -0000
Received: from unknown (HELO mailgateway.cursor-system.com) (62.189.7.27)
    by mta2.grp.scd.yahoo.com with SMTP; 22 Aug 2002 11:46:29 -0000
Received: from exchange1.cps.local (unverified) by
    mailgateway.cursor-system.com (Content Technologies SMTPRS 4.2.10) with
    ESMTP id <T5cde81f695ac1d100407d@mailgateway.cursor-system.com> for
    <forteana@yahoogroups.com>; Thu, 22 Aug 2002 13:14:10 +0100
Received: by exchange1.cps.local with Internet Mail Service (5.5.2653.19)
    id <PXX6AT23>; Thu, 22 Aug 2002 12:46:27 +0100
Message-Id: <5EC2AD6D2314D14FB64BDA287D25D9EF12B4F6@exchange1.cps.local>
To: "'zzzzteana@yahoogroups.com'" <zzzzteana@yahoogroups.com>
X-Mailer: Internet Mail Service (5.5.2653.19)
X-Egroups-From: Steve Burt <steve.burt@cursor-system.com>
From: Steve Burt <Steve_Burt@cursor-system.com>
X-Yahoo-Profile: pyruse
MIME-Version: 1.0
Mailing-List: list zzzzteana@yahoogroups.com; contact
    forteana-owner@yahoogroups.com
Delivered-To: mailing list zzzzteana@yahoogroups.com
Precedence: bulk
List-Unsubscribe: <mailto:zzzzteana-unsubscribe@yahoogroups.com>
Date: Thu, 22 Aug 2002 12:46:18 +0100
Subject: [zzzzteana] RE: Alexander
Reply-To: zzzzteana@yahoogroups.com
Content-Type: text/plain; charset=US-ASCII
Content-Transfer-Encoding: 7bit

Martin A posted:
Tassos Papadopoulos, the Greek sculptor behind the plan, judged that the
 limestone of Mount Kerdylio, 70 miles east of Salonika and not far from the
 Mount Athos monastic community, was ideal for the patriotic sculpture.

 As well as Alexander's granite features, 240 ft high and 170 ft wide, a
 museum, a restored amphitheatre and car park for admiring crowds are
planned
---------------------
So is this mountain limestone or granite?
If it's limestone, it'll weather pretty fast.

------------------------ Yahoo! Groups Sponsor ---------------------~-->
4 DVDs Free +s&p Join Now
http://us.click.yahoo.com/pt6YBB/NXiEAA/mG3HAA/7gSolB/TM
---------------------------------------------------------------------~->

To unsubscribe from this group, send an email to:
forteana-unsubscribe@egroups.com



Your use of Yahoo! Groups is subject to http://docs.yahoo.com/info/terms/
```

Figure 21 Benign Raw Email Example

The screenshot above is an example of how each email is structured in its raw form. Everything under line 47 is the content within the message of the email and everything above are the different headers and features regarding the email. Table 1 shows the data of all the features that were extracted from the email in Figure 21. The only feature in the table that is later switched to a binary value is Languages.

Table 1 Feature Extraction Example

Receiver	2
CC	0
Subject	1
Forward	0
Reply	1
Length	899
Priority	3
HTML	0
ContentParts	1
ifMultipart	0
Attachment	0
URL	1
Emoji	0
Phone_Number	0
Advertisments	1
Images	0
anger	0
fear	0
trust	0.166667
surprise	0
sadness	0

disgust	0
joy	0.083333
anticipation	0.25
positive	0.416667
negative	0.083333
polarity	0.134333
subjectivity	0.637333
Languages	en
Punctuation_!	2
Punctuation_"	0
Punctuation_#	0
Punctuation_$	0
Punctuation_£	0
Punctuation_%	0
Punctuation_&	1
Punctuation_'	3
Punctuation_(	0
Punctuation_)	0
Punctuation_*	0
Punctuation_+	1
Punctuation_,	9

Punctuation_-	139
Punctuation_.	8
Punctuation_/	12
Punctuation_:	4
Punctuation_;	0
Punctuation_<	0
Punctuation_=	0
Punctuation_>	2
Punctuation_?	1
Punctuation_@	1
Punctuation_[	0
Punctuation_\	0
Punctuation_]	0
Punctuation_^	0
Punctuation__	0
Punctuation_`	0
Punctuation_{	0
Punctuation_|	0
Punctuation_}	0
Punctuation_~	2

Malware detection

For detecting malware from email attachments, I required a dataset of malicious files or the hashes of malicious files. For this project I choose to download only the hashes as I did not want to run the risk by downloading a PE file (portable executable) and potentially corrupt my own workspace. As I mentioned previously the dataset chosen for detecting malware was found on Kaggle [28]. My reason for choosing this dataset was because of its small size and because it was created in the last 3 years. This dataset was only needed to develop the functionality to show this proof of concept works and that it can be detected.

To use the VirusTotal API you are required to provide an API key. To acquire a key, you must sign up and request one, there are two keys available, one is free of charge but has some limitations and the other requires payment, but it has no limitations. The limitations to using the free key is that you are

limited to making 4 requests per minute. Hence why in the figure bellow there is a sleep command which waits 60 seconds after 4 requests have been made. When a file or hash is sent to the API there is a followed response in json specifying how many different systems detected malicious intent. When making a request only 3 parameters are required, the APIs URL, a key and a file or hash.

```
for m in malwares[0:]:
    count+=1
    params = {'apikey': api, 'resource': m[0]}
    response = requests.get(url, params=params)
    stringval = "'detected': True"
    if stringval in str(response.json()):
        print(" ",counter,"/50 Malicious Files", m[0], end='\r')
        fPass+=1
    else:
        fFail+=1
    counter+=1
    if count == 4:
        time.sleep(60)
        count = 0
```

Figure 22 VirusTotal API

Machine Learning Classification

Train and Test split

After all features are extracted from both phishing and benign datasets they are then merged, and all phishing emails are labelled 1 in the final column and benign emails are labelled 0. This is a required step for preparing the data for classification. There are four variables that are required to develop any machine learning classification model they are X train, Y train, X test and Y test. The X variables together are a matrix of the entire dataset except for the last column that specifies if an email is phishing or benign. The Y variables together are a list for the classification output of the variables in the X matrix. The two X train and Y train variables are used to train the machine learning algorithms to create a model that can classify between phishing and benign emails. Once the machine learning models have been trained the X test and Y test variables are used to evaluate the effectiveness of the machine learning models classification. When splitting up the train and test variables there are many different ratios that can be used but the three most common splits are 80% train to 20% test, 67% train to 33% test and 50% train to 50% test [50].

The scikit learn utility that I used to make this split was called train_test_split() [51]. For my application I parsed in four parameters, the first was an entire dataset except for the last column, the second was the other part of the dataset that was not listed, the third was the test size and the last was a shuffling value. The test size parameter represents the proportion ratio of the test and training split. The

shuffling parameter is called random_state and this controls the shuffling applied to the data before it is split the two most popular passed integers are 0 and 42 [52].

```
#variables for training and testing ML models
x_train, x_test, y_train, y_test = train_test_split(dataframe.iloc[:,:-1],
    dataframe['malicious'], train_size = 0.67, random_state = 42)
```

Figure 23 Train Test Split

To decide the most suitable train to test split ratio to use against the dataset I performed a series of small tests against a few machine learning algorithms to decide which split yielded the best score and results. The two algorithms used for this demonstration was the Linear Support Vector Classification and the Decision Tree. The results show that the higher the training value the less labels are incorrectly classified however the scikit learn scores are relatively similar for each ratio in both algorithms. From analysing the number of mislabelled points in both tables the distance between the training values from 80 to 67 is much smaller than the distance between 67 and 50. The highest scoring split ratio for both algorithms is 67% training and 33% testing therefore this will be the ratio to use when testing and evaluating the remaining models.

Train	Test	Score	Mislabelled Points
80	20	0.9544	74
67	33	0.9597	108
50	50	0.9576	172

Table 2 Train Test Split: Linear Support Vector

Train	Test	Score	Mislabelled Points
80	20	0.9741	42
67	33	0.9772	61
50	50	0.9751	101

Table 3 Train Test Split: Decision Tree

Linear Support Vector Classification

The first classification model used is the Linear Support Vector Classification model. This model variant uses the implementation of liblinear which is a classifier for data with large amounts or instances and features. An advantage this gives is more flexibility in choice of penalties and can scale better with more samples. This classification support is handled using a one vs the rest scheme which is a heuristic method that uses binary classification algorithms for multi-class classification. The purpose for using random state as a parameter is to help get a more accurate score for the same instance if executed multiple times. This is done by setting a numerical value for the seeded state in this case 42 which

helps the machine learning algorithm select similar random instances upon execution. For this model it was required to import the module LinearSVC to use this classification algorithm.

```
## SVM Kernel: linear
def SVMLinear():
    from sklearn.svm import LinearSVC
    linear = LinearSVC(random_state=42)
```

Figure 24 Linear Support Vector Classification Code

Logistic Regression

This logistic regression model uses a linear model to classify an output. It can handle both dense and sparse inputs of data. For this model there are two parameters parsed in the solver which is used to specify the algorithm to use in the optimisation problem and the maximum iterations. The default multi class solver used by this model is the one vs the rest scheme and the liblinear optimiser just like the heuristic method used in the Linear Support Vector Classification model. The purpose for setting a maximum iteration is to help the accuracy by increasing the number of iterations it takes for the solver to converge. If nothing is set the default maximum iteration value is set to 100.

```
## Logistic Regression
def LogReg():
    from sklearn.linear_model import LogisticRegression
    logreg = LogisticRegression(solver='liblinear',max_iter=600)
    start_train = time.perf_counter()
```

Figure 25 Logistic Regression Code

In table 4 shows the results of the number of mislabelled point and the scikit learn score per max iteration. For this model's implementation the maximum iteration is set to 600 because that produced the highest score and had the least mislabelled points, increasing it any further showed to be redundant.

Max Iteration Value	Score	Mislabelled Points
0	0.5085	1318
100	0.9548	121
200	0.9563	117
300	0.9608	105
400	0.9664	90
500	0.9668	89
600	0.9720	75
700	0.9720	75

Table 4 Logistic Regression Max Iteration Value

Naive Bayes: Bernoulli

For the naïve bayes model there were a few varying options available but the algorithm I went with was the Bernoulli. This model uses multivariate for its classification and is best suited for discrete sets of data. This model may not perform as well as the compared to the other classification models this is because this algorithm is designed and more suited towards Boolean features.

```
## Naive Bayes: Bernoulli
def NaiBay():
    from sklearn.naive_bayes import BernoulliNB
    bnb = BernoulliNB()
```

Figure 26 Naïve Bayes: Bernoulli Code

Decision Tree Classifier

When selecting a decision tree model to use there were two available options a classifier and regressor. For this project selecting the classifier was the most logical choice because the results need to be classed into one of two groups. This classifier it uses “gini” impurity to measure the likelihood of incorrectly classifying a random variable and within the tree it creates the minimum leaf samples required at each node is 1. The minimum leaf samples can be increased if needed.

```
## Decision Tree Classifier
def DeciTree():
    from sklearn import tree
    dtClass = tree.DecisionTreeClassifier()
```

Figure 27 Decision Tree Code

Random Forest Classifier

This classifier requires importing the module random forest from the sklearn ensemble. The default number of trees this classifier creates is 100, by increasing this value can help increase the accuracy but increases the time it takes to compute. Another default parameter in this function is that the whole dataset is not used to build each tree. It creates its own class weighting for each feature this could create unbalanced weights for each class.

```
## Random Forest Classifier
def RanFor():
    from sklearn.ensemble import RandomForestClassifier
    rdClass = RandomForestClassifier()
```

Figure 28 Random Forest Code

Neural Network

For the neural network classifier, the implemented module was a multi-layer perceptron classifier. The quantity of hidden layers used to represent the number of neurons is 100 and the activation

function for each hidden layer is the rectified linear unit. The solver used to create the weight optimisation is referred to as “adam” which is the stochastic gradient-based optimizer this is known to work well on weighted large datasets. The constant learning rate used to optimise the weight classes is set to 0.001. For using random state, it is the same purpose as mentioned before which is to facilitate constant accurate score.

```
## Logistic Regression
def LogReg():
    from sklearn.linear_model import LogisticRegression
    logreg = LogisticRegression(solver='liblinear',max_iter=600)
    start_train = time.perf_counter()
```

Figure 29 Neural Network Code

The maximum number of iterations set is 40 this determines the maximum number of times it takes to converge to an output. This specific number was chosen by testing various values from 10 to 1000 table 5 shows the results from this test. The results in the table are very interesting any value equal to 50 and above will have the same result. However, the value 40 was the most accurate and the value 30 was the least accurate by a significant amount.

Max Iteration Value	Score	Mislabelled Points
10	0.9649	94
20	0.9336	178
30	0.9086	245
40	0.9817	49
50	0.9660	91
60 <	0.9660	91

Table 5 Neural Network Max Iteration Value

K-Nearest Neighbors

The final classifier used is K-Nearest Neighbor. The default algorithm used to compute the nearest Neighbor is a mixture of different appropriate methods, they are BallTree, KDTree and brute force. The class weighting for this function uses a uniform weight by setting all features to be equal. The default value for n-neighbors is 5 but for this algorithm the value one was used.

```
## Nearest Neighbors
def NearNeigh():
    from sklearn.neighbors import KNeighborsClassifier
    nbrs = KNeighborsClassifier(n_neighbors=1)
```

Figure 30 K-Nearest Neighbor Code

From testing varying values ranging from 1 to 50 the most effective parameter value for n-neighbors for this classification method is 1. The number 1 is the best value and the lowest value that can be used as a parameter, from the evidence shown in table 6 as value increases the results get worse.

N-Neighbor Parameter Value	Score	Mislabelled Points
1	0.9067	250
2	0.8978	274
3	0.8851	308
4	0.8825	315
5	0.8743	337
6	0.8762	332
7	0.8642	364
8	0.8635	366
9	0.8560	386

Table 6 N-Neighbor Parameter Value

Evaluation used

For the evaluating the results and performance varying measurements and methods were used. The function perf counter was used to measure the time it takes to train each model and test each model. This function was used over time clock because it is more precise with measuring the time in seconds. From this measurement we can infer the effectiveness of the model based of its accuracy and the time it requires.

```
start_train = time.perf_counter()
linear.fit(x_train, y_train)
end_train = time.perf_counter()
```

```
start_test = time.perf_counter()
y_pred_class = method.predict(x_test)
end_test = time.perf_counter()
```

Figure 31 Perf Counter Time Taken

In Scikit Learn there is a function called score which compares the predictions of the model against real labels. The percentage produced is a score of the accuracy of the training used within the model that makes correct predictions when using the testing datasets.

```
print(" Score: ", method.score(x_test, y_test))
```

Figure 32 Scikit Learn Score

Another performance indicator used to evaluate the results for each model is the confusion matrix. The function used to create this matrix is imported from scikit learn metrics, from this matrix we can infer the True Positives and Negatives and the False Positives and Negatives. Using these classifiers,

we can compute other performance indicators that can determine the effectiveness for each machine learning algorithm.

```
#get confusiton matrix
confusion = metrics.confusion_matrix(y_test, y_pred_class)
TP = confusion[1, 1]
TN = confusion[0, 0]
FP = confusion[0, 1]
FN = confusion[1, 0]
```

Figure 33 Confusion Matrix

The performance indicators that can be computed from the confusion matrix are Accuracy, Precision, Recall, and F-Measure. These metrics are all calculated using specific equations and the statistics of the True Positives and Negatives and the False Positives and Negatives.

```
Precision = (TP / (TP + FP))
Recall = (TP / (TP + FN))
print(" Accuracy: ", ((TP + TN) / (TP + TN + FP + FN)))
print(" Precision: ", Precision)
print(" Recall: ", Recall)
print(" F-Measure: ", (2 * (Precision * Recall) / (Precision + Recall)))
```

Figure 34 Accuracy, Precision, Recall and F-Measure

The module matplotlib is used to import the function pyplot which is used to create diagrams for each of the machine learning algorithms. The diagrams and charts produced all contain the metric weighting applied to each feature for the different machine learning algorithms. For the produced graphs, the style used is fivethirtyeight, the positive weighted statistics are coloured blue and negative weighted statistics are coloured red. For each algorithm used a graph is produced and saved as a PNG (portable network graphic) and stored within the folder named charts.

```
pyplot.style.use('fivethirtyeight')
fig, ax = pyplot.subplots()
colors = ['red' if c < 0 else 'blue' for c in importance]
ax.bar(features, importance, color=colors)
pyplot.xticks(features, rotation=270)
pyplot.title(title)
pyplot.ylabel("Weightings")
pyplot.xlabel("Features")
figure = pyplot.gcf() # get current figure
figure.set_size_inches(18, 9)
pyplot.savefig("charts/"+title+".png", dpi=300, orientation="landscape",
    bbox_inches="tight", pad_inches=0.3)
```

Figure 35 Pyplot Design

Interface

The final implementation of this software is an interface which was created for the singular purpose in helping the user interact with the different functionalities of this tool. From this interface all the different scripts can be executed, the functionality with scanning malicious files, generating the dataset of email features, and executing the individual machine learning algorithms. To run this tool only the Main_Controller python file needs to be ran, from the command line varying options will be provided to the user after running this file. Only specific numerical values are accepted as inputs if any incorrect values are inputted than an error message is printed telling the user to select a new option. This is not the final interface for this tool it is only a temporary design that is used to show the different functionalities and help with automating this seamless process.

```
def MainMenu():
    print("[1] Scan the Malicious Files")
    print("[2] Generate dataset of email features")
    print("[3] Run Machine Learning Models")
    print("[0] Exit the Program")
def MLMenu():
    print("[1] Run Support Vector Machine: linear ")
    print("[2] Run Logistic Regression")
    print("[3] Run Naive Bayes Bernoulli")
    print("[4] Run Decision Tree")
    print("[5] Run Random Forest")
    print("[6] Run Neural Network")
    print("[7] Run K-Nearest Neighbors")
    print("[8] Run all Algorithms")
    print("[0] Go Back ")
```

Figure 36 Interface Options

Chapter 5: Results and Evaluation

Dataset Evaluation

To ensure that there is an even split of phishing and benign emails within both testing and training data splits is essential for creating a functional and accurate classification algorithm. As seen in figure 52 the split between both email classes is even and the ratio split used is 67% training and 33% testing.

```
Training:
Rows in X train: (5444, 76)
Rows in Y train: (5444,)
Benign       2786
Malicious    2658
Name: malicious, dtype: int64

Testing:
Rows in X test: (2682, 76)
Rows in Y test: (2682,)
Benign       1364
Malicious    1318
Name: malicious, dtype: int64
```

Figure 37 Testing and Training Data Split

Within each model certain features were assigned a higher weighting because they had a bigger impact when classifying the results. Within table 7 and 8 shows a few of those features that were assigned a higher importance than the rest. Table 7 displays the average scores for each emotional measurement with the phishing dataset and benign dataset. All the emotional values are very similar within both datasets except for the feature Trust which has a higher average score within the phishing dataset. This result is likely because phishing scams must be convincing if they are not then they are not effective hence why the average score for trust is higher in the phishing dataset. Many occurrences have taken place where scammers have pretended to be lawyers and tricked people into sending them their private and personal details though fake compensation forms or inheritance documents [53]. With this stolen information the scammer is able to commit many attacks against them such as identity theft.

The five features within table 8 are specific features that commonly appeared with high weighting in many of the algorithms. As seen in most of the previous graphs the feature advertisements are usually weighted the highest and as I suspected there was bias within the dataset. Within the benign data 65% of the emails contained advertisements and 0.4% were contained in the phishing dataset which is why this feature was so heavily weighted in the models. The two other features out of the seven that scored higher within one dataset over the other was the URL feature and the Reply feature. There

are URLs within 86% of the benign emails and 37% within the phishing emails this explains why this feature was set to high importance because it was so heavily weighted on one side. The same one-sided score also affected the feature Reply, 55% within the benign dataset and only 2% within the phishing dataset. This is a likely result having more replies within the benign dataset because within phishing attacks it is very unlikely for messages to be sent back and forth between a user and a scammer.

Table 7 Emotion Feature Average Score

Emotion	Avg Score Phishing	Avg Score Benign
Anger	0.05	0.04
Fear	0.06	0.07
Trust	0.23	0.13
Surprise	0.05	0.04
Sadness	0.04	0.06
Disgust	0.02	0.03
Joy	0.08	0.06
Anticipation	0.10	0.11
Positivity	0.25	0.25
Negativity	0.09	0.15

Table 8 Common Feature Average Scores

Feature	Avg Score Phishing	Avg Score Benign
Advertisements	0.004	0.65
HTML	0.16	0.05
URL	0.37	0.86
Reply	0.02	0.55
Punctuation $	0.12	0.02

Malware Detection Performance

The capability with detecting malicious files was performed by using a dataset consisting of 50 malware hash files and scanning each one using the Virus Total API. As mentioned previously the

detection scan rate is limited to 4 scans per minute so the expected length of time should average between 12 and 13 minutes.

As shown in the figure out of the 50 malicious files each one was detected successfully, and zero files failed to not be detected. However, the time it took to perform this scan was 742 seconds this roughly translates to 12 and half minutes which is quite long. This test was only intended to show that this functionality is possible and that these designs can be used for further research and experimentation.

```
[1] Scan the Malicious Files
[2] Generate dataset of email features
[3] Run Machine Learning Models
[0] Exit the Program

Please Select and option: 1
  50 /50 Malicious Files 025c63d266e05d9e3bd57dd9ebd0abe904616f569fe4e2b78cf2ac52493cb460
  50 Malicious files detected
  0 Malicious failed to detected
  738.3595899999999 Time Taken to complete scan
```

Figure 38 Malware Detection Results

The number of malicious hashes used to test this function is very small so to thoroughly test this systems capability it would require a much large dataset of hashes. A dataset of a thousand hashes consisting of malicious and non-malicious files would be ideal but the execution time to perform this test would take over 4 hours. The results produced in the interface provides sufficient detail regarding the overall success with detecting the malicious files however its usability relies more on its technical functionality that proves it can be used to detect malicious files.

Classification Performance

The main target for this project is to create a system that uses machine learning to classify phishing and benign email. This next section provides the results and evaluation of different machine learning algorithms used for this classification.

Linear Support Vector Classification

The Linear Support Vector Classification model showed to be effective with classifying phishing and benign emails. From the confusion matrix shown in Figure 38 we can see that 4% were falsely categorised and 96% were truthly categorised but the most important figure is the False Negatives. They represent the quantity of phishing emails that were categorised as benign emails, minimising this value is the most important to reduce the possibility of someone getting impacted by a phishing attack.

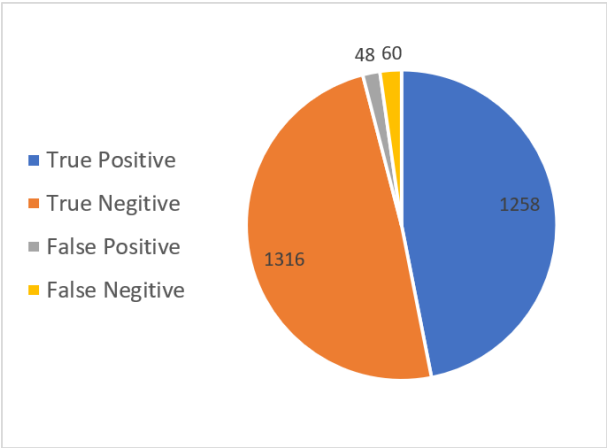


Figure 39 Confusion Matrix: Linear Support Vector Classification

This model applies individual weights to each feature to help with its classification as certain features have a bigger impact on the model. Figure 39 shows the weightings applied to each feature, the positively weighted features assisted with the detection of phishing emails and the negatively weighted features assisted with the detection of benign emails. The top three positively weighted features are all different quantities of punctuation the highest is “<”, then “$” and finally “!”. These features help with the detection of phishing emails, so it is understandable that the dollar sign is one of them as the dataset used is primarily money theft and scam phishing attacks. Regarding the detection of benign emails, the three lowest weighted features that assisted detecting benign emails were advertisements, subjectivity, and replies. Advertisements are heavily weighted opposed to the rest, weighing in at -0.8. This means there is an 80% chance that if an email contains information regarding subscriptions or mailing then it is a benign email. All features that detect emotion are negatively weighted, from this we can infer that this algorithm could not detect patterns of emotion within phishing emails however detecting patterns within the benign emails was easier.

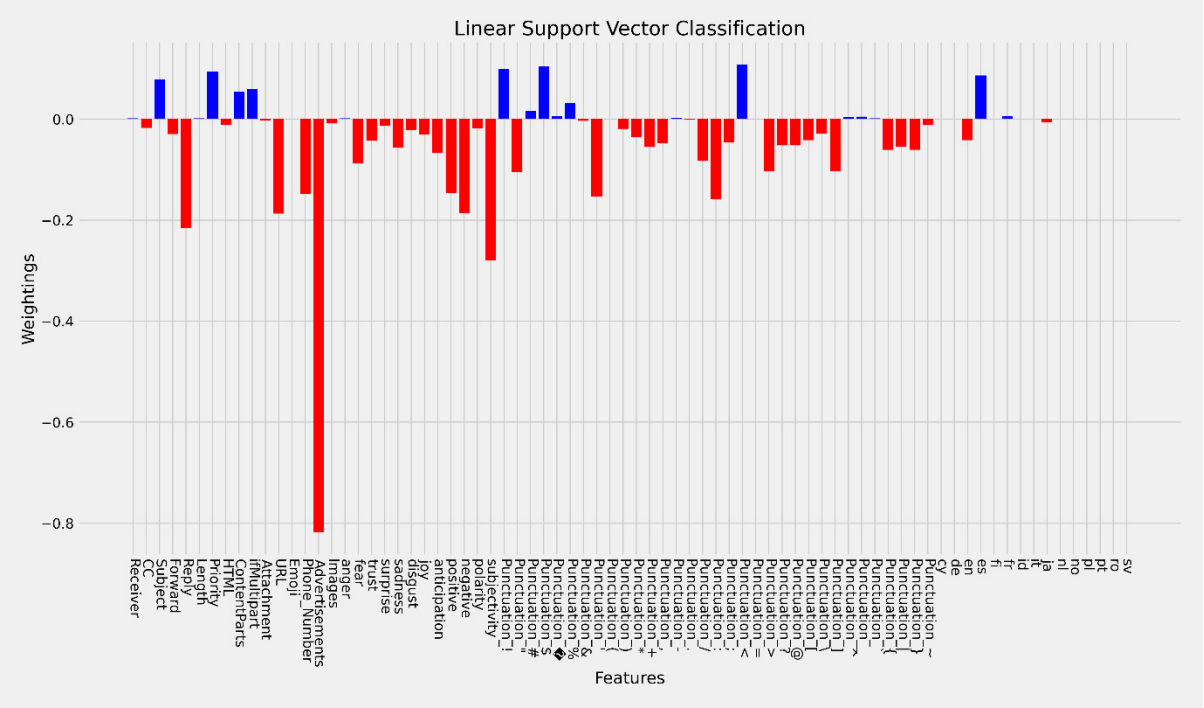


Figure 40 Feature Weighting: Linear Support Vector Classification

Logistic Regression

Moving onto the Logistic Regression model from looking at the figure bellow we can see that this method has also produced an effective classifier for detecting phishing emails. The confusion matrix bellow shows that 3% of the emails were falsely classified and 97% were correctly classified. From all these results 42 phishing emails were categorised as benign emails and 33 emails were incorrectly categorised as phishing.

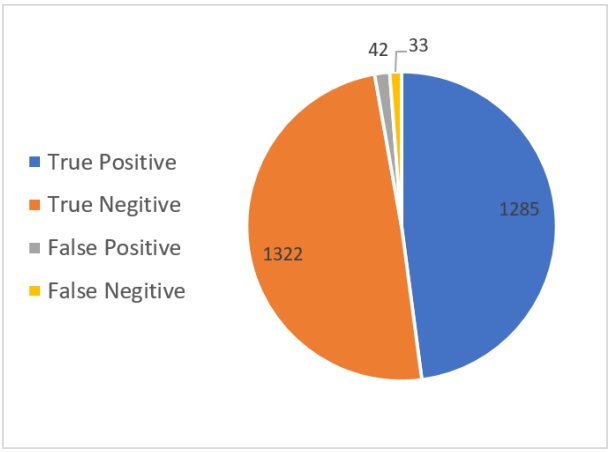


Figure 41 Confusion Matrix: Logistic Regression

If we observe figure 41 it shows each feature's weighting for the logistic regression model. This model has a much larger spread of positively weighted feature to negatively weighted features than the Linear Support Vector model. Like the last model before the same methodology applies the positively

weighted features detect phishing and the negative detect benign. The three most positively weighted features that had the biggest impact were the emotion measurement for Trust, the language French, and the detection of HTML code. From these results it is understandable that trust should have a high score because if a phishing scam is not trustworthy how effective can it be with misleading the user. Regarding the language feature why has this model scored French so highly and in the previous method nil where Spanish is the most positively weighted. Like before the label advertisements have the highest negatively weighted score this may be likely due to bias within the dataset. The highest other two negatively weighted scores were the use of the punctuation "|" and the detection of URLs. Generally, URLs are used in phishing attacks so seeing this feature used to detect benign emails is an interesting result this might again be the result of bias within the used datasets.

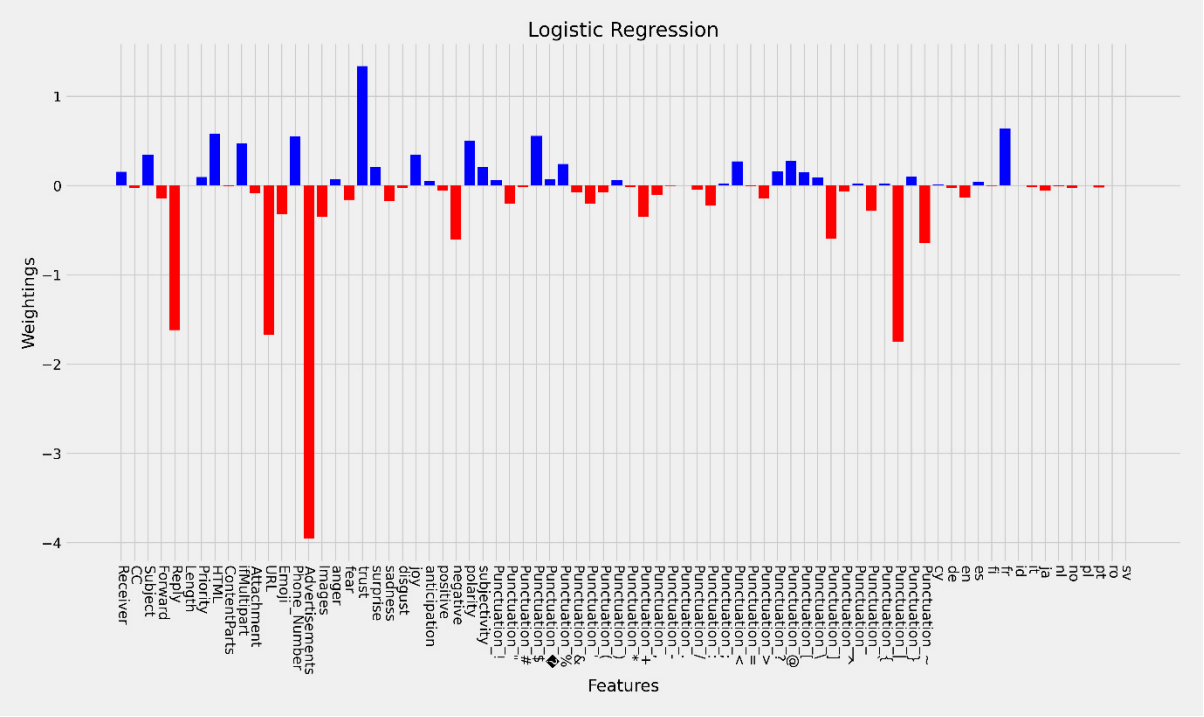


Figure 42 Feature Weighting: Logistic Regression

Naive Bayes: Bernoulli

The Naïve Bayes model also showed to be an effective classifier for distinguishing phishing emails. From observing the confusion matrix in Figure 42 we can see that this model only mislabelled 27 phishing emails as benign which is only 1% and that performs better than the two previous models. The only issue lacking with this model are that the number of False Negatives is nearly double the quantity of False Positives averaging about 2%.

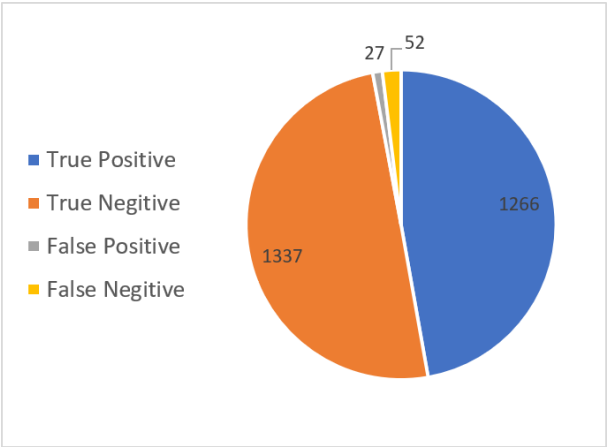


Figure 43 Confusion Matrix: Naive Bayes – Bernoulli

This model, feature weighting is structured differently to the previous algorithms as there are only negatively weighted labels. Many of the features are not heavily weighted because this model requires the dataset features to be stored in a binary format therefore many features in this model were not utilised. The only features used in this algorithm contained ones and zeros, so all the sentiment and emotional features were not used. However, the algorithm still performed effectively with classifying phishing emails. The binary features that were heavily weighted in this algorithm were all the different languages, the forwarded emails, and the punctuation character "^". For this model to be more effective each non-binary class should be separated into multiple binary classes for example the feature length could have been separated into short, medium, and long groups.

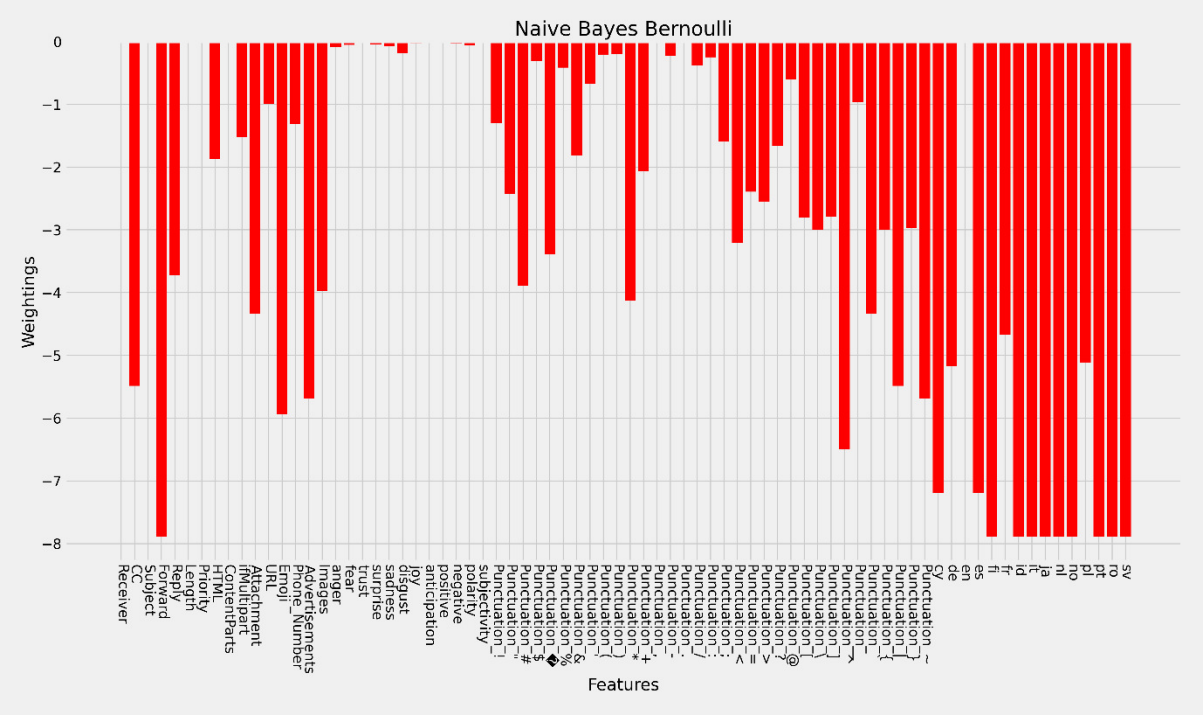


Figure 44 Feature Weighting: Naive Bayes - Bernoulli

Decision Tree Classifier

The decision tree classifier proved to be better at classification than the naïve bayes model overall however this model classified more phishing emails as benign (False Positives). If this algorithm were used, then 41 phishing attacks would have been undetected. Only 3% of the emails were falsely classified which is a good result but it can and should be better.

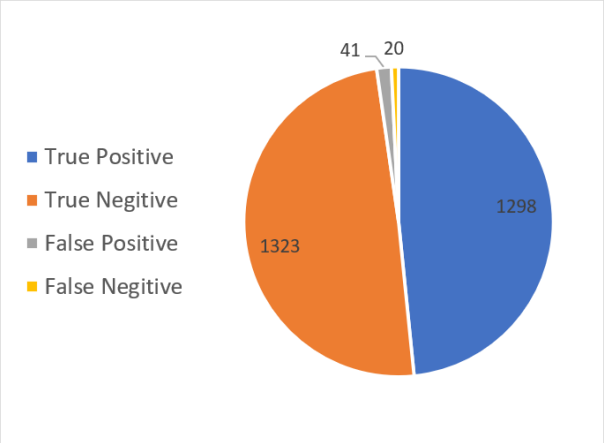


Figure 45 Confusion Matrix: Decision Tree

The weighting applied to the features in this algorithm were heavily biased towards certain features. Only five classes were given any substantial weighting these features were advertisements, length, the emotional measurement trust, and the punctuation quantities “>” and “/”. All other features within the model have not been weighted more than 0.01 which is very low compared to advertisements which is weighted over 0.45. The feature advertisement seems to be an occurring feature used in classifying emails I suspect this could likely be because many of the benign emails are classified as advertisements. Among the emotional features another occurring feature that is heavily weighted is trust, an analysis will be done on certain features within the dataset to see if there is any bias toward benign or phishing emails. The feature length is heavily weighted within this model, and this is the only algorithm that has used this class in all other models it has been disregarded and unused. I would like to understand how this was derived and what steps the algorithm took to classify this features weighting.

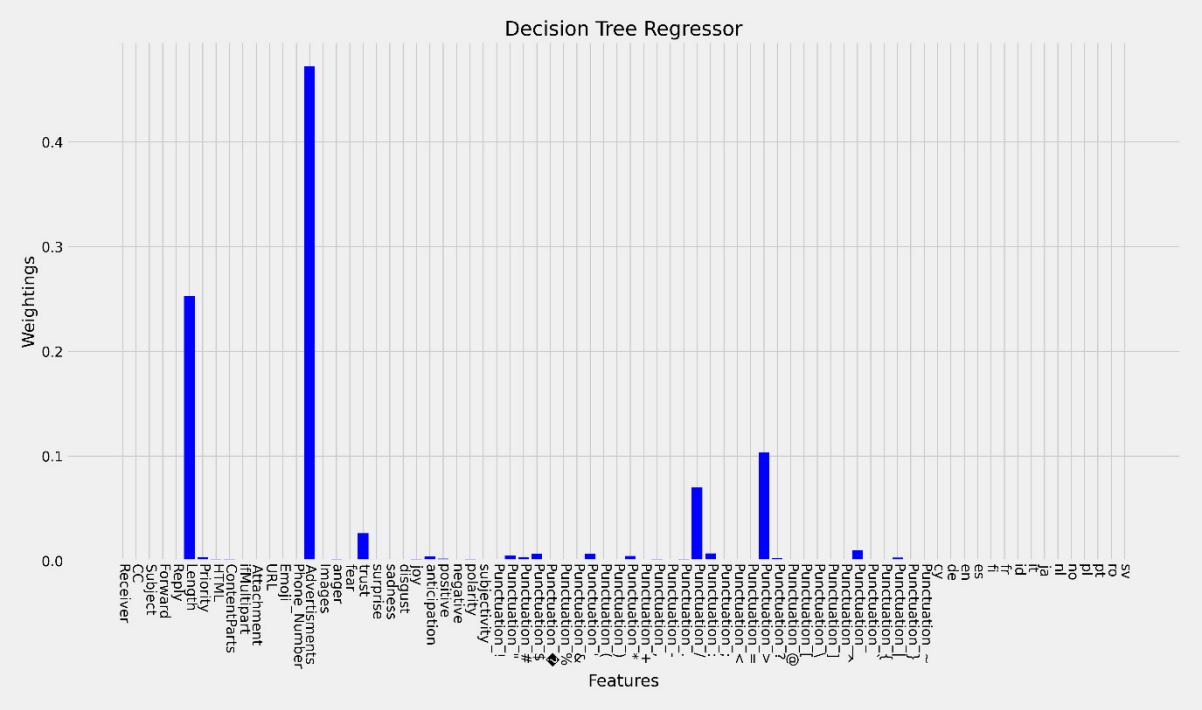


Figure 46 Feature Weighting: Decision Tree

Random Forest Classifier

Of all algorithms used this model produced the best classification results. Only 18 emails were classified incorrectly which is 0.67%, five of those emails were phishing emails which were categorised as benign. It is inevitable for this model to outperform the decision tree in classification but to have achieved these results is impressive. Room for improvement can always be found but for this algorithm it would be difficult as the classification results are so accurate.

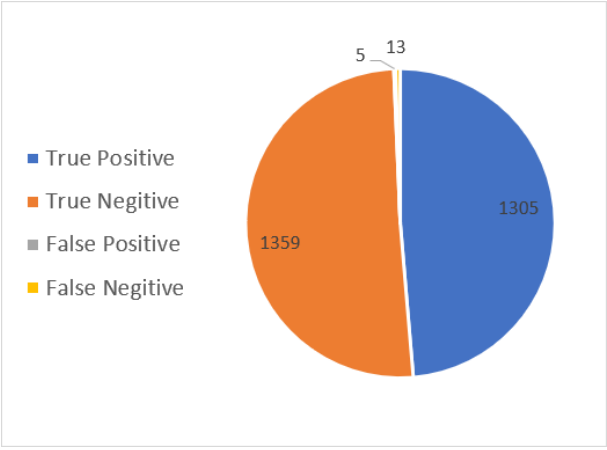


Figure 47 Confusion Matrix: Random Forest

To understand how these results were derived we must look at the feature weighting applied by the algorithm in figure 47. The random forest algorithm has used a more even spread of weighting for each feature. All the weighted features in the decision tree are also repeated within this model too,

the highest weighted feature is advertisements, scoring 0.14 which is around four times smaller than the weighting in the decision tree. However, this model uses other features such as the punctuation for ”$” and “%”, the emotional measurement for Trust, and if the email is a reply. This model was able to perform effectively because every feature has been individually weighted and refined to a state where the algorithm can accurately classify every email with precision. There are a few features within this model that have no weighting set these are the detected languages, emojis, attachments, forward, subject, and receiver.

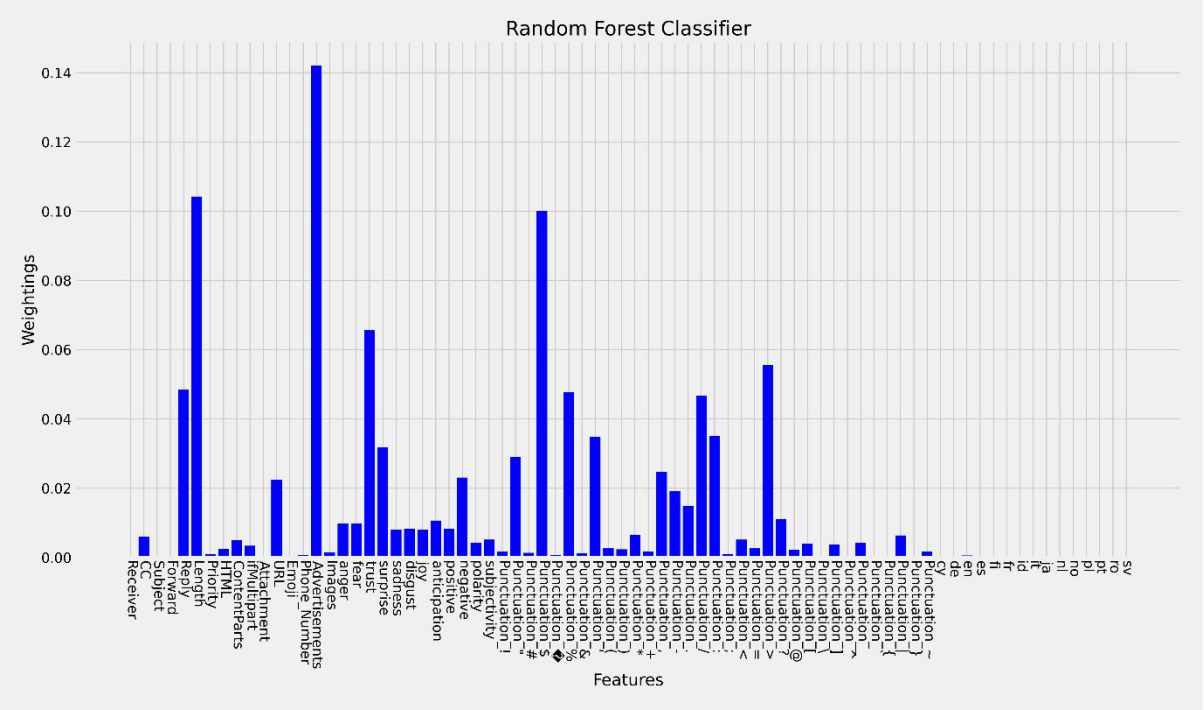


Figure 48 Feature Weighting: Random Forest

Neural Network

The Neural Network classifier has shown to also be an effective algorithm as it has been able to classify 98.5% of emails correctly. Only 49 emails were incorrectly classified, however better results could have been achieved if more parameters were added and refined. From making small adjustments to the maximum of iterations I was able to improve the model’s accuracy and reduce the number of incorrectly classified emails. This algorithm was able to classify benign emails easier than phishing emails as seen in the results. Gathering the features of importance for this algorithm is not easily done because the learning rate optimizer constantly updates and changes the weighting.

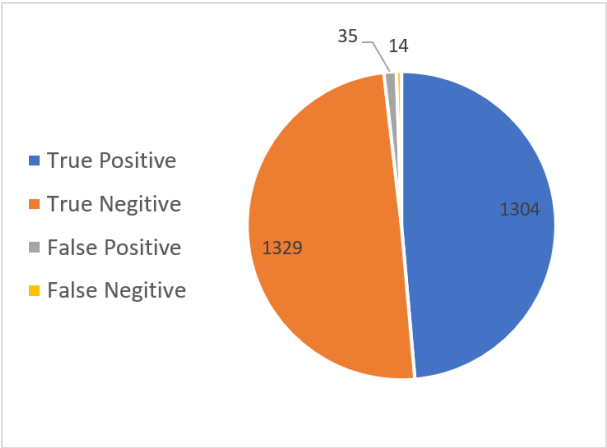


Figure 49 Confusion Matrix: Neural Network

K-Nearest Neighbor

Compared to all models this algorithm performed the worst where a total of 250 emails were incorrectly classified which is 10% of the emails. This is the only model that I would classify as a failure because of the quantity of phishing emails that were undetected. One possible reason as to why this model performed badly might be because it used a uniform weighting to all the features. This model could have performed better if further testing were done on the dataset to see which specific classes made less of an impact. There is no diagram for this model showing the features of importance because all the features were weighted equally.

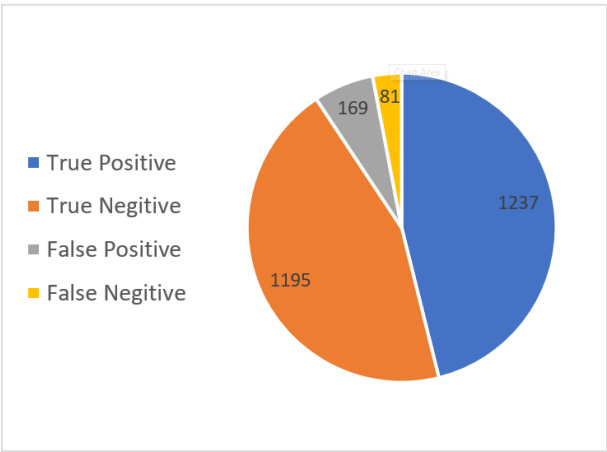
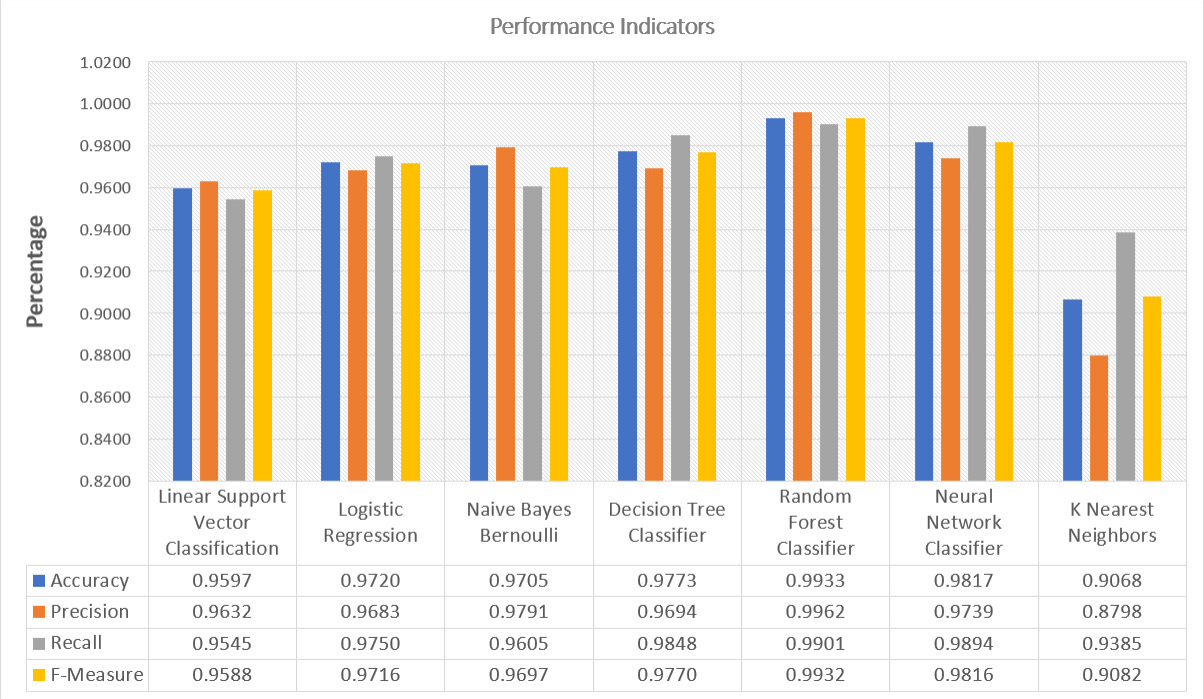


Figure 50 Confusion Matrix: Nearest Neighbor

Performance Indicators

Using the confusion matrix results gathered from each algorithm assisted with producing further performance evaluation indicators. Such as Accuracy, Precision, Recall and F-Measure. The time taken to test and train each algorithm were recorded and within this section further analysis will be performed to determine which classifier is the most effective and suitable.



	Linear Support Vector Classification	Logistic Regression	Naive Bayes Bernoulli	Decision Tree Classifier	Random Forest Classifier	Neural Network Classifier	K Nearest Neighbors
Accuracy	0.9597	0.9720	0.9705	0.9773	0.9933	0.9817	0.9068
Precision	0.9632	0.9683	0.9791	0.9694	0.9962	0.9739	0.8798
Recall	0.9545	0.9750	0.9605	0.9848	0.9901	0.9894	0.9385
F-Measure	0.9588	0.9716	0.9697	0.9770	0.9932	0.9816	0.9082

Figure 51 Performance Indicators

Accuracy

The diagram in figure 50 represents the accuracy for each model, as we can see the Random Forest Classifier has the highest accuracy which scored 99% and the Neural Network Classifier which scored 98%. The model that scored the worst was K-Nearest Neighbor which scored 90% this is likely because 250 emails were incorrectly classified and because a uniform weighting was applied across all features. From the lack of flexibility with assigning individual weighting to each feature the model was limited in the accuracy it could produce. However, every other model that could apply individual weighting achieved a score higher than 95%.

Precision

As expected, the Random Forest Classifier scored the highest precision at 0.99 and the Naïve Bayes algorithm scored the second highest at 0.98 and like previously the Nearest Neighbor model scored the lowest. The Precision scores measurement is a representation of the percentage of positively classed emails. Overall, the neural network is more accurate but regarding detection of phishing emails naïve bayes has more precision with classifying positive values. These results are likely because the positive classified emails are more susceptible to the detection from binary values than percentile values especially as naïve bayes utilises binary values more.

Recall

The Random Forest Classifier scored the highest again with a Recall score of 0.99 however the Neural Network Classifier scored 0.989 which is 0.001 less. The Recall score is the proportional rate of positive

phishing email predications. The two highest scored algorithms were the best at detecting phishing email because they misclassified the least amount of phishing emails which is why this score should be considered as one of the most important measurements in deciding which model performs the best.

F-Measure

Finally, the F-Measure results are the derived value of the mean score of the precision and recall. As expected, the Random Forest Classifier achieved the highest score because it outclassed every other algorithm in classification and K Nearest Neighbor achieved the worst results because it mislabelled the most emails and had the worst accuracy. The Neural Network algorithm scored the second highest score because overall its recall score is very high which made the deciding factor that pushed the F-Measure score close to the score of the Random Forest Classifier.

Time Taken

The elapsed time in testing and training for each model are very different from each other and to their results as seen in Figure 51. As an overall comparison the model that took the least time to train and test is Naïve Bayes which took 0.028 seconds however there were other models that had faster training times or testing times. Across all algorithms the training time is always longer except for K Nearest Neighbor which had a longer testing time than training time, it took 0.004 seconds to train the whole model and a third of a second to test the whole model. The lack in training time is a result of its simple algorithm which is the probable cause of its underperformance. The Neural Network and Logistical Regression Classifier both performed very similar regarding time taking 1.8 seconds to train the model and around 0.005 to test, however in conjunction to the classification performance the Neural Network is more effective. The Linear Support Vector Classification had the most average training time across all the models but the fastest testing time. It was inevitable that the Random Forest Classifier would take longer than the Decision Tree Classifier, but the size of the difference was unknown. From the results we can see that it was ten times longer to train the Random Forest and four times longer to test with a difference of only 2% increase in accuracy. The dataset contains over 8,000 emails if a larger dataset was used that was ten times the size, then as an estimate it would only take 12 to 13 seconds to train the classifier and in retrospect this is not a lot of time especially as training the model is only required once.

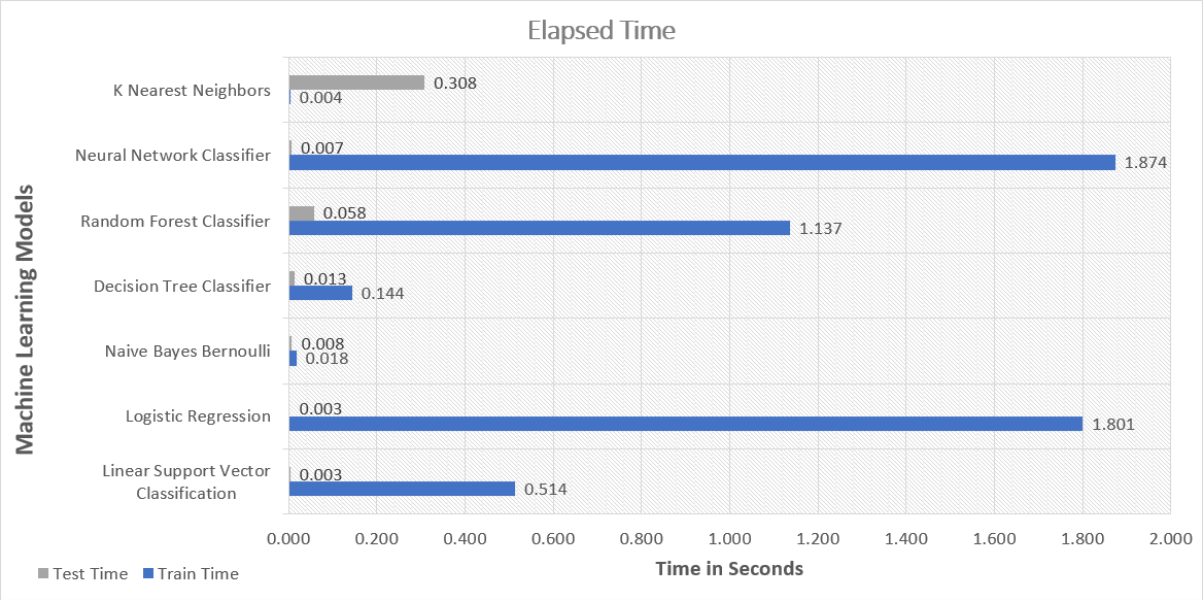


Figure 52 Training and Testing Results

Requirement Fulfilment

Functional

Requirement	Pass / Fail	Evidence
Raw email files must be processed as an input for the extraction features.	Pass	The phishing and benign datasets used are both collections of raw emails that each contain multiple headers and specific information other than just the subject header and email message.
The features generated from each email must be a numerical value.	Pass	All features stored within the datasets are only numerical values as the machine learning models do not accept non numerical values.
Two separate datasets should be created one of Benign emails and another of Phishing emails after the extraction of the features from the data.	Pass	The extraction of features from both datasets are done in separates stages and stored within different output csv files. These files are matrix_Benign.csv and matrix_Phish.csv. They are then merged at a later stage and stored in the file merged_data.csv.
A total of five separate machine learning algorithms should be	Pass	There are a total of seven Machine Learning Classifiers used the first five are: Linear Support

used and compared to derive the classification results.		Vector, Logistic Regression, Naive Bayes Bernoulli, Decision Tree, and Random Forest.
Most emails must be classified into their correct class.	Pass	All algorithms classified most of their emails correctly and they all scored more than 90% accuracy.
The tool must be able to identify malware within malicious files or from malicious hash files.	Pass	The malware detection functionality of the tool can scan the hash of a file distinguish if it contains malicious content or not. All 50 malicious hashes were detected to contain malware.
Extras:		
Two extra machine learning algorithms should be added and compared to the classification process.	Pass	The two remaining machine learning algorithms used are Neural Network and K-Nearest Neighbor.
The software should use a balanced dataset of benign to phishing emails.	Fail (Pass)	The datasets used contains 3976 phishing emails and 4150 benign emails. It is not an exact 50-50 split however it is relatively balanced when rounded to the nearest thousand.
A summary evaluation of each classifying model should be logged.	Pass	The accuracy, precision, recall, and f-measure were all calculated using the data from the confusion matrix from each algorithm. The testing and training values for each model were all recorded.

Non-Functional

Requirement	**Pass / Fail**	**Evidence**
Reliability: the tool needs to be able to compute without any errors and be available all the time. It needs generate a dataset of features, classify phishing	Pass	The system runs without any errors or bugs and can classify between phishing and benign emails. It can also detect malware from scanning malicious hashes.

emails, and detect malicious files without any errors.		
Usability: when running the different capabilities of the system is should be easy to use and to navigate.	Pass	A basic interface was created for this tool so executing the separate functionalities is easy to perform.
Speed: the time it takes to process the datasets, detect the malicious files, and classify the results should be reasonably quick but justifiable if it extends for more time.	Pass	There are three functionalities of this program, scanning malicious hashes, generating a dataset of features and the classification of emails. Scanning the malicious files takes 20 minutes, and generating the dataset takes 45 minutes this needs to be improved to execute faster. However, the classification part of the program takes a few seconds.
Size: the size of the system will not exceed 1000 megabytes as a dataset of this size would take up more storage than required. This system only needs to demonstrate the functionality of the detection and classification.	Pass	The total size of the whole program is under 70 MB.
Re-useability: the implementation of the system needs to be broken down into separate modules so that they can be further used by other projects or systems.	Pass	The individual functionalities of the system are all executed on separate scripts so it can be re-engineered and used by other projects or systems.

Limitations

After the analysis of the implementation and the results many limitations have been uncovered and if they were avoided then it would have improved this tool. One of the biggest limitations that this tool has is the amount of time it takes to extract features from the datasets. This is caused by the language detector and translation API, if too many requests are sent at once then the access gets restricted,

and it takes a couple of minutes until it can be used again. So, to avoid this every time the API was used the program waits a couple of seconds before executing again. From analysing the weighting of the features of importance for most of the models the language features had very low weighting and had little effect on the classification. Removing them would reduce the number of features in the dataset but would rapidly decrease the time it takes extract all the features.

Another limitation with this tool is the VirusTotal API which as mentioned before is limited to 4 scans a minute. If this limitation did not exist, then it would be a lot easier to test the full capability of this functionality. Larger datasets of malware and non-malware hashes could be used to test if this API can detect all malicious hashes and to see how it handles when detecting non malicious files.

One of the final limitations of this system is the size of the dataset of phishing emails. There is an average of 8,000 emails from two different sources, if the system were put into production and used against live phishing attacks then it would be unlikely to detect phishing attacks with the same accuracy. To achieve these goal small and large datasets from various sources would need to be used to train and test the best algorithm, so the highest accuracy can be achieved that can protect against different types of phishing attacks. To also add to the limitation the datasets used are old and were not created recently so the classification models would only be effective against older phishing attacks.

Chapter 6: Future Work

Due to the limited time constraint for this project, there are a few aspects of the software that could have been improved. The first aspect would be spending more time on fine tuning the parameters for each Machine Learning model so that the best performance can be extracted. Comparing and evaluating the true capabilities of each system will help decide which classification algorithm is the most effective with detecting phishing emails. Another aspect regarding the Machine Learning that could have been further developed is experimenting and investigating other models to use and compare.

A second aspect that could have been further developed if more time were available is expanding and improving all the datasets for phishing emails, benign emails, and malware hashes. Within this improvement multiple datasets from different sources could be added to the total dataset of emails thus increasing the wider range and detection of different phishing email types. For this expansion, the systems scripts will need to be accommodated to receive multiple dataset inputs.

A third aspect would be creating a more user-friendly interface and extending the capabilities into an extension so it can be used on live production systems to detect incoming phishing attacks via email. The system within its current state is proof of a functional minimum viable product. However, if the system is further expanded to a state where it can be used against live systems then it should be developed to accommodate as a security tool against email phishing attacks. This should be the final goal to achieve as this is the intended use for developing a system with this capability.

Chapter 7: Conclusion

The purpose of this project was to develop a system that uses machine learning to detect phishing emails that contain malicious files. In order to achieve this goal multiple Machine Learning models were trained and tested. The dataset used to train the models were a collection of features extracted from many emails. All the results were compared and evaluated to distinguish which is the most effective model. The main findings regarding the classification of phishing and benign emails were that all the models achieved an accuracy more than 90% and the most efficient model was the Random Forest Classifier which scored just over 99% accuracy and the worst performing model was K Nearest Neighbor which scored 90%. The Neural Network algorithm showed to also have high accuracy too however this model required some fine tuning to the parameters to be effective. The high accuracy results were achieved because of the various quantities of features available that each model was able to utilise when selecting the features of importance during the classification training. The resource cost for using the Random Forest algorithm is very small although the memory usage was not measured the time it takes to train and test the model were. It took 1.1 seconds to classify 5444 emails in the training section and 0.06 seconds to classify 2682 emails in the testing section. If 10 times this number of emails were used in training and testing, then it would only take just over 11 seconds to train the model with 54,000 emails. In addition to the classification functionality the software also supported the functionality to scan hashes of a file and detect if they contain any malware. The results for this functionality were 100% and all the hashes of the malicious files that were scanned were detected to contain malware.

My recommendation for how companies can use this work is to take the functionality used within the scripts and re-engineer them into their own infrastructure security. The software can be used to scan all emails within their company to look for any phishing emails and it can be set up to monitor and classify all emails before any user receives them in their inbox. This solution does not fully address the whole issue regarding email phishing attacks, but it does increase the chances in detecting phishing attacks and reduce overall likelihood in being affected by them.

Chapter 8: Reflection on Learning

While I worked on this project I learned about many things and from researching and experiencing these new areas I have improved my knowledge and gained new skills. I tired experimenting with different ideas and methods some went well, and some went bad, and sometimes things changed overtime however every experience I had taught new and greater lessons.

One of the greatest topics I have personally learned from this project is understanding the design requirements with how to build and create a Machine Learning Classification model. Before I started this project, I did not have any knowledge on Machine Learning, so I had to self-teach myself its process and the different stages within this technology. After learning about what data, I needed to input into a model I delved deeper and learned about specific Machine Learning classifiers with how they operate and how they differ to each other. Some of the algorithms that I learned about were Linear Support Vector Classifier, Logistic Regression, Naive Bayes Bernoulli, Decision Tree, Random Forest, Neural Network, and K-Nearest Neighbor. Regarding my technical skills, I have learned from using the software scikit learn I have experienced how to fully implement functional machine learning classification models.

Whilst learning about phishing attack via emails I shocked by the sheer quantity of people that are affected by this every year it showed me that we need a better solution to protect ourselves against these attacks. Machine Learning algorithms require datasets so they can create models to be able to classify phishing and benign emails. The dataset I produced within this project was a list of extracted features from many emails. Upon researching different features that could be extracted I learned about the raw structure of an email and all the extra headings and information we do not see-through standard user interfaces.

During this project I tried experimenting with different ideas. One of them was using a data set of malicious files to develop the functionality to detect malware. However, when I installed a data set that contained live viruses, it infected the drive it was installed on. This caused me to take some drastic measures to rid myself of the files as they were not easy to delete, which is why I used the hashes of malicious files.

On a positive side one change that I made during this project that worked out as a benefit was using a different API to detect emotion. The first tool that I found worked well and could measure up to four emotions, but my supervisor introduced me to Robert Plutchik's wheel of emotions that can categorise emotions into one of eight classes. After learning about this emotional classifier, I decided to change the tool I was using instead to another API that can measure all emotions on Plutchik's

wheel. This change gave me a wider range of features with detecting emotion and as a result improved my classification models.

Another idea I was intending to implement into this software was a scanner which used VirusTotal to scan all the URLs within an email to detect for phishing links. However, developing a tool to gather all the URLs within every email would take a large portion of time to develop, test and de-bug so this idea was scrapped. Luckily, the classification models are very effective with detecting phishing emails however this could a feature to add for extra security.

Whilst developing this tool there was one slight mistake, I made at the start which I had to correct. It was gathering a dataset of Spam emails and not phishing emails, when I realised my mistake, I had to look for a new dataset of phishing emails. When I was coding the implementation method for extracting the different features, I was using the dataset of spam emails to test my program. Regarding the modifications made to the script to amend the issue, only the import method needed changing which was easily done.

Throughout the duration of this project, I have gained and learned many experiences from working on this task, my confidence, management and programming skills have all excelled to a new level. With my newfound knowledge and skills, I can utilise them to overcomes obstacles within my future career in infrastructure security engineering.

References

Last name, First initial. (Year published). Article Title. Journal, [online] Volume (Issue), pages. Available at: URL [Accessed Day Mo. Year].

[1] Jones, C. Mar 2021. 50 Phishing Stats You Should Know In 2021, Expert Insights, [Online] Available at: https://expertinsights.com/insights/50-phishing-stats-you-should-know/ [Accessed 18th April 2021]

[2] Sep 2020. New research shows significant increase in phishing attacks since the pandemic began straining corporate IT security teams, Security Magazine, [Online] Available at: https://www.securitymagazine.com/articles/93194-new-research-shows-significant-increase-in-phishing-attacks-since-the-pandemic-began-straining-corporate-it-security-teams [Accessed 18th April 2021]

[3] Oct 2020. 2020 Phishing statistics you need to know to protect your organization, Keep Net Labs, [Online Blog] Available at: https://www.keepnetlabs.com/phishing-statistics-you-need-to-know-to-protect-your-organization/# [Accessed 18th April 2021]

[4] 2020. Data Breach Investigations Report, Verizon, [Online] Available at: https://enterprise.verizon.com/resources/executivebriefs/2020-dbir-executive-brief.pdf [Accessed 18th April 2021]

[5] Pupale, R. Jun 2018. Support Vector Machines (SVM) — An Overview, Towards Data Science, [Online] Available at: https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989 [Accessed 19th April 2021]

[6] Pant, A. Jan 2019. Introduction to Logistic Regression, Towards Data Science, [Online] Available at: https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148 [Accessed 19th April 2021]

[7] Molnar, C. April 2021. Interpretable Machine Learning, Github, [Online] Available at: https://christophm.github.io/interpretable-ml-book/logistic.html [Accessed 19th April 2021]

[8] z_ai. Feb 2020. Logistic Regression Explained, Towards Data Science, [Online] Available at: https://towardsdatascience.com/logistic-regression-explained-9ee73cede081 [Accessed 19th April 2021]

[9] Soni, D. May 2018. Introduction to Naive Bayes Classification, Towards Data Science, [Online] Available at: https://towardsdatascience.com/introduction-to-naive-bayes-classification-4cffabb1ae54 [Accessed 19th April 2021]

[10] Gupta, P. Nov 2017. Naive Bayes in Machine Learning, Towards Data Science, [Online] Available at: https://towardsdatascience.com/naive-bayes-in-machine-learning-f49cc8f831b4 [Accessed 20th April 2021]

[11] What are Naïve Bayes classifiers?, PICO, [Online] Available at: https://www.pico.net/kb/what-are-naive-bayes-classifiers [Accessed 20th April 2021]

[12] Liberman, N. Jan 2017. Decision Trees and Random Forests, Towards Data Science, [Online] Available at: https://towardsdatascience.com/decision-trees-and-random-forests-df0c3123f991 [Accessed 20th April 2021]

[13] Ronaghan, S. May 2018. The Mathematics of Decision Trees, Random Forest and Feature Importance in Scikit-learn and Spark, [Online] Available at: https://towardsdatascience.com/the-mathematics-of-decision-trees-random-forest-and-feature-importance-in-scikit-learn-and-spark-f2861df67e3 [Accessed 20th April 2021]

[14] Simone, Dec 2020. The Grand Benchmark Table of Embedded Machine Learning, Eloquent Arduino, [Online] Available at: https://eloquentarduino.github.io/2020/10/decision-tree-random-forest-and-xgboost-on-arduino/ [Accessed 20th April 2021]

[15] Yiu, T. Jun 2019. Understanding Random Forest, Towards Data Science, [Online] Available at: https://towardsdatascience.com/understanding-random-forest-58381e0602d2 [Accessed 20th April 2021]

[16] Knocklein, O. Jun 2019. Classification Using Neural Networks, Towards Data Science, [Online] Available at: https://towardsdatascience.com/classification-using-neural-networks-b8e98f3a904f [Accessed 20th April 2021]

[17] Apr 2021. Artificial neural network, Wikipedia, [Online] Available at: https://en.wikipedia.org/wiki/Artificial_neural_network [Accessed 20th April 2021]

[18] Subramanian, D. Jun 2019. A Simple Introduction to K-Nearest Neighbors Algorithm, Towards Data Science, [Online] Available at: https://towardsdatascience.com/a-simple-introduction-to-k-nearest-neighbors-algorithm-b3519ed98e [Accessed 20th April 2021]

[19] Nelson, D. Aug 2020. What is a Confusion Matrix?, Unite, [Online] Available at: https://www.unite.ai/what-is-a-confusion-matrix/ [Accessed 20th April 2021]

[20] Waleed, A. Jun 2020. Particle Swarm Optimization-Based Feature Weighting for Improving Intelligent Phishing Website Detection, [Online] Available at: https://www.researchgate.net/figure/Confusion-matrix-for-the-problem-of-phishing-website-detection_tbl2_342323434 [Accessed 20th April 2021]

[21] Guest Author, Dec 2020. A simple guide to building a confusion matrix, Oracle, [Online] Available at: https://blogs.oracle.com/ai-and-datascience/post/a-simple-guide-to-building-a-confusion-matrix [Accessed 20th April 2021]

[22] Heller, M. Jul 2019. The best machine learning and deep learning libraries, InfoWorld, [Online] Available at: https://www.infoworld.com/article/3163525/the-best-machine-learning-and-deep-learning-libraries.html [Accessed 21st April 2021]

[23] Tatman, R. 2017. Fraudulent E-mail Corpus, Kaggle, [Online] Available at: https://www.kaggle.com/rtatman/fraudulent-email-corpus [Accessed 22nd April 2021]

[24] Radev, D. (2008), CLAIR collection of fraud email, ACL Data and Code Repository, ADCR2008T001, [Online] Available at: http://aclweb.org/aclwiki [Accessed 22nd April 2021]

[25] 2006, Spam Assassin, datasets, [Online] Available at: https://spamassassin.apache.org/old/publiccorpus/ [Accessed 22nd April 2021]

[26] Gangavarapu, T., Jaidhar, C.D. & Chanduka, B. Applicability of machine learning in spam and phishing email filtering: review and approaches, Artif Intell Rev 53, 5019–5081 (2020). [Online] Available at: https://link.springer.com/article/10.1007%2Fs10462-020-09814-9 [Accessed 22nd April 2021]

[27] Gbenga Dada, E., Stephen Bassi, J., Chiroma, H., Muhammad Abdulhamid, S., Olusola Adetunmbi, A., Emmanuel Ajibuwa, O. Machine learning for email spam filtering: review approaches and open research problems, Heliyon, [Online] Available at: https://doi.org/10.1016/j.heliyon.2019.e01802 [Accessed 22nd April 2021]

[28] Saravana, N. 2018. Malware Detection, Kaggle, [Online] Available at: https://www.kaggle.com/nsaravana/malware-detection/activity [Accessed 22nd April 2021]

[29] M. Rudd, Ethan., Harang, R., Saxe, J. Apr 2018. MEADE: Towards a Malicious Email Attachment Detection Engine, Sophos Group PLC, [Online] Available at: https://arxiv.org/pdf/1804.08162.pdf [Accessed 22nd April 2021]

[30] Sharma, N. May 2020. Spam Filtering Using Bag-of-Words, Heartbeat, [Online] Available at: https://heartbeat.fritz.ai/spam-filtering-using-bag-of-words-1c5484ff07f1 [Accessed 22nd April 2021]

[31] Azad, B. 2013. Identifying Phishing Attacks, Stanford, [Online] Available at: https://www.semanticscholar.org/paper/Identifying-Phishing-Attacks-Azad/776564e63fde46d9cfadc086f6e71ed8903ee4b4 [Accessed 22nd April 2021]

[32] Dewan, P., Kashyap, A., Kumaraguru, P. Jun 2014. Analyzing Social and Stylometric Features to Identify Spear phishing Emails, Indraprastha Institute of Information Technology, [Online] Available at: https://arxiv.org/pdf/1406.3692.pdf [Accessed 23rd April 2021]

[33] VirusTotal, [Online] Available at: https://www.virustotal.com/gui/ [Accessed 2nd May 2021]

[34] Mark M. Bailey. 2019, NRCLex, [Online] Available at: https://pypi.org/project/NRCLex/ [Accessed 2nd May 2021]

[35] Steven L. 2020. API Reference, Text Blob, [Online] Available at: https://textblob.readthedocs.io/en/dev/api_reference.html [Accessed 2nd May 2021]

[36] April 2021, Python quopri, Encode and decode MIME quoted-printable data, [Online] Available at: https://docs.python.org/3/library/quopri.html [Accessed 2nd May 2021]

[37] Apr 2021, Pandas Version 1.2.4, [Online] Available at: https://pandas.pydata.org/ [Accessed 2nd May 2021]

[38] Fabian, P., Gael, V., Alexandre, G., Vincent, M. 2010. scikit-learn 0.24.2, API Reference, [Online] Available at: https://scikit-learn.org/stable/modules/classes.html [Accessed 2nd May 2021]

[39] Rachael, T. 2017. Fraudulent E-mail Corpus, Kaggle, [Online] Available at: https://www.kaggle.com/rtatman/fraudulent-email-corpus [Accessed 3rd May 2021]

[40] Jan 2006. Spam Assassin, Public Corpus, [Online] Available at: https://spamassassin.apache.org/old/publiccorpus/20030228_hard_ham.tar.bz2 [Accessed 4th May 2021]

[41] Jan 2006. Spam Assassin, Public Corpus, [Online] Available at: https://spamassassin.apache.org/old/publiccorpus/20030228_easy_ham_2.tar.bz2 [Accessed 4th May 2021]

[42] Jan 2006. Spam Assassin, Public Corpus, [Online] Available at: https://spamassassin.apache.org/old/publiccorpus/20030228_easy_ham.tar.bz2 [Accessed 4th May 2021]

[43] Steven5538. Jun 2018. EMOTLIB, emotlib: Python emoji + emoticon Library, [Online] Available at: https://emotlib.readthedocs.io/en/latest/ [Accessed 5th May 2021]

[44] Saif M. Mohammad., Peter D. Turney. 2010. Emotions Evoked by Common Words and Phrases: Using Mechanical Turk to Create an Emotion Lexicon, Anthology ID: W10-0204, [Online] Available at: https://www.aclweb.org/anthology/W10-0204/ [Accessed 6th May 2021]

[45] shellgreenier. Feb 2021. Putting Some Emotion into Your Design – Plutchik's Wheel of Emotions, Interaction Design, [Online] Available at: https://www.interaction-design.org/literature/article/putting-some-emotion-into-your-design-plutchik-s-wheel-of-emotions [Accessed 6th May 2021]

[46] Parthvi, S. Jun 2020. Sentiment Analysis using TextBlob, Towards Data Science, [Online] Available at: https://towardsdatascience.com/my-absolute-go-to-for-sentiment-analysis-textblob-3ac3a11d524 [Accessed 6th May 2021]

[47] Michal, D. 2014. Langdetect, pypi, [Online] Available at: https://pypi.org/project/langdetect/ [Accessed 6th May 2021]

[48] Jenny, L. Nov 2020. Benchmarking Language Detection for NLP, Towards Data Science, [Online] Available at: https://towardsdatascience.com/benchmarking-language-detection-for-nlp-8250ea8b67c [Accessed 6th May 2021]

[49] Guest Contributor. 2014. Text Translation with Google Translate API in Python, Stack Abuse, [Online] Available at: https://stackabuse.com/text-translation-with-google-translate-api-in-python/ [Accessed 6th May 2021]

[50] Jason, B. Jul 2020. Train-Test Split for Evaluating Machine Learning Algorithms, Machine Learning Mastery, [Online] Available at: https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/ [Accessed 8th May 2021]

[51] Fabian, P., Gael, V., Alexandre, G., Vincent, M. 2010. scikit-learn 0.24.2, API Reference, [Online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html [Accessed 8th May 2021]

[52] Smith, M. 2007. 42 – The Answer to Life, The Answer to the Ultimate Question of Life, the Universe, and Everything is 42, Wikipedia, [Online] Available at: https://en.wikipedia.org/wiki/Phrases_from_The_Hitchhiker%27s_Guide_to_the_Galaxy#The_Answer_to_the_Ultimate_Question_of_Life,_the_Universe,_and_Everything_is_42 [Accessed 8th May 2021]

[53] SRA. Nov 2019. Scams, Solicitor Regulation Authority, [Online] Available at: https://www.sra.org.uk/consumers/problems/fraud-dishonesty/scams/ [Accessed 27th May 2021]