Detecting Phishing Emails with Ensemble Learning

Neo Michael Cass - nc721@bath.ac.uk

Master of Computer Science and Mathematics
The University of Bath
Academic Year: 3

Detecting Phishing Emails with Ensemble Learning

Submitted by: Neo Michael Cass

Abstract

Phishing is an increasingly dangerous and dynamic attack vector for fraudsters. This project aims to further the research into the automated detection of phishing emails using ensemble learning. Our ensemble algorithm of Random Forest, Naive Bayes and Support Vector Machine performed with a mean accuracy of 95.9% when tested over two datasets. Testing a range of ensemble and base classifiers over multiple datasets gave us a new insight into the generalisation of machine learning algorithms, which we have not seen elsewhere in existing literature.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank Amanda Leman, my mum; Tamsyn Hawken, my study skills tutor; Jade Dunn, my girlfriend; Marina De Vos, my supervisor and all my friends for supporting me throughout the project. This would not have been possible without them.

Chapter 1

Introduction

Phishing is a type of fraud which uses social engineering to trick victims into revealing personal information via malware or fake websites.

This can happen in two ways: a link leads you to a website masquerading as another, or it leads you to a download of malware without you knowing.

Phishing is a growing problem. The Anti-Phishing Working Group, who aim to unify the world's fight against phishing, reported 316,747 attacks last December - the highest in APWG's history (established 2003) [4].

We could argue this does not reflect actual financial loss, as the reported attacks need not be successful. However, Area 1 Security, a Californian cyber-security company, reported $354 million in direct losses in 2021 [5]. Phishing is predominantly done over email, you can see this in [4] as they only cite emails as an attack vector, although other cases of phishing do exist (e.g. via SMS [26]). So the importance of preventing the attacks is clear.

Estimating the impact of phishing is challenging when we consider that fraud is a hidden crime [21]. However, attempts have been made to estimate the financial loss due to identity theft from phishing. One estimate states $160.4 million is lost each year [25]. This does not even consider the emotional damages a user may incur by losing private data and access to their accounts.

Studies have shown that attempts to prevent the users from accessing dangerous websites has been ineffective [15]. Further, the authors in [15] found that 90% of users are tricked by phishing websites. We conjecture this percentage has increased significantly as phishers have become smarter and the former study was carried out in 2006.

Humans alone have been shown to be poor at detecting phishing scams. Studies testing participants' ability to detect scams shows us that users are better suited to detecting scams in conjunction with software than alone [20].

Spam filters have existed for many years, such as Apache SpamAssassin (as used in [17]), but fail to detect smarter phishing attacks which are becoming increasingly prevalent [4].

The need for a more generalised detection technique is clear. Machine Learning appears to be the popular choice among researchers. However, existing attempts vary in accuracy and there is no standardised way to evaluate these algorithms.

Machine Learning algorithms require re-tuning of their hyper-parameters. This is especially the case when the domain of phishing changes so rapidly. We need more robust algorithms which do not rely on tuning or particular datasets.

The answer to this is ensemble learning. This is when we combine multiple Machine Learning techniques together. Ensemble learning creates more robust and generalised algorithms.

This paper aims to further the research of ensemble algorithms for detecting phishing scams.

1.1 Overview

The remainder of this paper is organised as follows:

- Chapter 2 outlines existing literature and covers key terms and algorithms used throughout the paper;
- Chapter 3 states our hypotheses for the project;
- in Chapter 4 we design our experiments and give justifications for the choices we make;
- in Chapter 5 we demonstrate how the experiments were carried out;
- in Chapter 6 we present our experimental results;
- in Chapter 7 we analyse these results, giving reference to our experimental hypotheses;
- Chapter 8 concludes the paper, contextualising what we have achieved and stating what our future work will entail;
- Appendix A contains the raw output of our data from the experimental software and
- Appendix B contains a full code listing for the software.

Chapter 2

Literature and Technology Survey

2.1 Introduction

In this chapter we will define key terms and outline existing literature regarding detection of phishing, Machine Learning classification and ensemble techniques.

We gather information on various types of phishing to justify our research and cater our project to better detect scams.

We review literature on existing techniques for detecting phishing emails so we can build our own algorithms from this. Further, we research more general classification techniques to improve upon existing algorithms.

2.2 Phishing

Phishing is a type of fraud which uses social engineering to deceive victims into revealing personal information via malware or fake websites.

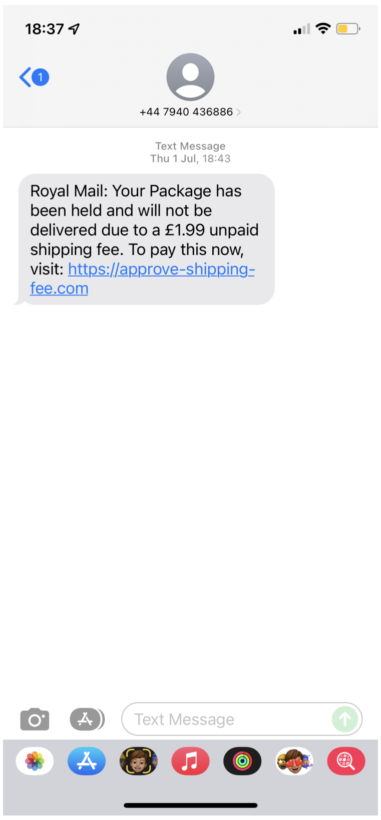
Phishing scams are distributed in many ways. These include, but are not limited to: email, SMS and social media. Phishing scams via SMS have recently gained popularity, being covered widely by news outlets [26]. You can see an example of this in Figure 2.1. However, we will be focusing on email as it is the larger issue. There are two main categories of phishing: a link leads the recipient to a website masquerading as another, or to a download of malware without them knowing. We will be detecting the scam from the contents of the email, so we can ignore the type.

2.2.1 Types of Phishing

Most, if not all, phishing leads you to a malicious website. The attacks therefore differ in the website and in the distribution of the website's URL.

Spear Phishing

Spear phishing is when a scammer targets a specific group of people with a phishing attack. These attacks can be more effective as the phisher can predict emails the victim will receive and tailor their scams to this.



Royal Mail: Your Package has been held and will not be delivered due to a £1.99 unpaid shipping fee. To pay this now, visit: https://approve-shipping-fee.com

Figure 2.1: A phishing attack via SMS we received.

A notable example of spear phishing would be Business Email Compromise (BEC). This is where the phisher gains the credentials of an employees business email. The phisher then uses the email to deceive other employees into giving them money or information. BEC attacks account for the loss of billions of dollars [4]. This attack vector is clearly popular among phishers. An APWG member, PhishLabs, reports 51.8% of malicious emails reported by corporate customers were attempting credential theft [4].

SMiShing

SMiShing is phishing via SMS text message. It has become well known with the recent widespread Royal Mail scam [26]. There is a distinct lack of research in this area, showing us how quickly the field of phishing can change.

2.2.2 Financial Loss

It is challenging to estimate financial loss due to phishing as fraud is considered a hidden crime [21]. However, there have been numerous studies estimating the loss by examining data from several sources. In [25] the authors estimate $160.4m is lost each year. This value is calculated from collecting phishing data over 8 weeks and extrapolating. This value is much lower than Gartner's estimate of $2bn. These statistics are from 2007, and the number of phishing attacks has greatly increased. In APWG's report from December of 2021, they reported 316,747 unique phishing websites [4], we can extrapolate this to 3,800,964 sites per year. Using the same values as in [25] [1], each site will stay up approximately 61 hours, luring 30 victims total. The paper states the average cost of identity theft is \$572 per victim. This would yield $3,800,964 \cdot 30 \cdot 572 = \$65,224,542,240 = \$65$bn lost per year.

2.3 Machine Learning

Machine Learning (or ML) is a branch of Artificial Intelligence (AI) which analyses existing data to form predictions.

We implement ML in many ways, this comes down to the choice of algorithm. In this paper we will focus on classification algorithms.

Classification algorithms (or classifiers) partition a dataset into multiple classes. In our case this is "phishing" or "non-phishing".

Partition here means the classes are disjoint. The classifier will identify an item as precisely one class. Further, a dataset, for our purpose, is a table of features, their values and a class label.

Classification algorithms require a feature vector. To get these features we must perform feature extraction. However, this is beyond the scope of the project. We will use datasets where feature extraction has already been performed.

2.3.1 Datasets and Creation

For our purposes, a dataset is a set of vectors $\mathbf{x}_i = (f_0, \dots, f_n) \in \mathbb{R}^n$, called feature vectors. Each f_j is called a feature.

Features

Features are attributes extracted from the object we are trying to classify. For example, the presence of "https" in a hyperlink could yield 0 or 1. Features can be discrete or continuous. E.g. $f_j \in \mathbb{R}$ or $f_j \in \{0, 1\}$. We choose the features to extract based on expert knowledge of how phishers operate.

Feature Selection and Extraction

Feature extraction is the process of producing a numerical value from an attribute of the object we want to classify. Feature selection is the process of how we choose what our features should be. This process is generally said to be one of the main challenges for detecting phishing scams with ML [28].

2.3.2 Supervised vs Unsupervised

ML can be divided into the following categories: supervised, semi-supervised and unsupervised [12].

[1]This is inaccurate as there is no guarantee the average lifespan of a phishing site, the number of victims or the average money lost has remained constant. However, this calculation is still a useful indication of how much phishing has grown, even with the large advancements in cybersecurity.

Supervised Learning

Supervised learning is where the data is labelled. For example, this could entail labelling each feature vector with a 1 for phishing and a 0 for non-phishing. This means our dataset is now a set of tuples $(\mathbf{x}_i, y_i)$ where $\mathbf{x}_i$ is a feature vector and y_i is the label. In our case, $y_i \in \{\textit{phishing}, \textit{non-phishing}\}$ which may be denoted as $\{1, -1\}$ or $\{1, 0\}$ respectively.

Unsupervised Learning

Unsupervised learning is the counterpart to supervised learning: the data is unlabelled. This requires us to use different algorithms.

In [1] the authors implement k-means clustering, an unsupervised learning technique, to detect phishing emails. They achieve an accuracy of 99.7%, which is higher than most supervised algorithms we encountered in our research.

Semi-Supervised Learning

Semi-supervised learning is where the algorithm learns from a dataset of labelled data $(\mathbf{x}_0, y_0), \dots, (\mathbf{x}_k, y_k)$ and unlabelled data $\mathbf{x}_{k+1}, \dots, \mathbf{x}_{k+n}$.

2.3.3 Types of Algorithm

In this subsection we introduce some important ML algorithms. We will be providing more detail to the algorithms we are going to use in our own solution.

Support Vector Machine

Support Vector Machine (or SVM) is a classification algorithm which separates an n-dimensional feature space with an $(n-1)$-dimensional hyper-plane. The hyperplane is of the form:

$$\mathbf{w}^\top \mathbf{x} - b = 0$$

where $\mathbf{w}$ is called the weight vector and b is the bias. $\mathbf{w}$ is also a vector normal to the plane.

The plane is chosen to be a maximum margin hyperplane that separates the two classes. This means the distance between the points closest to the hyperplane from each class are at the maximum possible distance apart.

The margin's size is $\frac{c}{||\mathbf{w}||}$, where c is some constant, so maximising this means we must minimize $||\mathbf{w}||$. The classical way of solving this problem is to reduce it to the dual representation:

$$\underset{\alpha}{\operatorname{argmax}} \sum_j \alpha_j - \frac{1}{2} \sum_{j,k} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j \cdot \mathbf{x}_k)$$

where $\alpha_j \geq 0$ and $\sum_j \alpha_j y_j = 0 \ \forall j$ and these can be computed with quadratic programming [29]. Then:

$$\mathbf{w} = \sum_j \alpha_j \mathbf{x}_j$$

and $\frac{b}{||\mathbf{w}||}$ is the the perpendicular distance to the origin from the line.

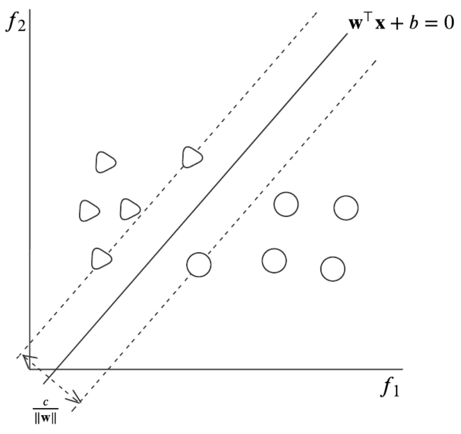

f_2
$\mathbf{w}^\top\mathbf{x} + b = 0$
f_1
$\frac{c}{\|\mathbf{w}\|}$

Figure 2.2: A 2-class separating hyperplane.

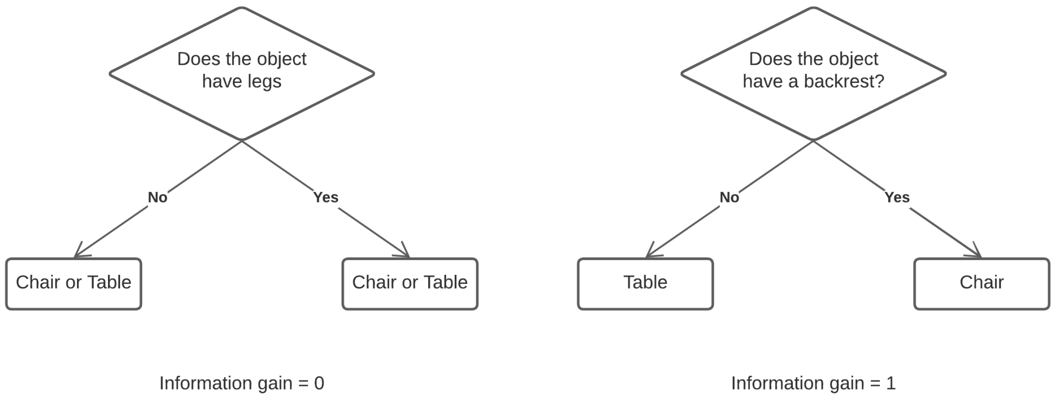


Figure 2.3: Example decision tree for chair or table example.

Data may not be separable by a linear hyperplane, in this case a kernel is used. A kernel is a function which maps the vectors into higher dimensional space where the data is linearly separable.

You can see in Figure 2.2 how SVM would work in 2-dimensional space. There are 2 classes, each with 2 features extracted (f_1 and f_2).

Decision Trees

The key concept behind decision trees is how we evaluate how well a tree splits a dataset. For example, if we are trying to classify chairs and tables: asking if the object has legs is a bad question as both tables and chairs have legs. Whereas, asking whether the object has a backrest (in most cases) will allow you to instantly differentiate between chairs and tables. See Figure 2.3.

We will cover two metrics for evaluating the effectiveness of decision trees: Gini and entropy.

Entropy at a node t is defined as

$$E(t) := -\sum_{i=1}^{N} P(i \mid t) \log_2 P(i \mid t)$$

where N is the number of classes and $P(i \mid t)$ is the probability of selecting a sample in class i randomly at node t. A node being a single decision in the tree.

To evaluate decision trees we use information gain: this is the difference in entropy before and after testing on a feature.

Reconsidering our chairs and tables example: if we have a dataset of 2 chairs and 2 tables:

Table 2.1: Chairs and Tables Dataset

n_Legs	hasBackrest	ChairOrTable
4	1	Chair
6	0	Table
5	1	Chair
4	0	Table

Clearly the choice of feature to test on in hasBackrest, it has an information gain of 1 because $-(0.5 \log_2 0.5 + 0.5 \log_2 0.5) - 0 = 1 - 0 = 1$

Gini impurity at a node t is defined as:

$$Gini(t) := 1 - \sum_{i=1}^{N} P(i \mid t)^2$$

where N and $P(i \mid t)$ are defined as before. It is a measure of how often a sample will be misclassified if it were classified according to the probability distribution over the classes [10].

Naive Bayes

For this subsection let $\mathbf{x} = (f_1, \dots, f_n)$ be a feature vector and let y be a class label. Naive Bayes (or NB) relies on Bayes Rule which is stated as:

$$P(y \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid y) \cdot P(y)}{P(\mathbf{x})}$$

The numerator of the fraction can be written as $P(\mathbf{x}, y)$ which denotes the probability that $\mathbf{x}$ and y are true. This can be rewritten as:

$$P(f_1, \dots, f_n, y) = P(f_1 \mid f_2, \dots, f_n) \cdots P(f_{n-1} \mid f_n, y) \cdot P(f_n \mid y) \cdot P(y) \tag{2.1}$$

The "naive" in the name for this algorithm comes from the assumption that features are independent. This allows us to write Equation 2.1 as:

$$P(f_i \mid f_{i+1}, \dots, f_n, y) = P(f_i \mid y).$$

This means:

$$Proba^y(\mathbf{x}) := P(y \mid f_1, \dots, f_n) = \frac{1}{P(\mathbf{x})} P(y) \prod_{i=1}^{n} P(f_i \mid y).$$

Then we classify the object dependent on the class with the maximum probability:

$$Predict(\mathbf{x}) := \underset{y}{\operatorname{argmax}}\, Proba^y(\mathbf{x}).$$

See [35] for further details.

Gaussian Naive Bayes (or GNB) is the Naive Bayes algorithm but the probability distribution of features is assumed to be Gaussian:

$$P(f_i \mid y) = \frac{1}{\sqrt{2\pi {\sigma_y}^2}} \exp\left(-\frac{(f_i - \mu_y)^2}{2{\sigma_y}^2}\right)$$

where μ_y and σ_y are the mean and standard deviation of the distribution, which are parameters estimated from the data [27].

Other Notable Algorithms

- *k* Nearest Neighbors (or KNN) - a popular supervised learning algorithm;
- C4.5 - a decision tree algorithm which uses information gain as its metric;
- Artificial Neural Networks (or ANN) - a supervised or unsupervised technique which is a neural network that somewhat mimics the human brain (see [29]) and
- Logistic Regression (or LR) - a supervised learning algorithm which is similar to SVM but relies on all the data points instead of just the "support vectors" (see [29]).

2.4 Ensemble Learning

Ensemble learning is when we combine the decisions of multiple ML algorithms together. This is favourable as different algorithms will perform better in different datasets.

Literature using ensemble learning to detect phishing scams is scarce. Many papers identify it would be beneficial to run multiple algorithms. This is because their run-times for the individual algorithms were low enough. However, very few papers describe the method of combining them or run the tests.

There are many papers on ensemble learning for classification in other contexts, which is what we will use to guide our research.

2.4.1 Generalisation

First, we give more rationale for why ensemble algorithms are needed. A generalised classifier is a classifier which can accurately predict classes of samples outside of the existing training and testing datasets [2]. The generalisation error is the amount of errors the classifier makes when predicting the classes of these unseen samples. We wish to minimize this error, but we cannot calculate it directly as this would require knowledge of ground truth information about classes [34]. This is infeasible as class labels cannot be known for unseen samples.

Combining algorithms in an ensemble creates more generalised algorithms. This is one of the main focuses of our project.

2.4.2 Combination Methods

Voting and Averaging

There are several techniques to combine the results of individual classifiers. These include averaging and voting. Each of these methods can be implemented in different ways. For example, we will be using simple soft voting.

Soft voting is used when each classifier's output is a tuple of probabilities, one for each class $(h_i^1, ..., h_i^l)$ where there are l classes. For example, $(P(Phishing), P(NonPhishing))$ or $(0.7, 0.3)$. Simple soft voting is when we treat each classifiers prediction equally. So the prediction for each class is:

$$H^j(\mathbf{x}) := \frac{1}{N}\sum_{i=1}^{N} h_i^j(\mathbf{x})$$

as seen in [34]. $h_i^j(x)$ is classifier j's prediction that $\mathbf{x}$ is in class i and N is the total number of classifiers.

However, it is very likely one classifier will be more accurate than another, so we may want to introduce unequal weights:

$$H^j(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} w_i h_i^j(\mathbf{x})$$

where w_i is the weight associated with classifier i. Simple soft voting is where we set each $w_i = 1 \ \forall i$.

A further complication would be to introduce class-classifier-specific weights w_i^j. This is discussed in [34] but we will consider this future work.

Hard voting is where we classify the sample based on the class with the modal largest probability over all the classifiers. For example, if 4 of the classifiers predict the email is 70% likely to be phishing, and 3 others predict 20%, the final verdict will be that the email is phishing (as $4 > 3$).

Boosting and Bagging

Boosting and Bagging are different from Voting and Averaging. Voting and Averaging train separate classifiers and then combines their results. Whereas, Boosting and Bagging use very

[2] Notably: phishing email classifiers need to be generalised, as the way phishing scams are executed changes regularly.

similar base classifiers.

Boosting is an ensemble of weak classifiers. Boosting trains one of its classifiers, then trains its next classifier to correct the mistakes of the first. The process is iterative and we can choose how many weak classifiers we have [34]. An example algorithm of Boosting is AdaBoost.

Bagging is where we sample data with a method called bootstrapping with replacement [8]. We then train separate learners on these samples and use a hard or soft vote on their predictions.

Random Forest

Random forest (or RF) is an ensemble algorithm which scores very highly in classification. The main idea is to use multiple decision trees to reduce overfitting (see Overfitting and Noise)[24].

Further, adding more trees does not cause overfitting but instead reduces the generalisation error [9].

Each decision tree is made up of randomly selected subsets of features and grown using the Classification and Regression Tree (CART) methodology [10]. This methodology searches the space of possible nodes to use at each level of a tree. The algorithm then chooses the node based on the Gini impurity index at that node [9].

Once the trees have been generated they perform a hard vote and this forms the final result.

RF solves the issue of dimensionaility as each tree is only trained with a subset of features. Further, the training of entirely independent trees lends itself to parallelisation. This has the opportunity to increase performance even further with the correct implementation.

2.4.3 Diversity

Creating effective ensemble configurations requires diversity in the base classifiers [34]. In [11] the authors state three ways in which this can be done

1. Vary the starting point for the hypothesis space (boosting);
2. Manipulate the training data (bagging) and
3. Use dissimilar base classifiers.

We will concentrate on (3).

2.5 Existing Systems To Detect Phishing Scams

2.5.1 Machine Learning

In [22] the authors achieve a very high TPR of 99.39% and an accuracy of 99.09%. The authors used: Random forest, SVM, ANNs, LR and NB. Their feature extraction across multiple plain-text datasets is what sets their results apart. However, their dataset is quite small (2141 phishing and 1918 legitimate) and this could lead to overfitting. They do use 10-fold cross-validation so this could counteract this. RF gained the best results, which is the only ensemble classifier they used.

In [30] the authors implement ANN, LR and SVM. They extract 10 features from 4000 emails (2000 phishing, 2000 legitimate). They achieve a maximum accuracy of 94.5%, which is quite poor for using ANN. Furthermore, they do not use cross-validation, so this accuracy could be even less in reality.

In [31] the authors implement several algorithms to detect phishing emails. Their dataset consisted of 11,000 emails with 30 features extracted. The authors mainly focus on varying hyper-parameters - this seems futile as they go on to say there is no optimal way to choose hyper-parameters which work in every dataset. They conclude that their best results come from ensemble classifiers such as RF and AdaBoost.

In [17] the authors present a new phishing detection tool, which utilises several different classification algorithms. Some examples include: NB, SVM, RF and AdaBoost (with C4.5 as the base classifer). The authors perform their own feature extraction and were able to identify 96% of the phishing emails. They only tested over one dataset and they only recorded two metrics (false positive/negative rate).

The authors of [1] used both supervised and unsupervised techniques to detect suspicious URLs on Twitter. They achieved an accuracy of 99.7% with *k*-means clustering, the highest of any algorithm we came across during our research. This could be due to the size of their dataset - 450,176 URLs, 23% being malicious.

2.5.2 Ensemble Learning

In [18] the authors use varying combinations of LR, NB, SVM and RF to classify decisions in the software development cycle to improve documentation. They performed feature extraction and built the dataset themselves - achieving an F1 score of 0.727 - which is very high considering the lack of data. They use precision, recall and F1 score as their scoring metrics. They did not consider any negative metrics (e.g. fall out). This data would have been useful to analyse the effects of voting classifiers on error rates.

In [7] the authors provide us with an implementation of ensemble learning in phishing. They use 3 different combinations of classifiers: ANN+RF, KNN+RF and C4.5+RF. They combine the algorithms in a simple soft vote. However, they did not take advantage of a key testing opportunity: using multiple data-sets. Furthermore, no consideration was given to the diversity of the algorithms.

2.5.3 Spam Filters

In [17] the authors use Apache SpamAssassin to pre-process emails. Apache SpamAssasin is a very popular open source spam filter. This has the same advantages as ensemble learning. It also means their phishing algorithm does not (technically) produce any erroneous results, as the emails would have already been marked as spam. The authors gained very high results with each algorithm not performing differently to one another within any statistical significance. This is surprising, as many other papers have much larger differences between algorithms. We believe the differentiating factor is the quality of features extracted and the age of the paper. Being released in May 2007, phishing was less prevalent. APWG reported 37,438 phishing websites in May 2007 [3], compared to 316,747 attacks in December 2021, which was the highest in their reporting history [4]. As you can see this is an increase by a factor of 10, phishers have become smarter in evading detection, so it is harder to extract useful features.

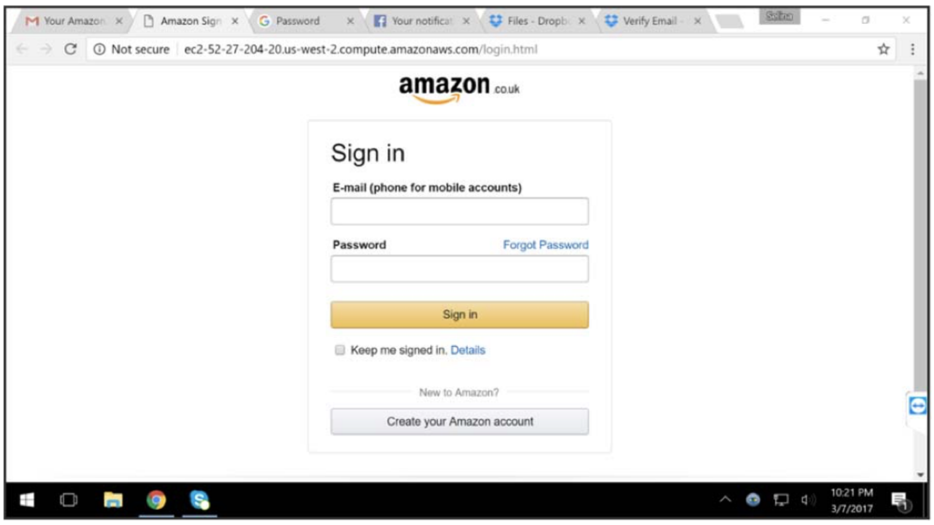

Not secure
ec2-52-27-204-20.us-west-2.compute.amazonaws.com/login.html
amazon.co.uk
Sign in
E-mail (phone for mobile accounts)
Password
Forgot Password
Sign in
Keep me signed in. Details
New to Amazon?
Create your Amazon account
10:21 PM
3/7/2017

Figure 2.4: An Amazon phishing scam from [20].

2.5.4 Humans as a Security Sensor

Humans as a Security Sensor is where we employ humans to detect security issues. In [20] they compare humans detecting phishing scams with various other existing systems, such as, the email provider, browsers and anti-virus. For certain scams, the human sensors detected scams which very few of the automated systems did.

For example, 6 out of 7 human sensors detected the scam in Figure 2.4 whereas only one of the automated sensors detected it. The human sensors noticed that there was no "https" present in the URL and that the domain was AWS not Amazon. The authors state that HaaSS should be used in conjunction with existing security software. In this sense, a phishing filter may flag an email, as it is the user's choice whether to trust it or not. This means false positives are less detrimental.

Chapter 3

Experimental Hypotheses

In this chapter we state our experimental hypotheses for different areas of interest.

3.1 Ensemble Learning

h_1 We conjecture that the ensemble classifiers perform better than the individual classifiers over all the datasets. This will be measured with the metrics you can see in Table 4.6.

h_2 We hypothesise, that no ensemble including RF will perform better than random forest on its own. This is because RF is already generalised and has the benefits of an ensemble classifier, so it is likely it cannot be that diverse from other classifiers.

h_3 We will also evaluate which ensemble performs the best overall. We conjecture the ensemble with the most base classifiers will perform the best.

3.2 Generalisation

h_4 Even with un-tuned parameters, the ensemble classifiers perform better over the two datasets than the individual classifiers.

3.3 Wider Implications

h_5 If the ensemble classifiers perform well across both datasets, then we need to research why phishing remains such a prevalent issue given the advances in generalised ML techniques.

Chapter 4

Design of Experiments

In this chapter we describe the design of the experiments to be undertaken and give motivation for the choices of experiments.

4.1 Datasets

In this section we introduce the datasets we will run our tests on, discussing why we have chosen to run tests over multiple datasets.

Table 4.1: Datasets Used

Name	Number of Features	Size	% Phishing	Dates Extracted	Source
Dataset 1	48	10,000	50	January - May 2015 and May - June 2017	[32]
Dataset 2	30	11,055	55.7	March 2015	[16]

Table 4.1 lists both datasets we will use to run our tests. The majority of papers only test over one dataset. However, we hypothesise this does not evaluate the algorithms well. Phishing attacks change regularly, with the mean average lifespan of a phishing website being 61.69 hours [25], so the algorithms must not rely on certain features to be more predictive. An algorithm performing well over both datasets would indicate the algorithm is well generalised.

4.1.1 Benchmarking

Currently, there appears to be no unified method of evaluating ML classifiers. The evaluation technique largely depends on the domain. For example, in the domain of phishing, sensitivity (see Metrics) is very important as flagging legitimate emails as dangerous is not as detrimental as flagging dangerous emails as legitimate.

There have been attempts to create benchmarking datasets for phishing (e.g. [19]). However, the work is unfortunately unfinished. They state that there are many works where the performance of the algorithms is dataset dependent. This is justification for why we must test our algorithms over multiple datasets to ensure this is not the case in our project.

4.2 Example Features

In this section, we give examples of features in each dataset. A full list of features can be found in [13] and [16].

Table 4.2: Example Features from Dataset 1

Name	Type	Rule from [13]
PctExtHyperlinks	Continuous	Counts the percentage of external hyperlinks in webpage HTML source code
NumDots	Discrete	Counts the number of dots in webpage URL
PctExtNullSelfRedirectHyperlinksRT	Categorical	Counts the percentage of hyperlinks in HTML source code that uses different domain names, starts with "#", or using "JavaScript ::void(0)". Apply rules and thresholds to generate value.
FrequentDomainNameMismatch	Binary	Checks if the most frequent domain name in HTML source code does not match the webpage URL domain name.

In Table 4.2 we highlight 4 features from Dataset 1, one of each type in the dataset is included.

In Table 4.3 we highlight 3 features from Dataset 2, one of each type in the dataset is included.

4.3 Combinations

In this section we highlight what combinations of algorithms we will test and which algorithms will be included in the ensembles.

We will be using combinations of 3 different classifiers in a simple soft vote as described in Combination Methods. The list combinations can be seen in Table 4.4.

The 3 algorithms we will be using are Support Vector Machine, Naive Bayes and Random Forest. Every possible combination has been chosen to gather a complete set of results. We also test the individual classifiers for comparison.

4.3.1 Reasons for Selection

Support Vector Machine

Support Vector Machine is a very common algorithm used in the majority of ML papers. We selected it as it generalises very well [29].

Table 4.3: Example Features from Dataset 2

Name	Type	Rule from [16]
having_IP_Address	Binary	If the domain part has an IP address then the webpage is phishing.
Google_Index	Binary	If the webpage is indexed by Google then it is legitimate.
age_of_domain	Categorical	If the webpage uses https, the issuer is trusted and the age of certificate is older than 1 year then the webpage is legitimate. If the webpage is using https and the issuer is not trusted then the webpage is suspicious. Otherwise, the webpage is considered phishing.

Table 4.4: Table of Ensembles

Name	Combination
ECLF1	SVM+RF+GNB
ECLF2	SVM+RF
ECLF3	SVM+GNB
ECLF4	GNB+RF

Naive Bayes

Naive Bayes is also very commonly used in phishing classification papers [18] [22]. The performance of the algorithm will also highlight whether features are independent of one another, as the algorithm assumes probabilistic independence between features. We will be using Gaussian Naive Bayes, which assumes the likelihood of features is Gaussian (see Naive Bayes).

In [6] the authors measure how sensitive SVM and NB (and others) are to noise (see Overfitting and Noise). They find that NB is the most resistant to noise. Its accuracy only increases a negligible amount as the noise is reduced. This is a further reason why we chose to use GNB.

Random Forest

Random Forest is also a very common algorithm in phishing literature. It is also a well-known ensemble algorithm, so it generalises well. This could mean it will perform badly in ensembles, as it cannot be that diverse from the other algorithms.

The probability predictions of each ensemble is defined by the probability function:

$$Proba^j : \mathbb{R}^k \to \mathbb{R}_{\geq 0}$$

$$Proba^j(\mathbf{x}) := \frac{1}{N}\sum_{i=1}^{N} proba_i^j(\mathbf{x})$$

where N is the number of classifiers in the ensemble, $\mathbf{x}$ is the feature vector for the email, k is the number of features and $proba_i^j$ is the i-th individual classifier's probability function that the email is in class j:

$$proba_i^j : \mathbb{R}^k \to \mathbb{R}_{\geq 0}$$

$N \leq 3$ as we are not using ensembles with more than 3 classifiers.

The final prediction is made by:

$$Predict : \mathbb{R}^{48} \to \{0, 1\} \text{ or}$$
$$Predict : \mathbb{R}^{30} \to \{-1, 1\}$$

for Dataset 1 and 2 respectively.

$$Predict(\mathbf{x}) := \underset{j}{\operatorname{argmax}}\, Proba^j(\mathbf{x})$$

This is the same as stating the class with the largest probability is the final prediction. Further, 0 and -1 denote legitimate and 1 denotes phishing.

We chose to combine the algorithms in a simple soft vote as adjusting the weights of the algorithms over two datasets would reduce generalisation. Further, a simple soft vote is still favourable to a hard vote. This is because, the more predictive an algorithm is will affect the classification. For example, if RF gives the output of 99% that the email is phishing, this sways the overall average more than if RF outputted 1 (e.g. "phishing") in a hard vote. So algorithms which can differentiate between classes more decisively sway the result more, even if the voting weights are all 1.

4.4 Metrics

In this section we cover all the metrics we will judge our algorithms by.

Table 4.5: Classification of Results

Name	Identifier	Description
True Positive	TP	The number of emails which are correctly classified as phishing.
True Negative	TN	The number of emails which are correctly classified as legitimate.
False Positive	FP	The number of emails which are **in**correctly classified as phishing.
False Negative	FN	The number of emails which are **in**correctly identified as legitimate.

Table 4.6: Table of Metrics

Metric	Equation
Accuracy	$\frac{TP+TN}{TP+FP+FN+TN}$
F1 Score	$2 \cdot \frac{precision \cdot sensitivity}{precision+sensitivity}$
Precision	$\frac{TP}{TP+FP}$
Sensitivity	$\frac{TP}{TP+FN}$
Fall out	$\frac{FP}{FP+TN}$
Specificity	$\frac{TN}{TN+FP}$
Area under ROC curve	The receiver operating characteristic (ROC) curve is a plot of sensitivity against fall out. AUC is the area under this curve. We will refer to this as AUROC.
Negative predictive value	$\frac{TN}{TN+FN}$
False negative rate	$\frac{FN}{TP+FN}$
False discovery rate	$\frac{FP}{TP+FP}$
Predict time	The time it takes the classifier to predict whether the email is phishing or not in seconds.
Fit time	The time it takes to train the classifier in seconds.

There are several metrics used to compare ML algorithms. From [7] [23] [18] [22] [30] [31] we have observed multiple different metrics. As these metrics are trivial to collect, we will collect all of them to ensure we do not miss any potentially interesting results. The full list of metrics can be found in Table 4.6.

These metrics are derived from how we classify our algorithm's predictions. In Table 4.5 we show how we will do this.

4.4.1 Descriptions of Metrics

Accuracy

The proportion of the time the algorithm correctly classifies the email.

F1 Score

The weighted average of precision and sensitivity.

Precision

The proportion of the phishing predictions the algorithm made that are correct.

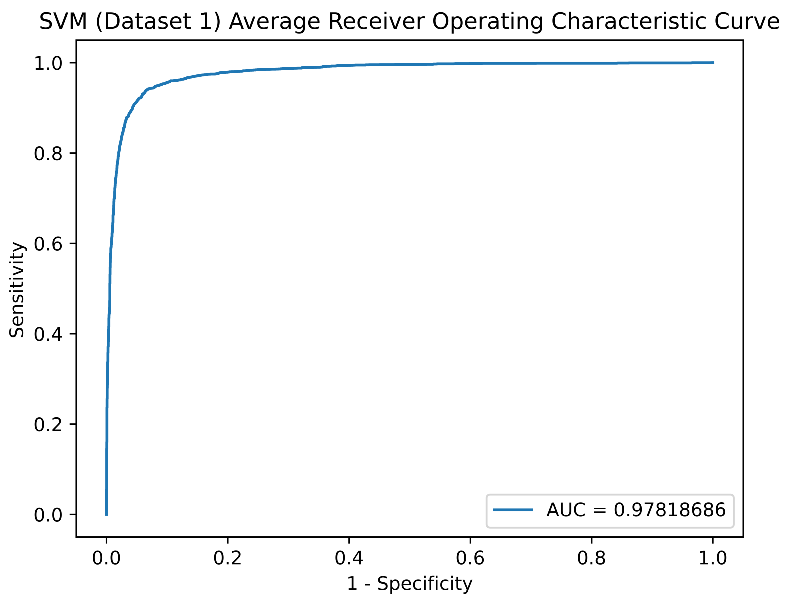


Figure 4.1: An example ROC curve.

Sensitivity

The proportion of phishing emails the algorithm correctly classified.

Fall Out

The proportion of legitimate emails incorrectly identified as phishing.

Specificity

The proportion of legitimate emails correctly identified as legitimate.

ROC Curve

An ROC curve is a plot of sensitivity against fall out $(1 - \text{Specificity})$ at different thresholds. Theoretically, we plot over every possible threshold in $[0, 1] \subset \mathbb{R}$. However, we instead pick a subset of this, depending on the implementation. The area under the ROC curve (AUROC) is a measure of how well the algorithm can differentiate between classes. An AUC of 1 would imply the algorithm has $\mathit{Proba}^j(\mathbf{x}) \in \{0, 1\}\ \forall \mathbf{x}, j$. See Figure 4.1 for an example ROC curve for SVM.

Negative Predictive Value

The proportion of the legitimate predictions the algorithm made that are correct.

False Negative Rate

The proportion of phishing emails the algorithm incorrectly classified.

False Discovery Rate

The proportion of phishing predictions the algorithm made that are incorrect.

Predict Time

The amount of time (in seconds) the algorithm takes to make a prediction.

Fit Time

The amount of time (in seconds) it takes to train the algorithm.

4.5 Parameters

We will not run any parametric tests to tune our algorithms. This is beyond the scope of the project - we are investigating the effects of combining algorithms.

4.6 Implementation

We will be using Sci-Kit Learn (or SKLearn) [27], a Python library, to implement the classifiers. This is because it is a well documented library which is used in many papers [1] and by many businesses [2]

To read the comma separated value (CSV) files we will use Pandas, which is a Python data analysis library.

4.7 Scoring

4.7.1 Overfitting and Noise

Scoring a classifier accurately is challenging. This is because of overfitting. This is when a classifier is fit too closely to the training data and learns the "noise" of the dataset.

Noise can be classified as either attribute noise or class noise [36]. Class noise is where the data is mislabelled and attribute noise is when we make errors collecting values for features. Class noise can be due to multiple samples having identical feature values but different class labels [36].

There are many techniques for reducing over-fitting. One technique is prepossessing the data itself to eliminate the noise. The noise can be detected with ensemble techniques as seen in [33]. However, we will not do this and instead reduce overfitting by using ensemble algorithms instead of single algorithms.

In phishing detection, creating generalised algorithms is particularly important as phishing attacks are mostly zero-day.

[1]Notably: [18] which is a paper on ensemble classification (see Ensemble Learning)
[2]Notably: JPMorgan and Spotify (see SKLearn's website).

4.7.2 Cross Validation

To combat overfitting, we will score our classifiers using stratified k-fold cross validation. k-fold cross validation is described as follows:

1. The dataset is partitioned into k equally sized parts (folds)
2. The classifier we are scoring is trained on $k - 1$ of the folds
3. The trained classifier is tested on the remaining fold
4. Steps 1-3 are repeated until each of the k folds have been the fold that is tested on

Stratified k-fold cross validation is where we partition the dataset into k folds, but we preserve the ratio of the classes in the folds. For example, if the dataset is 25% phishing emails, then each fold should contain 25% phishing emails. During the cross validation, each email is predicted exactly once. This will allow us to calculate TP, TN, FP, FN and all the other relevant metrics.

At each fold, the predict and fit time will be calculated, then the mean average will be calculated to get a more accurate set of timings. Averaging the results will reduce the effects of scheduling by our test system's operating system. Timing is recorded as the classifier will need to be retrained regularly to maintain accuracy over the changing phishing techniques.

4.8 Testing Environment

Table 4.7: Testing Configuration

Operating System	macOS Monterey Version 12.3.1
Model	MacBook Pro (13-inch, 2020, Four Thunderbolt 3 ports)
Processor	2 GHz Quad-Core Intel Core i5
Memory	16 GB 3733 MHz LPDDR4X
Graphics Processing Unit	Intel Iris Plus Graphics 1536 MB
Python Version	3.9.2
SKLearn Version	1.0.2

This section outlines the machine the tests will be run on. In Table 4.7 we list the specifications of the test environment. This includes versions of the libraries and operating system we will use and the hardware specifications of the machine.

Chapter 5

Experimental Implementation

In this chapter we describe how the experiments were undertaken, justify specific choices we made and critique the methodology.

5.1 Individual Classifiers

Unless otherwise stated, tests were ran with default parameters. This is because choosing generalised, optimised parameters is a non-trivial task that involves several estimations and is beyond the scope of the paper. The performance of some algorithms may suffer as a result, but the combinations of algorithms should still yield interesting results regarding voting classifiers. Furthermore, `GridSearchCV`, SKLearn's parameter searching algorithm, is not designed to consider multiple datasets.

5.1.1 Support Vector Machine

Table 5.1: Parameters for Support Vector Machine

Main Parameters	Description	Value Used
`kernel`	The kernel to be used.	`linear`
`n_jobs`	The number processes to run in parallel when running `.fit` and `.predict`.	−1 - this means the number of processes is set to the number of cores on the machine.
`probability`	Determines whether `.predict_proba` can be used or not.	`True`

SVM was implemented using `SVC()`, which is SKLearn's implementation of SVM. As stated in Section Parameters we will run tests using predominantly default parameters. However, for the ensemble algorithms we need SVM to output probabilities, so we must set `probability=True` which forces `SVC()` to have a `.predict_proba()` function which essentially implements the function *proba* (see Combinations). The default kernel in SKLearn in RBF, but we will use `kernel="linear"` as this is closer to default in the mathematical context.

5.1.2 Random Forest

Table 5.2: Parameters for Random Forest

Main Parameters	Description	Value Used
`n_estimators`	The number of trees in the forest.	`default = 100`
`n_jobs`	The number processes to run in parallel when running `.fit` and `.predict`.	−1 - this means the number of processes is set to the number of cores on the machine.
`criterion`	The criteria by which the algorithm judges how well a decision tree splits the data.	`default = "gini"`

RF was implemented using `RandomForestClassifier()`, which is SKLearn's implementation of RF. Tests were ran using the default parameters which can be seen in Table 5.2.

5.1.3 Gaussian Naive Bayes

Table 5.3: Parameters for Gaussian Naive Bayes

Main Parameters	Description	Value Used
`priors`	Specified probabilities of each class (phishing/not phishing).	Adjusted according to the data by SKLearn.

We implement GNB using `GaussianNB()`, which is SKLearn's implementation of GNB. Tests were ran using the default parameters which can be seen in Table 5.3.

5.2 Ensemble Classifiers

We used SKLearn's `VotingClassifier` to implement the simple soft voting. `VotingClassifier` can perform a soft or hard vote depending on the parameters, we used `voting="soft"` with `weights` left unspecified. The default value of `weights` is `[1,...,1]` (e.g. this causes the vote to be simple). `VotingClassifier` also requires a list of tuples containing a classifier and as name for the classifier. For example, for ECLF1, we used:

```
VotingClassifier(estimators=[('svm', SVM), ('rf', RF), ('gnb',
    GNB)], voting = "soft", n_jobs = −1)
```

where SVM, RF and GNB are declared as in the previous section.

5.3 Scoring

We implement stratified 5-fold cross validation using SKLearn's `StratifiedKFold`. 5 and 10 fold cross validation is very common in ML literature. We chose to have 5 folds because

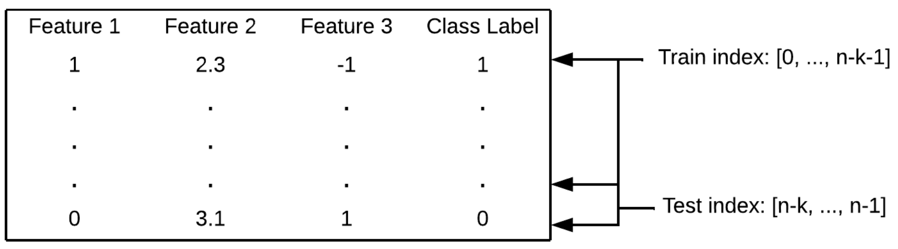


Figure 5.1: Diagram of how the indices for KFold are generated.

of hardware limitations. We ran `.split(X, y)`, which splits the dataset into `n_splits`, preserving the proportion of classes. X being the feature vector, y being the class label and `n_splits` being a parameter we set to 5. The `.split` function returns two arrays: train and test. These hold the indices for each train and test fold. An example of these can be seen in Figure 5.1.

At each iteration of cross validation, we did:

```
X_train, X_test = X.iloc[train_index], X.iloc[test_index]
y_train, y_test = y.iloc[train_index], y.iloc[test_index]

fitTime1 = timer()
eclf.fit(X_train, y_train) #Trains the classifier
fitTime2=timer()
fitTime = fitTime2-fitTime1
fitTimes.append(fitTime)
scoreTime1=timer()
y_pred = eclf.predict(X_test)
scoreTime2=timer()
scoreTime = scoreTime2-scoreTime1
scoreTimes.append(scoreTime)
y_prob = eclf.predict_proba(X_test)
y_preds+=y_pred.tolist()
y_tests+=y_test.tolist()
y_probs+=y_prob[:,1].tolist() #Gets the positive class
    predictions
```

This code records the predictions, the probability predictions and average permutation importance with its standard deviation. The code also records the fit and score time. `y_preds` stores the class predictions and `y_probs` stores the probabilities predicted for the phishing class (this is stored to plot the ROC curve). `y_tests` stores the test labels so we can calculate the confusion matrix against `y_preds`.

A confusion matrix is a matrix which is defined as:

$$\begin{pmatrix} TN & FP \\ FN & TP \end{pmatrix}$$

In SKLearn we use the following code to calculate the confusion matrix:

```
cm = confusion_matrix(y_tests, y_preds)
```

There were two options for calculating the confusion matrix:

1. calculate it at each iteration and calculate the mean average after the cross validation has finished;
2. Calculate it by comparing all the predictions with the true values after the cross validation has predicted each email once.

However, these values are the same, we choose to do (2) to reduce the number of calculations.

Scores were saved to a CSV file to reduce human error when recording results. We repeated each experiment once for each dataset for each algorithm.

Chapter 6

Experimental Results

In this chapter we present the results of the preceding chapter's experiments. A full output of results can be found in Raw Results Output, we will concentrate on summary metrics. These will include, F1 score, accuracy, sensitivity, AUROC, fit and score time. These indicate a classifiers overall performance more than the other metrics.

Sensitivity is particularly important in phishing detection. This is because the percentage of phishing emails correctly detected is directly related to preventing fraud. Misclassifying a legitimate email is undesirable behaviour but it is more important to flag dangerous emails correctly.

To present the results we first transposed the CSV file (as the results were too large for publication as they were). We then imported the CSV files into excel and used "=ROUND(Score,3 - (1 + INT(LOG10(ABS(Score)))))" to round to 3 significant figures. We then calculated the mean average of the results from the two datasets using "=AVERAGE(Score in Dataset 1, Score in Dataset 2)".

We chose to process our data to 3 significant figures as this is a standard in scientific studies.

6.1 SVM Results

Table 6.1: Average Scoring Results Over Both Datasets for SVM

Accuracy	F1 Score	Sensitivity	Fall out	AUROC	Score Time	Fit Time	Error Rate
0.930	0.935	0.946	0.0883	0.977	0.0739	19.3	0.0701

SVM achieved very high results. SVM's main downfall was fit time, although in a real life implementation this is not a large issue as we can allocate more powerful machines to training. SVM is very computationally complex, as it is computing linear separators in up to 48-dimensional space in this case.

Table 6.2: Average Scoring Results Over Both Datasets for RF

Accuracy	F1 Score	Sensitivity	Fall out	AUROC	Score Time	Fit Time	Error Rate
0.973	0.974	0.976	0.0325	0.995	0.0325	0.490	0.0276

6.2 RF Results

RF performed the best out of all the classifiers. We expect this is because it is a very well researched ensemble algorithm. The algorithm does not rely on specific features of either dataset, as its scores are high on both.

6.3 GNB Results

Table 6.3: Average Scoring Results Over Both Datasets for GNB

Accuracy	F1 Score	Sensitivity	Fall out	AUROC	Score Time	Fit Time	Error Rate
0.725	0.639	0.518	0.0294	0.947	0.00215	0.00672	0.276

Gaussian Naive Bayes performed the worst on all metrics, which is unexpected. Further, any ensemble classifier including GNB also performed very badly. GNB also had a very low sensitivity, only identifying half of the phishing emails. This is as bad as a random guess.

The score and fit times were very low. However this is expected since GNB is not very computationally complex. AUROC was high, however this does not matter when the accuracy is very low. This shows GNB was very decisive about decisions but its decisions were incorrect.

Fall out was low, meaning very few legitimate emails were flagged as phishing. This is favourable for real software implementation as user's would not want to miss legitimate emails because they were flagged.

To conclude, GNB performed poorly across all metrics. Reasons for this will be discussed in the next chapter.

6.4 ECLF1 Results

Table 6.4: Average Scoring Results Over Both Datasets for ECLF1

Accuracy	F1 Score	Sensitivity	Fall out	AUROC	Score Time	Fit Time	Error Rate
0.948	0.950	0.927	0.0265	0.991	0.125	19.3	0.0518

ECLF1 is the combination of all three base classifiers. It outperformed ECLF3 and 4 across all metrics, but not 2. Further, it did not outperform RF, which is as expected. ECLF1's AUROC score is higher than SVM and GNB. This is interesting, this experiment leads us to believe the more voters in an ensemble the more decisive the classifier is. However, RF is already an ensemble with a high AUROC score, so it is expected its decisiveness as part of an ensemble would be negatively affected.

6.5 ECLF2 Results

Table 6.5: Average Scoring Results Over Both Datasets for ECLF2

Accuracy	F1 Score	Sensitivity	Fall out	AUROC	Score Time	Fit Time	Error Rate
0.961	0.963	0.969	0.0491	0.994	0.125	19.8	0.0392

ECLF2 was the best performing voting classifier. This is unexpected as it has only two voters. However, this can be explained by GNB's high AUROC, which would have swayed ECLF1's decisions significantly.

6.6 ECLF3 Results

Table 6.6: Average Scoring Results Over Both Datasets for ECLF3

Accuracy	F1 Score	Sensitivity	Fall out	AUROC	Score Time	Fit Time	Error Rate
0.787	0.752	0.629	0.0279	0.973	0.0982	20.2	0.214

ECLF3 performed the worst out of the voting classifiers. It is the combination of the two weakest base classifiers so this is the expected outcome. Its sensitivity is 0.629 which is very poor. The fit time was also very high, at 20.2 seconds, which is lower than ECLF1. This does not align with our predictions, as the more classifiers you must train the more time it should take to train the ensemble.

6.7 ECLF4 Results

Table 6.7: Average Scoring Results Over Both Datasets for ECLF4

Accuracy	F1 Score	Sensitivity	Fall out	AUROC	Score Time	Fit Time	Error Rate
0.852	0.841	0.75	0.0268	0.989	0.0471	0.843	0.148

ECLF4 shows us that GNB reduces the performance of any classifier it is in an ensemble with. ECLF4 is the combination of GNB and RF - RF is the strongest classifier we tested but when it is paired with GNB the resulting ensemble is the second worst voting classifier.

Chapter 7

Analysis of Results

7.1 SVM

SVM was the second best base classifier. Although sources state SVM generalises well [29], this is after choosing a kernel to ensure the data is linearly separable. However, when we are considering two different datasets (with two different feature spaces) this is a very challenging task. We would argue that SVM only generalises well when there is a clear choice of kernel and this may not be possible with a dynamic feature set.

7.2 RF

RF performed the best as expected. Further, any ensemble RF was a part of was worse than RF itself. This is due to RF being a highly generalised algorithm already, meaning it cannot be diverse enough from other algorithms to perform well in an ensemble.

RF is an ensemble of decision trees. We believe its failures can be attributed to the same failures of standard decision trees. Decision trees fail if:

1. there is noise in the data;
2. the domain is non deterministic or
3. we cannot observe a feature which differentiates the classes [29].

We conjecture that the failure of RF was due to all three in part. The datasets were small enough for there to be noise in them (1), there may be websites which are harmful but not considered phishing (2) and phishing emails may not be differentiable from legitimate emails due to a phishers effectiveness (3).

In [9] a theorem is proven, which states increasing the number of trees in the random forest reduces the generalisation error and does increase the risk of overfitting. This is clearly not true in general for ensembles, we have seen that adding more algorithms into an ensemble can reduce the performance significantly. Further, this means the performance of RF could have also been improved by increasing `n_estimators`. This would not have a large impact on training times as the train times were very low at `n_estimators = 100`.

7.3 GNB

Clearly, GNB performed the worst. Each voting classifier GNB was included in performed considerably worse. This is likely because GNB relies on two conditions:

1. independence of features;
2. Gaussian (normal) distribution of features.

This leads us to conclude that the features are not entirely independent.

The sensitivity of ensembles with GNB as a base classifier was significantly lower than the other algorithms. So GNB correctly identifies legitimate emails more often than it correctly identifies phishing emails. This is undesirable behaviour. GNB's negative predictive values were also low, this means GNB made more legitimate classifications than phishing. Clearly the class probabilities were incorrectly skewed and the value of `priors` needs to be tuned.

7.4 ECLF1

ECLF1 performed worse than ECLF2, this is again due to GNB swaying the result. GNB also had an AUROC of 0.947, which means its incorrect results significantly swayed results in the ensembles it was a part of. However, GNB's bad performance does not have to be negative. In [34] the author states that ensembles made up of "weak learners", that are as bad as a random guess, can still perform as well as "strong learners". This leads us to conclude we should have altered the weights for voting, reducing GNB's weighting to much less than RF's and SVM's.

Table 7.1: Average Scoring Results Over Both Datasets for ECLF1 with Tuned Weights

Accuracy	F1 Score	Sensitivity	Fall out	AUROC	Score Time	Fit Time	Error Rate
0.959	0.961	0.948	0.0277	0.993	0.133	21.7	0.0407

We ran a very small experiment, precisely the same as before, with `weights = [0.9, 1.1, 0.65]` in `VotingClassifier` to see how this would affect the scores. The results of this can be seen in Table 7.1. An immediate improvement can be seen when compared with the unweighted ECLF1. With further tuning we conjecture it would outperform RF.

7.5 ECLF2

ECLF2's performance is largely due to RF. SVM is the second best base classifier so it did not impede the performance of RF significantly. Hence why ECLF2's performance is largely the same as RF's.

7.6 ECLF3

ECLF3 was the worst performing voting classifier. This is due to it being the combination of the two worst base classifiers: GNB and SVM. As before, ECLF3 performs better than GNB alone as its performance is increased by SVM.

7.7 ECLF4

ECLF4 performed minorly better than ECLF3. This performance can be attributed to RF's superiority over SVM.

7.8 Hypotheses

h_1 This is not true. The performance of SVM and RF was hindered when combined with GNB.
h_2 We accept this hypothesis. No ensemble outperformed RF, even after altering the weights.
h_3 We reject this hypothesis. ECLF2 performed the best, which did not contain the most base classifiers. The best ensemble was a combination of the best base classifiers.
h_4 This is not true in general. The performance of RF was always hindered when part of an ensemble. However, the performance of weak classifiers was improved when combined with stronger classifiers. For example, ECLF4 performed much better than GNB alone and this is due to RF correcting GNB's misclassifications.
h_5 We discuss this in Conclusions.

Furthermore, AUROC was higher across the board for the ensemble classifiers. We conjecture this is because the more voters in an ensemble the greater the $Proba^j(\mathbf{x})$ for the correct class. This increases the AUROC as it means the correct classification will be made at more of the thresholds tested over.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

Achieving accurate results for Machine Learning does not seem to be a challenging task. With the abundance of feature extraction research and ML libraries, very few phishing emails should make it past detection. However, [14] shows us that the research on detection rarely concentrates on the user-side of detection (e.g. fully fledged software solutions). Furthermore, knowledge of detection techniques allows phishing scammers to bypass detection by exploiting weaknesses [2].

Detection using ML also requires regular retraining. In particular, features we extract should change over time as phishers learn what bypasses detection. Choosing new features to extract must be done by humans, decreasing the level of automation ML hopes to provide.

We believe creating effective algorithms to detect phishing scams lies with researching ensemble techniques. This is because ensemble algorithms can enable a reduction in the generalisation error. This is needed when the classification of phishing emails changes so rapidly. We have seen this in our research - RF performed the best by far and it was the best tuned ensemble algorithm we tested.

We tested how well generalised the algorithms were by testing over two datasets, which we did not see in any literature for ensemble algorithms relating to phishing.

8.2 Future Work

During our experiments we have discovered many new areas of interest. In this section we outline them and state the rationale for them:

1. Analyse the diversity of algorithms using several metrics outlined in [11]. This will allow us to choose highly diverse algorithms which will increase performance;
2. Run tests on different parameters to conclude which parameters reduce the generalisation error;
3. Gather results for more ensembles, comparing different methods of combining. For example, boosting, bagging and non-simple soft voting;

4. Run cross validation with more folds to evaluate the models more accurately by identifying overfitting;
5. Run tests on newer emails to prove the legitimacy of our algorithms in real practice;
6. Run tests with different weights in soft voting to improve the performance of our algorithms.

Bibliography

[1] Afzal, S., Asim, M., Javed, A.R., Beg, M.O. and Baker, T., 2021. Urldeepdetect: A deep learning approach for detecting malicious urls using semantic vector models. *Journal of network and systems management* [Online], 29(3), p.21. Available from: https://doi.org/10.1007/s10922-021-09587-8.

[2] AlEroud, A. and Karabatis, G., 2020. Bypassing detection of url-based phishing attacks using generative adversarial deep neural networks [Online]. *Proceedings of the sixth international workshop on security and privacy analytics*. New York, NY, USA: Association for Computing Machinery, IWSPA '20, p.53–60. Available from: https://doi.org/10.1145/3375708.3380315.

[3] Anti Phishing Working Group, 2007. Phishing activity trends report [Online]. Available from: https://docs.apwg.org/reports/apwg_report_may_2007.pdf.

[4] Anti Phishing Working Group, 2021. Phishing activity trends report [Online]. Available from: https://docs.apwg.org/reports/apwg_trends_report_q4_2021.pdf?_ga=2.135071922.228042674.1646011812-618113612.1646011812&_gl=1*14wpjo8*_ga*NjE4MTEzNjEyLjE2NDYwMTE4MTI.*_ga_55RF0RHXSR*MTY0NjA1NTMzNS4yLjEuMTY0NjA1NTM0Mi4w.

[5] Area1Security, 2021. It started out with a phish: Area 1 security 2021 email threat report. [Online]. Available from: https://files.area1security.com/reports/Area-1-Security-2021-Security-Report.pdf.

[6] Atla, A., Tada, R., Sheng, V. and Singireddy, N., 2011. Sensitivity of different machine learning algorithms to noise. *J. comput. sci. coll.*, 26(5), p.96–103.

[7] Basit, A., Zafar, M., Javed, A.R. and Jalil, Z., 2020. A novel ensemble machine learning method to detect phishing attack [Online]. *2020 ieee 23rd international multitopic conference (inmic)*. pp.1–5. Available from: https://doi.org/10.1109/INMIC50486.2020.9318210.

[8] Breiman, L., 1996. Bagging predictors. *Machine learning* [Online], 24(2), pp.123–140. Available from: https://doi.org/10.1007/BF00058655.

[9] Breiman, L., 2001. Random forests. *Machine learning* [Online], 45(1), pp.5–32. Available from: https://doi.org/10.1023/A:1010933404324.

[10] Breiman, L., Friedman, J., Stone, C.J. and Olshen, R.A., 1984. *Classification and regression trees*. Boca Raton, Florida: Chapman & Hall :CRC.

[11] Brown, G., Wyatt, J., Harris, R. and Yao, X., 2005. Diversity creation methods: a survey and categorisation. *Information fusion* [Online], 6(1), pp.5–20. Diversity in Multiple

Classifier Systems. Available from: https://doi.org/https://doi.org/10.1016/j.inffus.2004.04.004.

[12] Burkov, A., 2019. *The hundred-page machine learning book*. [Place of publication not identified]: Andriy Burkov.

[13] Chiew, K.L., Tan, C.L., Wong, K., Yong, K.S. and Tiong, W.K., 2019. A new hybrid ensemble feature selection framework for machine learning-based phishing detection system. *Information sciences* [Online], 484, pp.153–166. Available from: https://doi.org/https://doi.org/10.1016/j.ins.2019.01.064.

[14] Das, S., Kim, A., Tingle, Z. and Nippert-Eng, C., 2019. All about phishing: Exploring user research through a systematic literature review. 1908.05897.

[15] Dhamija, R., Tygar, J.D. and Hearst, M., 2006. Why phishing works [Online]. *Proceedings of the sigchi conference on human factors in computing systems*. New York, NY, USA: Association for Computing Machinery, CHI '06, p.581–590. Available from: https://doi.org/10.1145/1124772.1124861.

[16] Dua, D. and Graff, C., 2015. UCI machine learning repository [Online]. Available from: https://archive.ics.uci.edu/ml/datasets/Phishing+Websites.

[17] Fette, I., Sadeh, N. and Tomasic, A., 2007. Learning to detect phishing emails [Online]. *Proceedings of the 16th international conference on world wide web*. New York, NY, USA: Association for Computing Machinery, WWW '07, p.649–656. Available from: https://doi.org/10.1145/1242572.1242660.

[18] Fu, L., Liang, P., Li, X. and Yang, C., 2021. A machine learning based ensemble method for automatic multiclass classification of decisions [Online]. *Evaluation and assessment in software engineering*. New York, NY, USA: Association for Computing Machinery, EASE 2021, p.40–49. Available from: https://doi.org/10.1145/3463274.3463325.

[19] Hannousse, A. and Yahiouche, S., 2021. Towards benchmark datasets for machine learning based website phishing detection: An experimental study. *Engineering applications of artificial intelligence* [Online], 104, p.104347. Available from: https://doi.org/10.1016/j.engappai.2021.104347.

[20] Heartfield, R., Loukas, G. and Gan, D., 2017. An eye for deception: A case study in utilizing the human-as-a-security-sensor paradigm to detect zero-day semantic social engineering attacks [Online]. *2017 ieee 15th international conference on software engineering research, management and applications (sera)*. pp.371–378. Available from: https://doi.org/10.1109/SERA.2017.7965754.

[21] International Public Sector Fraud Forum, Cabinet Office, Commonwealth Fraud Prevention Centre, 2020. Guide to understanding the total impact of fraud [Online]. Available from: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/866608/2377_The_Impact_of_Fraud_AW__4_.pdf.

[22] Jain, A.K. and Gupta, B.B., 2018. Towards detection of phishing websites on client-side using machine learning based approach. *Telecommunication systems* [Online], 68(4), pp.687–700. Available from: https://doi.org/10.1007/s11235-017-0414-0.

[23] Khan, S.A., Khan, W. and Hussain, A., 2020. Phishing attacks and websites classification

using machine learning and multiple datasets (a comparative analysis) [Online]. *Intelligent computing methodologies*. Springer International Publishing, pp.301–313. Available from: `https://doi.org/10.1007/978-3-030-60796-8_26`.

[24] Liu, Y., Wang, Y. and Zhang, J., 2012. New machine learning algorithm: Random forest. In: B. Liu, M. Ma and J. Chang, eds. *Information computing and applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp.246–252.

[25] Moore, T. and Clayton, R., 2007. Examining the impact of website take-down on phishing [Online]. *Proceedings of the anti-phishing working groups 2nd annual ecrime researchers summit*. New York, NY, USA: Association for Computing Machinery, eCrime '07, p.1–13. Available from: `https://doi.org/10.1145/1299015.1299016`.

[26] Peachey, K., 2021. Fresh warnings over royal mail parcel scam [Online]. Available from: `https://www.bbc.co.uk/news/business-56496203`.

[27] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E., 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research*, 12, pp.2825–2830.

[28] Rajab, M., 2018. An anti-phishing method based on feature analysis [Online]. *Proceedings of the 2nd international conference on machine learning and soft computing*. New York, NY, USA: Association for Computing Machinery, ICMLSC '18, p.133–139. Available from: `https://doi.org/10.1145/3184066.3184082`.

[29] Russell, S.J.S.J., 2016. *Artificial intelligence : a modern approach*, Prentice Hall series in artificial intelligence. Third edition / contributing writers, ernest davis [and seven others].; global edition. ed. Pearson.

[30] Salahdine, F., El Mrabet, Z. and Kaabouch, N., 2021. Phishing attacks detection a machine learning-based approach [Online]. *2021 ieee 12th annual ubiquitous computing, electronics mobile communication conference (uemcon)*. pp.0250–0255. Available from: `https://doi.org/10.1109/UEMCON53757.2021.9666627`.

[31] Shahrivari, V., Darabi, M.M. and Izadi, M., 2020. Phishing detection using machine learning techniques [Online]. Available from: `https://doi.org/10.48550/ARXIV.2009.11116`.

[32] Tan, C.L., 2018. Phishing dataset for machine learning: Feature evaluation [Online]. Available from: `https://doi.org/10.17632/h3cgnj8hft.1`.

[33] Verbaeten, S. and Van Assche, A., 2003. Ensemble methods for noise elimination in classification problems. In: T. Windeatt and F. Roli, eds. *Multiple classifier systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp.317–325.

[34] Zhou, Z.H., 2012. *Ensemble methods : foundations and algorithms*, Chapman & Hall/CRC machine learning & pattern recognition series. Boca Raton, Fla: CRC Press.

[35] Zhou, Z.H., 2021. *Machine learning*. Singapore: Springer Singapore Pte. Limited.

[36] Zhu, X., Wu, X. and Chen, Q., 2003. Eliminating class noise in large datasets. *Proceedings of the twentieth international conference on international conference on machine learning*. AAAI Press, ICML'03, p.920–927.

Appendix A

Raw Results Output

In this chapter we list all the results we recorded over both datasets. Table A.1 lists the results for Dataset 1, Table A.2 lists the results for Dataset 2 and Table A.3 lists the mean average of the results for Dataset 1 and 2.

Table A.1: Scoring Results For Dataset 1

	SVM	RF	GNB	ECLF1	ECLF2	ECLF3	ECLF4
Accuracy	0.934	0.974	0.843	0.952	0.965	0.849	0.894
F1 Score	0.934	0.974	0.825	0.951	0.965	0.832	0.887
Precision	0.925	0.978	0.929	0.969	0.966	0.936	0.942
Sensitivity	0.944	0.970	0.743	0.934	0.965	0.748	0.838
Fall Out	0.0766	0.0214	0.0568	0.0294	0.0342	0.0508	0.0512
Specificity	0.923	0.979	0.943	0.971	0.966	0.949	0.949
AUROC	0.978	0.996	0.932	0.990	0.995	0.971	0.984
Negative Predictive Value	0.943	0.970	0.786	0.936	0.965	0.790	0.854
False Negative Rate	0.0558	0.0300	0.257	0.0662	0.0350	0.252	0.162
False Discovery Rate	0.0750	0.0216	0.0711	0.0305	0.0342	0.0636	0.0576
Score Time	0.0577	0.0297	0.00229	0.105	0.100	0.0783	0.0428
Fit Time	33.9	0.582	0.00811	33.7	34.5	35.5	0.917
Error Rate	0.0662	0.0257	0.157	0.0478	0.0346	0.151	0.106

Table A.2: Scoring Results For Dataset 2

	SVM	RF	GNB	ECLF1	ECLF2	ECLF3	ECLF4
Accuracy	0.926	0.971	0.606	0.944	0.956	0.724	0.810
F1 Score	0.935	0.974	0.453	0.948	0.961	0.672	0.795
Precision	0.922	0.966	0.994	0.980	0.950	0.992	0.997
Sensitivity	0.947	0.982	0.293	0.919	0.972	0.509	0.661
Fall Out	0.100	0.0435	0.00204	0.0235	0.0639	0.00490	0.00245
Specificity	0.900	0.957	0.998	0.977	0.936	0.995	0.998
AUROC	0.975	0.994	0.962	0.992	0.993	0.975	0.994
Negative Predictive Value	0.931	0.977	0.529	0.905	0.964	0.617	0.701
False Negative Rate	0.0531	0.0182	0.707	0.0814	0.0276	0.491	0.339
False Discovery Rate	0.0775	0.0340	0.00551	0.0199	0.0497	0.00761	0.00294
Score Time	0.0900	0.0352	0.00201	0.144	0.150	0.118	0.0514
Fit Time	4.62	0.398	0.00532	4.93	5.18	4.96	0.768
Error Rate	0.0739	0.0294	0.394	0.0557	0.0437	0.276	0.190

Table A.3: Average Scoring Results Over Both Datasets

	SVM	RF	GNB	ECLF1	ECLF2	ECLF3	ECLF4
Accuracy	0.930	0.973	0.725	0.948	0.961	0.787	0.852
F1 Score	0.935	0.974	0.639	0.95	0.963	0.752	0.841
Precision	0.924	0.972	0.962	0.975	0.958	0.964	0.970
Sensitivity	0.946	0.976	0.518	0.927	0.969	0.629	0.750
Fall Out	0.0883	0.0325	0.0294	0.0265	0.0491	0.0279	0.0268
Specificity	0.912	0.968	0.971	0.974	0.951	0.972	0.974
AUROC	0.977	0.995	0.947	0.991	0.994	0.973	0.989
Negative Predictive Value	0.937	0.974	0.658	0.921	0.965	0.704	0.778
False Negative Rate	0.0545	0.0241	0.482	0.0738	0.0313	0.372	0.251
False Discovery Rate	0.0763	0.0278	0.0383	0.0252	0.0420	0.0356	0.0303
Score Time	0.0739	0.0325	0.00215	0.125	0.125	0.0982	0.0471
Fit Time	19.3	0.490	0.00672	19.3	19.8	20.2	0.843
Error Rate	0.0701	0.0276	0.276	0.0518	0.0392	0.214	0.148

Appendix B

Code

In this chapter we include the code listing for the software used to execute the experiments. We only used one file to run tests, changing the value of `dataset` and `config` to alter the experiments.

B.1 File: EnsembleLearningDetection.py

```python
import sklearn as sk
import pandas as pd
import numpy as np
import csv
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier,
    VotingClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import confusion_matrix, roc_auc_score,
    roc_curve
import matplotlib.pyplot as plt
from timeit import default_timer as timer

config="ECLF1␣test" #The name given to the classifier in the CSV
    file.
dataset = 1 #The dataset we are testing over.
cv=5 #Number of folds in cross validation.
y_probs = []
y_preds = []
y_tests = []
scoreTimes = []
fitTimes = []

#Single classifiers
SVM = SVC(kernel='linear', probability=True)
RF = RandomForestClassifier()
GNB = GaussianNB()

#Ensemble classifiers
eclf = VotingClassifier(estimators=[
        ('svm',SVM),('rf',RF),('gnb',GNB)], voting='soft',
            n_jobs=-1)

data = pd.read_csv("Datasets/Dataset"+str(dataset)+".csv")
if dataset==1:
    X, y = data.iloc[:,1:49], data.iloc[:,49]
if dataset==2:
    X, y = data.iloc[:,1:31], data.iloc[:,31]

skf = StratifiedKFold(n_splits = cv)

for train_index, test_index in skf.split(X,y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    fitTime1 = timer()
    eclf.fit(X_train, y_train)
    fitTime2=timer()
    fitTime = fitTime2-fitTime1
    fitTimes.append(fitTime)
    scoreTime1=timer()
    y_pred = eclf.predict(X_test)
    scoreTime2=timer()
    scoreTime = scoreTime2-scoreTime1
    scoreTimes.append(scoreTime)
    y_prob = eclf.predict_proba(X_test)
    y_preds+=y_pred.tolist()
    y_tests+=y_test.tolist()
    y_probs+=y_prob[:,1].tolist() #Gets the positive class
        predictions.


conf_matrix = confusion_matrix(y_tests, y_preds)

TN=conf_matrix[0,0]
FP=conf_matrix[0,1]
FN=conf_matrix[1,0]
TP=conf_matrix[1,1]
#Sensitivity
Sensitivity = TP/(TP+FN)
#Specificity
Specificity = TN/(TN+FP)
#Precision
Precision = TP/(TP+FP)
#Negative predictive value
NPV = TN/(TN+FN)
#Fall out
Fallout = FP/(FP+TN)
#False negative rate
FNR = FN/(TP+FN)
#False discovery rate
FDR = FP/(TP+FP)
#Error rate
ERR=(FP+FN)/(TP+FP+FN+TN)
#Accuracy
ACC = (TP+TN)/(TP+FP+FN+TN)
#F1 Acore
F1Score = TP/(TP+0.5*(FP+FN))
#AUROC score
AUROC = roc_auc_score(y_tests, y_probs)
```

```
averageScoreTime = 0
averageFitTime = 0

#Calculates average fit and score times.
for fitTime in fitTimes:
    averageFitTime+=fitTime

for scoreTime in scoreTimes:
    averageScoreTime+=scoreTime

averageScoreTime /=cv
averageFitTime /=cv

plt.clf()
#Creates ROC curve.
FPRs, TPRs, curThresholds = roc_curve(y_tests,y_probs)
plt.plot(FPRs, TPRs, label = "AUC␣=␣" + str(AUROC))
plt.xlabel('1␣−␣Specificity')
plt.ylabel('Sensitivity')
plt.title(config+"␣(Dataset␣"+str(dataset)+")␣Average␣Receiver␣
    Operating␣Characteristic␣Curve")
plt.legend()
plt.savefig("Graphs/"+config+"_average_dataset="+str(dataset)+".png",
    dpi=1000,bbox_inches='tight')

#Writes results to CSV file.
with open('scoring_results_dataset='+str(dataset)+'.csv', 'a',
    encoding='UTF8', newline='\n') as f:
    writer = csv.writer(f)
    writer.writerow([config,ACC,F1Score,Precision,Sensitivity,Fallout,Specificity,
```