September 18th, 2020
University of Twente
Enschede

Improving the Effectiveness of Phishing Detection Using Lexical Semantics

A Machine-Learning based approach

By:

Kylian Rijnbergen

Supervisors:

Abhishta, Abhishta

van Capelleveen, Guido

Bachelor Thesis Industrial Engineering and Management

UNIVERSITY OF TWENTE

This page is intentionally left blank.

Acknowledgements

A great part of the past half-year has been devoted to conducting research and writing my thesis. Despite spending hours upon hours in isolation to code, gather knowledge, or write, I could not have done it alone.

First of all, I would like to thank my supervisors, Abhishita Abhishta and Guido van Capelleveen, for their enthusiastic guidance and valuable feedback.

In the era of COVID-19, social interaction is not to be taken for granted. Interacting with my friends and family has changed. But not with my housemates. Thank you for being here with me every single day.

I would like to thank my closest friends. Whether it be playing games, working out, or having a good conversation, spending time with you is an absolute delight.

Last but not least, I would like to express my deepest gratitude to my family for their invaluable support. You have been of great value in both my personal and professional life.

Abstract

Many share the opinion that phishing emails should automatically be detected, such that these emails can be filtered out and do not end up in our inbox. However, a method that perfectly does this has not yet been found. Prior research describes several methods that attempt to identify phishing emails based on structural properties, but to our knowledge, a better alternative does not yet exist. In this thesis, we propose a method that allows us to filter out these emails based on lexical semantics. We make use of machine learning-based algorithms in combination with a technique that carries the name of word embeddings, to design a method that can be used in automatic email classification. By implementing this method, we can let our computers automatically filter emails by making a judgement based on the contents of the emails, just like how they are presented to us as human beings.

Contents

1 Introduction 1
1.1 Problem Area 1
1.2 Core Problem 1
1.3 Research Questions 1
1.4 Methodology: Design Science 2
1.5 Outline 4

2 Background 5
2.1 Introduction to AI and Machine Learning 5
2.1.1 Example: RoboCup 5
2.1.1.1 Artificial Intelligence 6
2.1.1.2 Machine Learning 6
2.2 Machine Learning Concepts 6
2.2.1 The Classification Problem 6
2.2.2 Classification Algorithms 6
2.2.2.1 The Support Vector Machine 7
2.2.2.2 Random Forests 7
2.2.2.3 Logistic Regression 7
2.2.2.4 Neural Networks and the Multilayer Perceptron 7
2.2.2.5 Adaptive Boosting 8
2.2.3 Features in Email Classification 8
2.2.3.1 Descriptive Features 9
2.2.3.2 Semantic Features 9
2.3 Related Work 11
2.4 Contribution to the Scientific Database 12

3 Research Approach 13
3.1 Research Methodology 13
3.2 Python and Scikit-Learn for Machine Learning 14
3.3 The Data Set 14
3.4 Data Pre-Processing 14
3.5 Labelling 15
3.6 Operationalisation and Performance Metrics 15

4 Solution Design 16
4.1 Data Augmentation 19
4.1.1 Approach One: Increasing the Number of Samples 20
4.1.2 Approach Two: Increasing Data Diversity 20
4.2 Designing Experiments 22
4.3 Comparing Algorithms and Configurations 22
4.3.1 Comparison of Algorithms and the Effect of Augmentation 23
4.3.2 The Five Best Models 26
4.4 Hyper-Parameter Optimisation 27

5 Experiments 29
5.1 Performance of the Best Classifier 29

5.1.1 Accuracy, Precision, Recall, and F1-Score 29
5.1.2 Confusion Matrices 30
5.1.3 Precision-Recall Curves 31
5.1.4 Receiver Operating Characteristics 33

6 Discussion **35**
6.1 Interpretation 35
6.2 Generalisable Theory 36
6.3 Limitations 36
6.4 Relation to Prior Research 37

7 Conclusions **38**
7.1 Research Questions Revisited 38
7.1.1 Main Research Question 39
7.2 Implications 39
7.3 Recommendations 40

References **41**

Appendix **43**
A1 Grid-Search Parameter Space 43
A2 Second Grid-Search Parameter Space 44
A3 Initial Highest Scoring Configurations 45
A4 Labelling Policy 47
A4.1 Definition of the Research Question 47
A4.2 Inclusion and Exclusion Criteria 47
A4.2.1 Inclusion Criteria 47
A4.2.2 Exclusion Criteria 47
A4.3 Used Databases 47
A4.4 Search Terms and Used Strategy 47
A4.5 Found Articles 48
A4.6 Conceptual Matrix 49
A4.7 Integration of Theory 50
A4.7.1 Definition of Phishing 50
A4.7.2 Labelling Policy 51
A4.8 Some Final Notes on Labelling 52
A5 Performance Metrics 53
A5.1 Accuracy 53
A5.2 Precision 54
A5.3 Recall 54
A5.4 F_β-Score 54
A5.4.1 F1-Score 55
A5.5 Precision-Recall and ROC Curves 55
A5.5.1 Precision-Recall Curves 55
A5.5.2 Receiver Operating Characteristics 56

List of Figures

4.1 Comparison of Accuracy for all Algorithms 24
4.2 Comparison of Precision for all Algorithms 24
4.3 Comparison of Recall for all Algorithms 25
4.4 Comparison of F1-Score for all Algorithms 25
4.5 Effects of Parameters C and Kernel on F1-Score 27
4.6 Optimisation of C and Augmentation Factor for SVM, Kernel = RBF . . 28
5.1 Precision-Recall Curve for Best Random State 31
5.2 Precision-Recall Curve for Worst Random State 32
5.3 Receiver Operating Characteristic for Best Random State 33
5.4 Receiver Operating Characteristic for Worst Random State 34
A5.1 Example of a Precision-Recall Curve 56
A5.2 Example of a Receiver Operating Characteristic Curve 57

List of Tables

1.1 Stakeholder Analysis . . . 3
2.1 Fruits and their Features . . . 9
3.1 Design Science Research Guidelines . . . 13
3.2 Performance Metrics . . . 15
4.1 Bag-of-Word Representations of Sample Texts . . . 16
4.2 Bag-of-Word Representations of Sample Texts - Normalised . . . 17
4.3 Comparison of Sample Texts . . . 18
4.4 Word Embeddings . . . 18
4.5 Synthetic Email Greetings . . . 20
4.6 Vectorising Texts and Similar Texts . . . 21
4.7 Creating Synthetic Vectors . . . 22
4.8 Best Scoring Configuration for each Algorithm Over Ten Random States 23
4.9 Best Scoring Configurations Over Ten Random States . . . 26
5.1 Model Performance . . . 29
5.2 Best, Worst, and Average Random State Performance for Top Configuration 30
5.3 Confusion Matrix for Best Random State . . . 30
5.4 Confusion Matrix for Worst Random State . . . 30
A1.1 Grid-Search Parameter Space . . . 43
A2.1 Second Grid-search Parameter Space . . . 44
A3.1 Best Configurations for the Support Vector Machine . . . 45
A3.2 Best Configurations for the Random Forests Algorithm . . . 45
A3.3 Best Configurations for Logistic Regression . . . 45
A3.4 Best Configurations for the Multilayer Perceptron . . . 46
A3.5 Best Configurations for Adaptive Boosting . . . 46
A4.1 Search Matrix . . . 48
A4.2 Found Articles . . . 48
A4.3 Articles Used in Defining Phishing . . . 49
A4.4 Conceptual Matrix . . . 49
A4.5 Overview of Concepts . . . 50
A4.6 Labelling Policy for Phishing Emails . . . 51
A5.1 Confusion Matrix . . . 53

1 Introduction

Due to technological developments in the last few decades, valuable information is stored and processed online. While this has very significant benefits, there are some downsides to this digitalization. Online criminals, more commonly known as cybercriminals, have succeeded in performing their criminal activities online, in a wide variety of fields. Some fields are relevant for individuals, some for businesses and some are relevant for both.

1.1 Problem Area

Phishing, sometimes referred to as spoofing, is one of those fields. In phishing, cybercriminals attempt to bypass security measures such as passwords by "fishing" for information. Cybercriminals interact with users, usually pretending to be part of a trusted instance. During their interaction, they try to obtain valuable information such as login credentials, credit card numbers, and social security numbers, among others (Lastdrager, 2018).

1.2 Core Problem

Most of the current email filtering methods do not specifically filter for phishing emails. Instead, they attempt to make a distinction between spam emails and regular emails, the latter also going by the name of ham emails (Schryen, 2007). In this study, we focus on improving the effectiveness of detecting phishing emails in specific.

1.3 Research Questions

As mentioned, this study specifically targets phishing emails. To do this, we explore machine learning methods. We use these methods to classify phishing emails from benign spam (i.e. all spam emails that are not phishing emails). In this section, we present a set of questions that we will answer to achieve our goal of improving the effectiveness of phishing detection. To answer the main question in a systematic manner, we break it down into three sub-questions.

In addition to the main question and related sub-questions, we formulate a question that is used to review our contributions to the scientific database and impact on the field of email filtering.

Our main research question is as follows:
Which machine learning methods can be used to effectively detect phishing emails?
This question has been broken down into the following sub-questions:

SQ1. *How can we prepare our data for further analysis?*

SQ2. *What are the most effective classification algorithms for our problem?*

SQ3. *How can we optimize hyper-parameters in our model to attain high performance?*

After answering these questions, we review our research by answering the following review question:
How do our found methods contribute to the field of email filtering and text classification?

Throughout the report, we will answer our sub-questions. SQ1 is answered in Chapter 3, and SQ2 and SQ3 are answered in Chapter 4. We answer our main question as well as our review question in Chapters 6.2 and 7.

1.4 Methodology: Design Science

The problem of email filtering and phishing detection is a type of problem methodologically known as a design problem. Design problems are problems that require an analysis of a situation in the real world in the form of a stakeholder analysis. Satisfying these stakeholder goals is the main objective of a solution to a design problem. It is important to note that, unlike in knowledge problems, a wide variety of solutions can coexist (Wieringa, 2014).

As design problems are fundamentally different from knowledge problems (a knowledge problem is a problem that asks for knowledge as it is), we need to take a different approach to solving these types of problems. The approach we will be taking is known as design science. In design science, we design an artifact to improve a problem context, in our case, phishing detection. This artifact is designed with the objective of satisfying stakeholder needs. As different stakeholders may have different needs, a single best answer does not exist in most cases (Wieringa, 2014).

As not all stakeholders have the same needs, not all stakeholders will benefit equally from a certain solution. Despite this, solutions need to be evaluated. In design science, this is done based on their utility. Utility, in turn, is dependent on stakeholder goals. In order to design a good solution, we will thus have to identify who our stakeholders are. In Table 1.1 below, we identify our stakeholders. We also provide brief explanations on what their needs are.

Table 1.1: Stakeholder Analysis

Stakeholder	**Needs**
1. Other researchers in the field of phishing detection, email filtering, and natural language processing	Other researchers in the field will want a solution that provides insights into the performance of our artifact. They are interested in how the artifact performs, why this is the case, and how we can apply the practices in our research to various fields in order to improve related problem contexts.
2. Policy makers for email filtering	Policy makers for email filtering are mainly interested in solutions that perform well. They want solutions that can correctly identify the vast majority of phishing emails without falsely identifying a regular email as being a phishing email. In addition to this, policy makers for email filtering also want insights on how our artifact can be tuned to such that they can make a trade-off between detection rates and false positive rates.
3. Email users	Although not as predominant, email users can also be seen as stakeholders in our research. By conducting this research, we potentially affect email users as our studies may influence beforementioned policy makers.

We identify other researchers to be our most important stakeholders. For our artifact, this means our solution is to be designed with the goal of finding new methods and approaches to solving our problem. We aim to retrieve generalisable knowledge that can be used in future research, as well as in solutions to current real-world problems.

Designing and improving this artifact is done based on seven guidelines. These guidelines, as well as how our research applies these guidelines, are found in Section 3.1.

1.5 Outline

We start off the next section (Chapter 2) by presenting the basic concepts related to artificial intelligence and machine learning. Following this, we dive deeper into the theory related to our type of problem. After presenting related theory, we discuss prior research and our own contributions to the field. In Chapter 3, we discuss our research approach and methodology. We also explain how we retrieved our data set, as well as the role of the data set in our study. Solution design is covered in Chapter 4, after which we evaluate our solution in Chapter 5 and discuss our research in 6. Finally, we conclude in Chapter 7, where we provide a short recap of our research and discuss future work.

2 Background

As humans, we do not like spending our time on uninspiring tasks, such as doing the dishes or washing our dirty clothes. A lot of these labour-intensive tasks can be partially or fully automated. Automatons (not to be confused with automation) like dishwashers and washing machines allow us to automate these tasks, meaning we have more time we can use to do other things. Some tasks, such as driving, however, require cognitive abilities, and this is where automation starts to get tricky. Automatons cannot think, reason, judge, or make decisions, meaning we have to find a different method of automating these tasks. This is where artificial intelligence comes into play. In artificial intelligence, from now on referred to as AI, we attempt to develop computers and machines to 'think' and act in a similar way to humans.

2.1 Introduction to AI and Machine Learning

Some fields, such as languages or emotions, are very hard for computers to understand. We as humans have learned to understand and apply these fields though numerous years of evolution. In artificial intelligence, we provide machines with the resources needed to imitate human cognitive functions. We program these machines to solve problems and perform tasks that require some form of reasoning. In machine learning, a subset of artificial intelligence, we take a different approach. Instead of programming the machine how to solve problems or make decisions, we teach it how to get better at performing tasks and making decisions through experience (Sheldon, 2019). In essence, we learn it how to learn.

2.1.1 Example: RoboCup

We demonstrate the difference using RoboCup as an example. RoboCup, which is short for Robot Soccer World Cup, is an annual soccer competition played by robots. The competition has several leagues with different restrictions. Examples of restrictions are size, to prevent very large robots that can block the goal entirely, and movement type, being either movement by driving, using wheels or caterpillars, or movement by walking using legs (the humanoid type) (Kitano et al., 1995).

2.1.1.1 Artificial Intelligence

In case we were to develop an AI for this competition, we would program our robots to use certain tactics and strategies by a set of rules. Examples of these rules are: always moving to open space when your team has control of the ball, never standing still, or always attempting to score a point if close to the opposing team's goal. In short, we program robots to do human-like things.

2.1.1.2 Machine Learning

In machine learning, we do none of these things. Instead of providing our robots with rules, we let them perform random actions and provide them with a performance indicator, such as goals scored. Over time, these robots will learn which actions are good in which situations, and how these actions can be combined into a winning strategy. To summarize, we do not program the robots to perform any specific actions. They will learn what to do, when to do it, and how to do it, all by themselves.

2.2 Machine Learning Concepts

Now that we know what machine learning is, let us define the basic concepts related to machine learning. In this section, we will introduce machine learning approaches related to our problem and related theories, as well as performance metrics.

2.2.1 The Classification Problem

In our problem, we make a distinction between benign spam emails and phishing emails. In other words, we classify our emails into two predetermined categories. These types of problems are known as classification problems.

2.2.2 Classification Algorithms

A wide variety of classification algorithms exists. Widely researched algorithms for email classification are the Support Vector Machine, Random Forests, Logistic Regression and Neural Networks (Abu-Nimeh et al., 2007). We use these algorithms as a basis for our research. In addition, we will evaluate the performance of the adaptive boosting algorithm, also known as AdaBoost. We provide brief explanations for each of these algorithms in the following five sections.

2.2.2.1 The Support Vector Machine

The Support Vector Machine (SVM) is one of the most popular classifying algorithms. This algorithm tries to separate our data points in space. This is done by fitting the hyperplane which maximizes the margin (distance from the points to the plane). Points that are on the boundaries of the margins are known as support vectors. While SVMs generally perform very well, their computation is intensive, and they are prone to overfitting (Abu-Nimeh et al., 2007). Overfitting is a modelling error that occurs when a model fails to make a distinction between significant features and residual variation (e.g. noise).

2.2.2.2 Random Forests

The Random Forests classifier (RF) is a classifier that makes use of decision trees. It generates a large number of decision trees, each tree using a random numer of samples and features from our data set. If classifiable data are input, the classifier returns the label that was decided on by the largest number of decision trees (Tin Kam Ho, 1995).

2.2.2.3 Logistic Regression

Logistic Regression (LogReg) is a well-established statistical model for classifying data. It binarily classifies data by fitting the data points to a logistic function. The classifier is very powerful for simple, linearly separable data, but its performance starts to decline for data with complex relationships between variables (Abu-Nimeh et al., 2007).

2.2.2.4 Neural Networks and the Multilayer Perceptron

Neural Networks are a type of classifier that attempt to mimic the biological brain. The network consists of connected layers of so-called nodes, which resemble neurons as we know them in biology. While neurons cannot do much by themselves, introducing a proper amount of connected neurons allows for evaluation of complex functions. Some of the simplest, though very useful structures can emulate logic gates (Obumneme Dukor, 2018). Slightly more complex networks are capable of classifying linearly separable data, but clever manipulation of input features (i.e. by the use of kernel functions) allows us to circumvent even this constraint (Freund and Schapire, 1999). Neural networks are highly configurable by setting hyper-parameters such as the number of hidden layers, number of neurons per layer and the optimization algorithm. Configuring the hyper-parameters of neural networks is a nontrivial task that generally requires a lot of experience and knowledge, though we can often initialize the model by making some educated guesses and improve from there (Alto, 2019).

Though many different types of neural networks exist, our research focuses on the Multilayer Perceptron (MLP), which is a type of feed-forward neural network. In feed-forward neural networks, nodes do not form a cycle, meaning neurons only output to the next layer of neurons. The Multilayer Perceptron is one of the most basic versions of the neural network, consisting of only an input layer, a configurable number of hidden layers and an output layer (Bishop, 2013).

2.2.2.5 Adaptive Boosting

Adaptive Boosting, also known as AdaBoost, is a machine-learning algorithm that combines a set of weaker classifiers into a stronger one. The classifier selects a "team" of other, simple, classifiers, such as the SVM (2.2.2.1) and Random Forests (2.2.2.2), and assigns a weight to each classifier. These individual classifiers will solve the problem separately, and vote on a decision based on their prediction. The AdaBoost algorithm then evaluates their performance and re-assigns the individual classifiers a weight based on used metrics. This process is repeated until the stopping criteria are met. The most effective combination of classifiers, along with their weights, will act as our final classifier. (Freund and Schapire, 1995) (Rojas, 2009).

2.2.3 Features in Email Classification

In classification problems, items are classified based on their properties. These properties, also known as explanatory variables or features, describe aspects of the classifiable item. An example for fruits is provided in table 2.1 below.

Table 2.1: Fruits and their Features

Fruit	**Shape**	**Colour**
Banana	Curved	Yellow
Lemon	Round	Yellow
Persimmon	Round	Orange
Clementine	Round	Orange

Now, if we were to classify a box of fruits, all items being either lemons or bananas, we can do this by looking at their shape. In case the box also contains persimmons, we also need to look at their colours. If we also add clementines, however, we run into a problem. The properties shape and colour, are not enough to distinguish clementines from persimmons. We need at least one additional feature to be able to tell these two items apart. In other words, the items "clementine" and "persimmon" are insufficiently described.

In addition to insufficiently describing items, it is also possible to provide too much information. In a box full of only bananas and lemons, the feature: colour, does not contribute to our decision. Non-contributory variables are known as insignificant variables.

Part of the art of properly classifying data lies in feature selection. We need enough unique features to make a distinction between all classes. We also want the number of features to be as low as possible, as adding insignificant features increases complexity without necessity. In our problem of email classification, there are two types of features we can use.

2.2.3.1 Descriptive Features

The first type of features are features we will refer to as "descriptive features". Descriptive features are generic features that describe the email based on its format, or certain aspects of its contents or structure. Commonly used descriptive features in email classification include, but are not limited to, presence of HTML, presence of hyperlinks, number of attachments, and number of words in the email (Martin et al., 2005).

2.2.3.2 Semantic Features

Features that reflect the meanings of words, or so-called semantic features, are the second feature type we can use to solve our problem. Instead of making a decision based on when an email is sent, to many people it is sent, or how it is constructed, we try to understand what the contents of the email mean.

In most cases, one will read the body of an email after opening it. Ham emails are read and answered without any trouble. Oftentimes, junk emails will have something that is a little off. An obnoxious advertisement, lots of images or poor grammar are not uncommon. Only when we realise this, we start looking at some descriptive features such as the presence of hyperlinks or attachments.

Oftentimes, phishing emails are designed to resemble ham emails. A personal greeting is included, the email has no attachments, and the email is well structured. Everything seems perfectly normal until your bank asks for your credit card details.

In using semantic features, we attempt to filter out exactly these emails. If the contents of the mail boil down to: "Provide me with your credit card details.", there is a high chance this email is a phishing email.

Unlike humans, computers do not memorize words by their meaning. Instead, computers memorize the characters in a word. For a computer, the word "cat" is more similar to the word "car" than it is to the word "dog", while most people would argue that the word "cat" is more similar to the word "dog".

Over the last two decades, a technique that goes by the names "word embeddings" and "word vectors", has been developed. This technique maps words to high-dimensional vectors of real numbers, such that words that have similar semantics are close in space (Pennington et al., 2014a) (Alkhereyf and Rambow, 2020). Using these vectors allows us to compare words, usually done by evaluating their cosine similarity (Karani, 2018), find lexical relationships between words by adding and subtracting vectors (Vylomova et al., 2015), and extract other linguistic properties. Popular word embedding models are GloVe, developed by Stanford University, Word2Vec by Google and fastText by FaceBook (Agrawal, 2019). In our research, as well as in most related work, the GloVe model is used. More on GloVe can be found here: (Pennington et al., 2014b).

2.3 Related Work

Email filtering is a concept that has been around for a long time, yet it is still not perfect. Both spam filtering and phishing detection techniques have been developed over the years, most of these techniques making use of what we refer to as descriptive features (2.2.3.1). Fette et al. (2007) use ten features, all being either continuous or binary, along with the Random Forests algorithm (2.2.2.2) to detect phishing emails. Toolan and Carthy (2010) take a similar approach in performing multinomial classification to make a distinction between spam, ham, and phishing emails.

We have found two prior studies that specifically target phishing emails. In one study, this filtering has been done using structural properties of the emails. These properties include 23 style marker attributes (e.g. the total number of words), two structural attributes (e.g. structure of email subject line), and 18 keywords (e.g. "account"). Emails are classified by an algorithm known as the support vector machine (Chandrasekaran and Narayanan, 2006). More information on the support vector machine can be found in 2.2.2.1.

The second study proposes a method given the name PILFER. This method uses ten features for detecting phishing emails. All these features are either binary features or continuous numeric features. Some examples of these features are the inclusion of JavaScript and the output of the spam filter SpamAssassin. Emails are classified using the Random Forests classifier. Some other classfiers are also evaluated. These classifiers yielded similar performance (Fette et al., 2007). More information on the Random Forests classifier can be found in 2.2.2.2.

Closest related to our research is that of Alkhereyf and Rambow (2020). They make use of the GloVe model for word embeddings (Pennington et al., 2014b), along with both the support vector machine (2.2.2.1) and the Extra-Trees algorithm (Geurts et al., 2006), to classify personal and business email.

2.4 Contribution to the Scientific Database

Our research focuses on making a distinction between phishing emails and benign spam by making use of the GloVe model for word embeddings (Pennington et al., 2014b) along with the most popular models for email classification, being the support vector machine as well as the Random Forests algorithm. In addition, we evaluate the performance of logistic regression, the multilayer perceptron and adaptive boosting (2.2.2). We conduct this research to gain a better insight into the performance and implementation of various machine learning algorithms in natural language processing for email classification.

3 Research Approach

In chapter 2, we discussed the current state of email classification. We also provided a brief outline of our research goal and design. In this chapter, we provide an in-depth description of our problem-solving approach.

3.1 Research Methodology

Our research has been conducted using the guidelines of the design science research methodology. In design science research, we design a so-called artifact; in our case, our model, which is iteratively improved upon. The methodology is based on seven guidelines (Hevner et al., 2004). These guidelines, as well as how they were applied in our research, are found in Table 3.1 below.

Table 3.1: Design Science Research Guidelines

Guideline	**Implementation**
1. Design as an Artifact	We have developed a machine learning model for email classification. Instead of taking a more standard approach by using descriptive features, we use semantic features as described in Section 2.2.3.2. We also implement a method that reduces modelling errors such as overfitting. More information on this method is found in Section: 4.1.
2. Problem Relevance	Research in Section 2.3, as well as in Appendix A4.3 shows that phishing detection is still a problem for many email users. Phishing emails end up in their inboxes, often leading to financial, emotional, or reputational damage, as well as a loss of personal information such as social security numbers, leading to identity fraud.
3. Design Evaluation	We have evaluated the performance of our model through the use of performance metrics in Chapter 5. More on performance metrics is found in Appendix A5.
4. Research Contributions	Research contributions are covered in Chapters 2, 6,, and 7
5. Research Rigor	Research methods are covered in Chapter 2 and Chapter 4.
6. Design as a Search Process	We have used peer-reviewed articles, books, papers and documentation, among other sources.
7. Communication of Research	Our research is presented both in this report and during the colloquium.

3.2 Python and Scikit-Learn for Machine Learning

Our study is conducted using Python, a high-level, general-purpose programming language. Python is widely considered to be the preferred language for machine learning purposes (Raschka et al., 2020). Python is an open-source programming language that offers a wide range of data processing libraries, such as NumPy, Pandas and Scikit-Learn (Python Software Foundation, 2020). Our research uses Scikit-Learn as its main library for machine learning. Scikit-Learn offers a wide variety of algorithms, performance metrics, and optimization methods (Hao and Ho, 2019), (Pedregosa et al., 2011).

3.3 The Data Set

Data for our research have been retrieved from the Anti-Phishing Working Group. The data set contains a large number of spam emails, some of which are benign and some of which are phishing emails. Data in our data set have not been labelled, although labels are, in fact, mandatory for our research. We labelled the emails ourselves, more information on this is found in Section 3.5 and in Appendix A4.

3.4 Data Pre-Processing

The data set, as we retrieved it from the APWG, is not in a workable format. We preprocess our data set so that we can assign labels to individual emails. In addition, we perform some preprocessing steps, such that we can use the data, along with its assigned labels, as an input for our machine learning study.

Preprocessing of the data set for labelling purposes is fairly simple. We extract the following features: sender email, subject of the email, and the raw body (i.e. contents) of the email. We use these features in labelling our emails, more on this in the next section.

As explained in 2.2.3.2, our study makes use of semantic features of our email. We have chosen to use the body of the email to perform classification. Unlike in labelling, however, readability is not important. As readability is no longer important, we further process our data. We remove all stop words, convert everything to lower case, remove some HTML tags, and discard all words that are not in the GloVe model.

None of these aspects contribute to the meaning of the emails, so we remove them to reduce noise in our models.

3.5 Labelling

Labelling the emails in our data set is not a trivial task. While most have a decent understanding of the differences between phishing emails and benign spam, there is a lot of grey area. In Appendix A4, we define phishing, and we create a policy table that is used in assigning labels to our emails. Using this policy, we annotated 604 emails, 303 of which were deemed to be phishing emails, with the remaining 301 being benign spam. For the sake of clarity, the definition used in our research is provided below.

"Phishing is a type of computer attack that communicates socially engineered messages to humans via electronic communication channels in order to persuade them to perform certain actions for the attacker's benefit." (Khonji et al., 2013).

3.6 Operationalisation and Performance Metrics

In order to compare and evaluate the performance of our solutions, we use a set of performance metrics. Using these performance metrics, we can identify the best option available for our situation.

We use a set of four performance metrics as a basis for our decision-making. These performance metrics, along with a brief explanation, are provided in Table 3.2 below. A more in-depth explanation, as well as the mathematical representation of these metrics, is found in Appendix A5.

Table 3.2: Performance Metrics

Metric	Explanation
Accuracy	Number of correct predictions as a fraction of total predictions.
Precision	Number of correct positives as a fraction of all predicted positives.
Recall	Number of correct positives as a fraction of all positives.
F1-Score	Harmonic mean between precision and recall.

4 Solution Design

First, we represent our emails by using the bag-of-words model. In this model, texts are represented by a set consisting of their words, along with the frequency in which they occur. Grammar and word order are not taken into account. We provide some examples in Table 4.1 below.

Table 4.1: Bag-of-Word Representations of Sample Texts

Text	**Words**		
	Banana	Lemon	Clementine
Banana	1	0	0
Lemon	0	1	0
Clementine	0	0	1
Banana Banana	2	0	0
Banana Lemon	1	1	0
Lemon Banana	1	1	0
Banana Banana Banana Banana Lemon	4	1	0

In principle, we could use this bag-of-words model as an input for our machine-learning algorithms. Using the model, we have successfully represented our texts as vectors. We can use vector operations to analyse, compare, and even classify texts. We can add the vector belonging to "Banana" to itself to get the vector "Banana Banana". We can also add the vectors "Banana" and "Lemon", which yields the vector belonging to both "Banana Lemon" and "Lemon Banana". While this may not sound all that useful, we can use some basic deductive reasoning to assist in our classification. Without the use of this model, letting a computer compare the texts "Banana Lemon" and "Lemon Banana", will yield no similarity. Using the bag-of-words model, we can tell our computers that these texts, are, in fact, similar.

There are two problems to this approach. The first problem has to do with the number of words in the texts. When given the text "Banana", which is more similar: "Banana Banana" or "Banana Lemon"? Intuitively, "Banana Banana" is the right answer. Using vector operations, however, this is debatable, as the addition of either the vector for "Banana" or the vector for "Lemon" will yield an answer. To solve this problem, we slightly change our approach. Instead of counting the actual number of occurrences of a word, we express its occurrence proportional to the total number of words in the text (i.e.

we normalise each row). In essence, we take the weighted average of the words in the text. The normalised table is found in Table 4.2 below.

Table 4.2: Bag-of-Word Representations of Sample Texts - Normalised

Text	**Words**		
	Banana	Lemon	Clementine
Banana	1	0	0
Lemon	0	1	0
Clementine	0	0	1
Banana Banana	1	0	0
Banana Lemon	0.5	0.5	0
Lemon Banana	0.5	0.5	0
Banana Banana Banana Banana Lemon	0.8	0.2	0

Normalisation of the table has both advantages and disadvantages. In normalising the table, we lost information. The vector belonging to "Banana" is now identical to that of "Banana Banana", while it previously was not. In our problem of email classification, this is a good thing. While the texts are not identical, they communicate similar information.

How about "Banana Banana Banana Banana Lemon"? Is it more similar to "Banana", or is it more similar to "Banana Lemon"? One can find valid arguments for both answers. In our approach, we favour the answer "Banana". While the text is about both bananas and lemons, we consider "Banana" to be sufficiently dominant to conclude the text is mainly about bananas, instead of being about bananas and lemons.

But how dominant is sufficiently dominant? Intuitively speaking, this is a difficult question to answer. Mathematically speaking, answering this question is a fairly straightforward task. We can use the mean squared error metric to decide which text is most similar to the text of interest. As seen in Table 4.3 below, the text is most similar to "Banana", as the MSE is smallest for this vector pair.

Table 4.3: Comparison of Sample Texts

Text	**Words**			**MSE**
	Banana	Lemon	Clementine	
Banana Banana Banana Banana Lemon	0.8	0.2	0	
Banana	1	0	0	
Difference	-0.2	0.2	0	0.08
Banana Banana Banana Banana Lemon	0.8	0.2	0	
Banana Lemon	0.5	0.5	0	
Difference	0.3	-0.3	0	0.18

The second problem to this approach is related to the meanings of the words in our text. While we are able to compare pieces of text based on which words they contain, and how often these words occur, there is still some texts we cannot compare. How does the word "Lemon" compare to the word "Clementine"?

As mentioned in Section 2.2.3.2, we make use of word embeddings in our classification process. We use the pre-trained GloVe model to represent our emails by a vector. This model is trained on the Common Crawl, having 840 billion tokens. The model features 2.2 million entries, all of which are represented in a 300-dimensional vector. Some examples of what these word embeddings look like are provided in Table 4.4 below.

Table 4.4: Word Embeddings

Word	**Dimension**						
	1	2	3	...	298	299	300
Banana	0.22937	-0.071676	-0.080207	...	-0.83732	-0.14226	0.38125
Lemon	-0.26118	0.6852	-0.32639	...	-0.58266	0.092507	0.20233
Persimmon	-0.034397	0.14763	-0.11865	...	-0.004818	-0.12312	0.1458
Clementine	-0.2617	0.069294	-0.34449	...	-0.21177	-0.096107	-0.36148

We combine the main ideas of these two methods to vectorise our emails. Emails will be represented by a weighted average of its word vectors after removal of stop words, words that are not in the GloVe model, some HTML tags, and conversion to lower case.

4.1 Data Augmentation

Just like humans, machine learning algorithms cannot learn everything about an object or the class it belongs to just by seeing an object once. Instead of memorising every detail about a certain type of car (e.g. the diameter of the wheels or the colour), we learn to recognise the most basic properties first. Most cars have four wheels, they have a shape that looks a bit like a box, and a steering wheel is in the front of the car. This system of learning general aspects is very convenient in classification. Firstly, we can recognize a car as being a car, even is we have never seen that type of car before. Secondly, we do not waste our time and energy on memorising irrelevant properties of every single car we have ever seen. Unless you are a car expert or hobbyist, the volume of the exhaust or the weight of the wheels is irrelevant.

In order to properly recognize a car as being a car, however, we need to be exposed to a fairly wide range of cars first. If you have only ever seen black cars with four seats, you may think a red Formula 1 car is a whole different type of vehicle.

In training our algorithms, two things are important. Firstly, we need a lot of data. Secondly, the samples need to have similar properties, but they should not be exact copies of each other as this will cause us to learn about the irrelevant details instead of relevant, general properties. When we learn the irrelevant details instead of relevant properties, we speak of overfitting.

Collecting data is expensive, and labelling data costs a lot of time and effort. In our research, we have labelled a little over 600 samples, 80% of which is used for training our algorithms, and 20% of which is used for performance evaluation. This results in about 483 samples to train our algorithms and 121 samples to evaluate them. While 483 is quite a bit, we would like to have more. We present two approaches to increase the size of our data set without actually collecting and annotating new samples.

4.1.1 Approach One: Increasing the Number of Samples

The simplest approach to creating more training data is providing our data to the algorithm multiple times. Instead of showing the emails once, we show them twenty times. This method allows us to extract a lot of information about our training data, but it is very prone to overfitting.

4.1.2 Approach Two: Increasing Data Diversity

In an attempt to combat overfitting, we need to increase the diversity of our data. Instead of providing the algorithms with the same email twenty times, we can show it the original email once and show a slightly different email twenty more times. These emails can be generated by replacing some words with their synonyms. The label of these slightly different emails is the same as that of the original, as the context of the email remains unchanged. We provide some examples in Table 4.5 below. We will refer to our newly generated data as synthetic data.

Table 4.5: Synthetic Email Greetings

Sample	**Text**
Original	Hello my friend,
Synthetic 1	Hi my friend,
Synthetic 2	Greetings my friend,
Synthetic 3	Hello my buddy,

Although we do not have to collect and label new data using this method, generating this synthetic data is still quite a bit of work. Especially in larger texts, we will have to replace multiple words with their synonyms, or their corresponding vector will not change by a significant amount. In addition, the entire text will have to be vectorised again.

To better understand how we can overcome these two problems, we take a closer look at what replacing words with their synonyms does. We go back to our example sentences from Table 4.5 above. We study our original text, as well as the first synthetic text to get a better idea of what is happening. We first vectorise all words in these sentences individually. These word vectors can be found in Table 4.6 below. As explained in Section 3.4, we convert all of our words to lower case before vectorising.

Table 4.6: Vectorising Texts and Similar Texts

Text	**Dimension**						
	1	2	3	...	298	299	300
"hello"	0.25233	0.10176	-0.67485	...	0.17869	-0.51917	0.33591
"hi"	0.028796	0.41306	-0.4669	...	-0.053029	-0.33494	0.36282
"my"	0.08649	0.14503	-0.4902	...	0.25909	-0.18599	0.0085633
"friend"	0.07781	0.17561	-0.59164	...	-0.076056	-0.12362	0.16777
Original	0.13887667	0.1408	-0.58556333	...	0.120575	-0.27626	0.170748
Synthetic	0.0643653	0.244567	-0.516247	...	0.043335	-0.21485	0.179718

As we can see, our original and synthetic texts have different vectors. The more words we replace by other words, the greater the difference between the new vector and the original vector will be. This difference is directly proportional to the semantic difference between the original word and its replacement. From this we deduce that, the more semantically similar two texts are, the more numerically similar their vectors will be.

We use this information to generate more input data for our study. Instead of generating new data by modifying our samples and then vectorising them, we can vectorise our samples and then modify them. This has three advantages. Firstly, modifying vectors requires fewer resources than modifying emails. Moreover, we do not have to spend any resources on finding synonyms for a large number of words present in our emails. Lastly, we have more control over the degree to which a sample is modified.

We modify our vectors by slightly altering each of their dimensions. We do this by taking the inner product between the original vector and our augmentation vector, denoted by $\overrightarrow{aug}$. $\overrightarrow{aug}$ is a 300-dimensional vector containing independent and identically distributed random numbers according to the normal distribution $X \sim \mathcal{N}(\mu, \sigma^2)$. We use $\mu = 1$ as this causes the average modification to equal zero. We initially estimate σ, which we will refer to as the augmentation factor, to be optimal in the range of [0, 0.05]. In 4.4, this parameter is optimised. In Table 4.7 below, we create a new sample using this method. The new sample is derived from our example text: "Hello my friend".

Table 4.7: Creating Synthetic Vectors

Vector	**Dimension**						
	1	2	3	...	298	299	300
Original	0.25233	0.10176	-0.67485	...	0.17869	-0.51917	0.33591
$\overrightarrow{aug}$, $\sigma = 0.05$	0.952702	0.962165	1.07088	...	0.816022	1.0046	0.962946
Synthetic	0.240395	0.0979099	-0.722686	...	0.145815	-0.521557	0.323463

This approach is used on each of our training samples to create 20 synthetic samples. Testing samples are not augmented using this method as this reduces internal validity.

4.2 Designing Experiments

In order to get an idea of which classifiers are possibly suitable for our problem, we have performed a wide grid-search, evaluating the performance of our classifiers and different hyper-parameter settings. Grid-search parameter space can be found in Table A1.1. All classifiers were evaluated using the same train-test split.

For each classifier, we then select the ten top-performing configurations, based on their F1-Score. These configurations, along with their F1-Scores, can be found in Appendix A3. Despite evaluating the performance of our configurations using the performance metrics accuracy, precision, recall, and F1-Score, we choose to only use F1-Score in order to identify our best configurations. This is done as F1-Score allows us to make a trade-off between precision and recall, which are two of our other metrics. Accuracy is not used as we are not sure whether the samples in the test split are balanced (i.e. both classes are equally present).

4.3 Comparing Algorithms and Configurations

After identifying the ten top-performing configurations for each classifier, we perform yet another grid-search. This time, we evaluate their performance over ten different train-test splits. We do this to reduce the effects of randomness on our outcome. Grid-search is conducted for all hyper-parameter settings that were part of a top-performing configuration in our previous grid-search. This means we also test configurations that were not part of our top ten.

This grid-search returns the optimal configurations we have tested so far. These configurations, along with their average F1-Score, are found in Table 4.8 below.

Table 4.8: Best Scoring Configuration for each Algorithm Over Ten Random States

Algorithm	**Parameters**	**F1-Score**
Support Vector Machine	C = 0.15	Average: 0.88872
	Kernel = rbf	Worst: 0.84956
	Gamma = scale	Best: 0.91667
	Augmentation = 0.025	Spread: 0.06711
Random Forests	#Trees = 250	Average: 0.88713
	Max Depth = 50	Worst: 0.84211
	Augmentation = 0.035	Best: 0.92913
		Spread: 0.08702
Multilayer Perceptron	Solver = lbfgs	Average: 0.88332
	Alpha = 0.1	Worst: 0.85938
	Layers: 2 (100, 10)	Best: 0.91729
	Augmentation = 0.035	Spread: 0.07518
Logistic Regression	C = 0.4	Average: 0.88412
	Augmentation = 0.02	Worst: 0.82883
		Best: 0.91057
		Spread: 0.08174
AdaBoost	Estimators = 50	Average: 0.88511
	Learning Rate = 0.6	Worst: 0.84211
	Augmentation = 0.025	Best: 0.92683
		Spread: 0.08472

4.3.1 Comparison of Algorithms and the Effect of Augmentation

Now that we identified the top-performing configuration for each of our algorithms, we visualise the performance of each of our algorithms using these configurations. We display the performance using the configurations as shown in Table 4.8, as well as the performance for the same configuration without augmentation (meaning only the original training data are used). A figure for each of our metrics is provided below. Accuracy is shown in Figure 4.1, precision is depicted in Figure 4.2, recall is in Figure 4.3, and F1-Score is displayed in Figure 4.4.

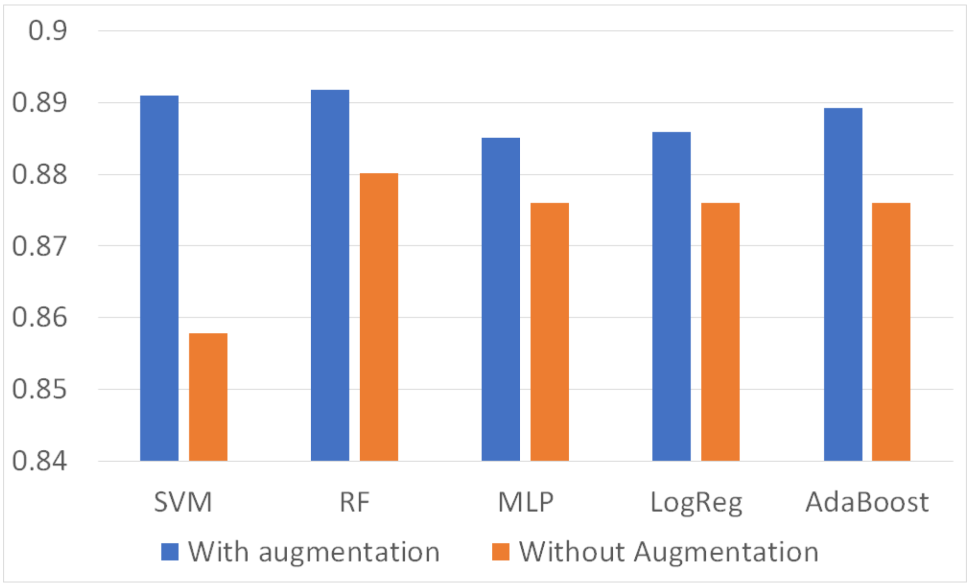

Figure 4.1: Comparison of Accuracy for all Algorithms

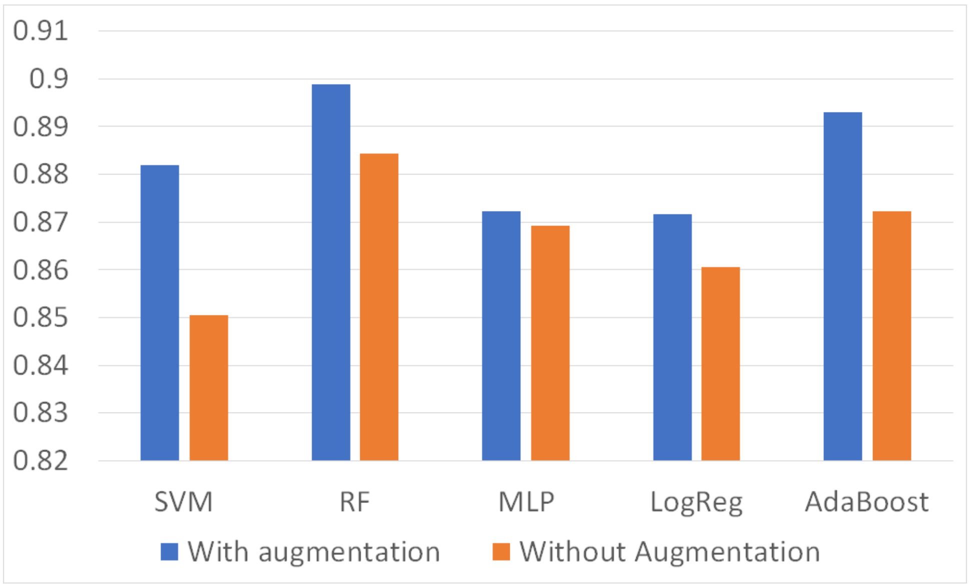

Figure 4.2: Comparison of Precision for all Algorithms

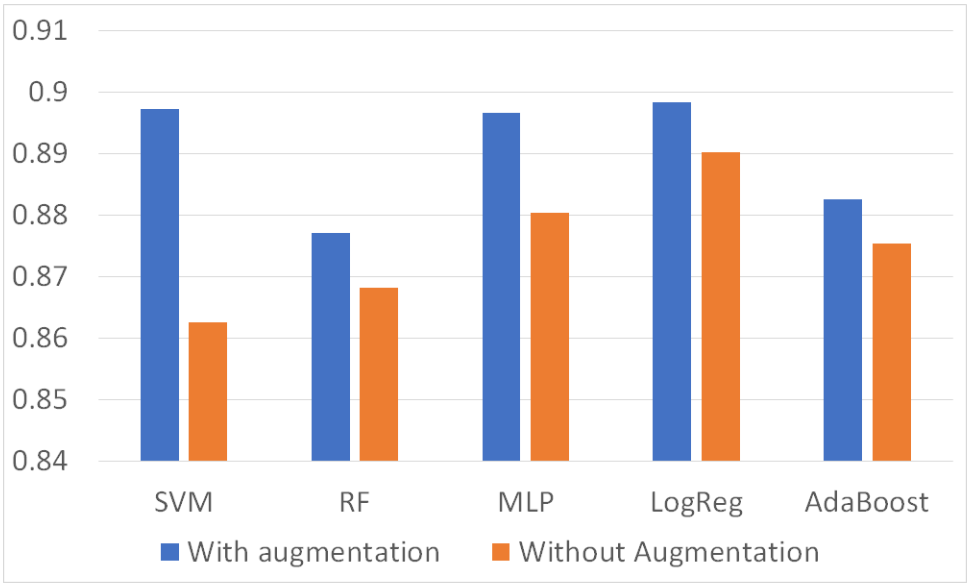

Figure 4.3: Comparison of Recall for all Algorithms

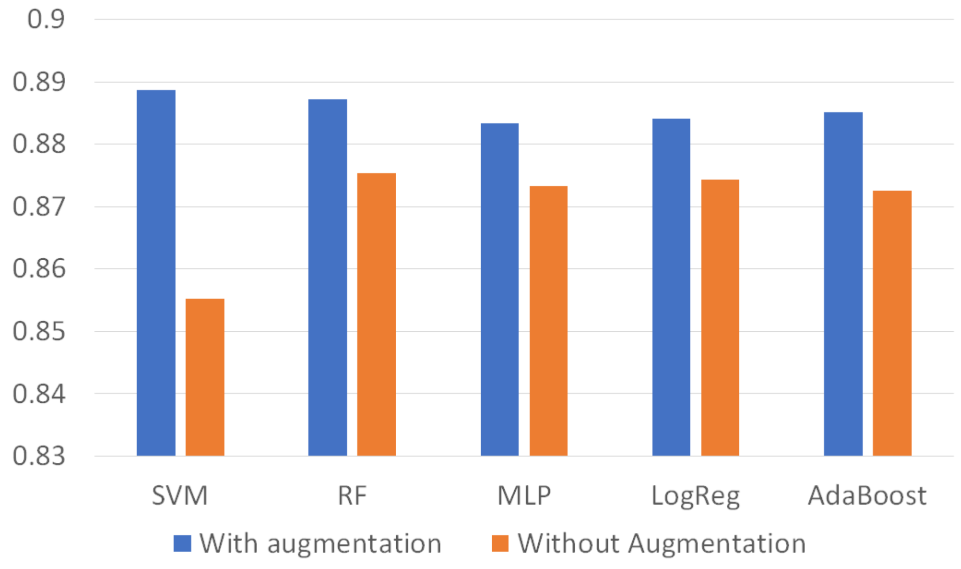

Figure 4.4: Comparison of F1-Score for all Algorithms

Looking at these figures, two main things stand out. Firstly, the use of augmentation yields better performance for all of our algorithms. Secondly, the SVM scores the lowest on all metrics when augmentation is not used, while it performs best on two out of four metrics when augmentation is used. In our runs, the random forests algorithm attained a higher accuracy than the SVM, both random forests and adaptive boosting algorithms attained a higher precision score than the SVM, and the highest recall score is returned by the logistic regression algorithm.

4.3.2 The Five Best Models

As mentioned before, we select the best classifiers based on their F1-Scores. In Table 4.9 below, our five top-performing classifiers are shown.

Table 4.9: Best Scoring Configurations Over Ten Random States

Configuration	Parameters	Average F1-Score

1. Support Vector Machine	C = 0.15
	Kernel = rbf
	Gamma = scale
	Augmentation = 0.025
		0.88872

2. Support Vector Machine	C = 0.15
	Kernel = rbf
	Gamma = scale
	Augmentation = 0.045
		0.887891

3. Support Vector Machine	C = 0.15
	Kernel = rbf
	Gamma = scale
	Augmentation = 0.02
		0.887831

4. Support Vector Machine	C = 0.15
	Kernel = rbf
	Gamma = scale
	Augmentation = 0.03
		0.887831

5. Support Vector Machine	C = 0.15
	Kernel = rbf
	Gamma = scale
	Augmentation = 0.035
		0.887831

As can be seen in this table, the five best scoring configurations were all support vector machines with parameters $Kernel = rbf$ and $C = 0.15$. We provide a plot that shows the effects of the parameters C and $Kernel$ in Figure 4.5 below. The experiments have been conducted using an augmentation factor of 0.025 as this is the best value we found so far. We investigate more values of the augmentation factor in the next section.

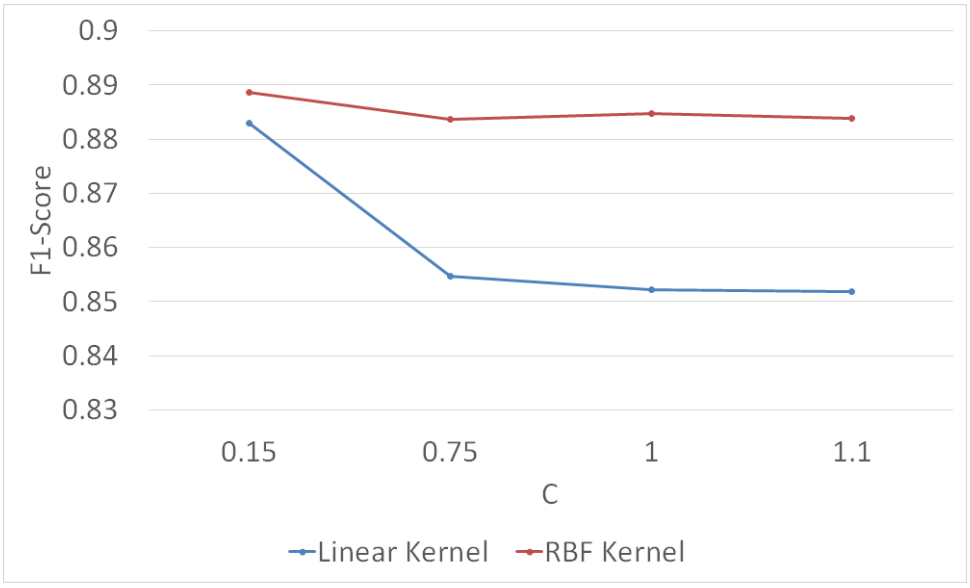

Figure 4.5: Effects of Parameters C and Kernel on F1-Score

4.4 Hyper-Parameter Optimisation

As shown in Table 4.9, the Support Vector Machine with $Kernel = rbf$, and $C = 0.15$ yielded the best results. To get the most out of the algorithm, we optimise its hyper-parameters. The main parameters to optimize are C, and our augmentation factor. We estimated the optimal value for C to be about 0.15. According to our prior experiments, the optimal augmentation factor is in the range of $[0.02, 0.045]$. Performing a final grid-search returned the optimal configuration of $C = 0.1725$. We provide the results for the final grid-search in Figure 4.6 below. Running the same experiments yielded the optimal augmentation factor to be 0.0225.

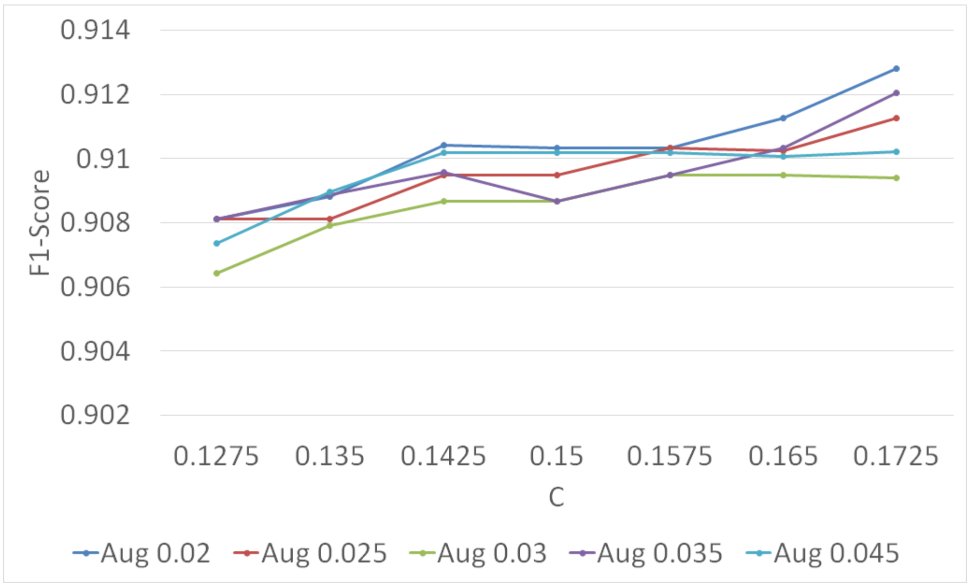

Figure 4.6: Optimisation of C and Augmentation Factor for SVM, Kernel = RBF

5 Experiments

In this chapter, we present the performance of our optimal solution to the problem of phishing detection. We evaluate accuracy, precision, recall, and F1-Score of the model. For the best and the worst random states, we also provide the confusion matrix, and precision-recall and receiver operating characteristic curves for various thresholds.

5.1 Performance of the Best Classifier

We trained and tested our optimal classifier over ten random states. After training and testing our classifier, we evaluate its performance by using our performance metrics as explained in Section 3.6 and Appendix A5. In the following sections, we provide our results along with a brief explanation on what these results mean. A more in-depth discussion is found in Chapter 6.

5.1.1 Accuracy, Precision, Recall, and F1-Score

We start off by presenting the accuracy, precision, recall, and F1-Score of our model. As mentioned, we evaluated performance of the model over ten random states. Results for all of these random states can be found in Table 5.1 below. In the following table, Table 5.2, we provide a summary of the ten runs. In this summary, we show the performance for the best random state, the worst random state, and lastly, the average performance over all ten random states.

Table 5.1: Model Performance

Random State	**Accuracy**	**Precision**	**Recall**	**F1-Score**
30	0.9256	0.9677	0.8955	0.9302
31	0.9091	0.9180	0.9032	0.9132
32	0.9091	0.9455	0.8667	0.9043
33	0.8512	0.8507	0.8769	0.8636
34	0.9091	0.8983	0.9138	0.9060
35	0.8843	0.9000	0.9000	0.9000
36	0.9008	0.9138	0.8833	0.8983
37	0.8760	0.8689	0.8833	0.8760
38	0.9339	0.9459	0.9459	0.9459
39	0.9256	0.9000	0.9474	0.9231

Table 5.2: Best, Worst, and Average Random State Performance for Top Configuration

Case	Accuracy	Precision	Recall	F1-Score
Best	0.9339	0.9459	0.9459	0.9459
Worst	0.8512	0.8507	0.8769	0.8636
Average	0.9025	0.9109	0.9016	0.9058

5.1.2 Confusion Matrices

Mathematically, these numbers are very useful. We can easily compare the performance of our model over different runs by simply comparing the values of these metrics. Intuitively, though, these numbers do not provide us with much information. To get an idea of how our model performs, we make use of confusion matrices. In these confusion matrices, we can see how the algorithm makes its predictions. Four our optimal classifier, we provide the confusion matrices for both the best and worst random state we tested. In Table 5.3, results for the best random state are shown. In Table 5.4, we do the same thing for the worst random state.

Table 5.3: Confusion Matrix for Best Random State

		Predicted	
		Positive (1)	Negative (0)
Actual	Positive (1)	43	4
	Negative (0)	4	70

As can be seen in Table 5.3 above, our classifier correctly identified 43 phishing emails as phishing emails. 70 benign emails were correctly classified as being benign emails. As for the incorrect predictions, it classified 4 benign emails as phishing emails and vice-versa.

Table 5.4: Confusion Matrix for Worst Random State

		Predicted	
		Positive (1)	Negative (0)
Actual	Positive (1)	46	10
	Negative (0)	8	57

In Table 5.4 above, we can see that, for our worst random state, the model correctly identified 46 phishing emails as being phishing emails. 57 benign spam emails were also correctly labelled. We can see that, for this random state, the model also made 18 incorrect predictions. Out of these 18 emails, 8 were benign emails the model identified as phishing emails, and 10 phishing emails were classified as benign spam.

5.1.3 Precision-Recall Curves

Precision-Recall curves are where it starts to become interesting. In Precision-Recall curves, we evaluate performance of the model for various threshold values. We start of with a very high threshold value, meaning the model only assigns the label "phishing" to emails that are almost certainly phishing emails. In doing so, precision is very high, as all emails it identifies as phishing emails will likely be phishing emails. As we lower the threshold value, two things happen. Firstly, Recall starts going up. As more emails are classified as being phishing emails, we are certain that a greater portion of the phishing emails are identified. Secondly, overall precision values start declining, though some increases in between may be present. These small increases occur when samples that score close to the threshold are miss-classified for high threshold values, while a lower value yields correct predictions.

In Figure 5.1 below, the Precision-Recall curve for the best random state is shown. In the following figure, Figure 5.2, we do the same for our worst random state.

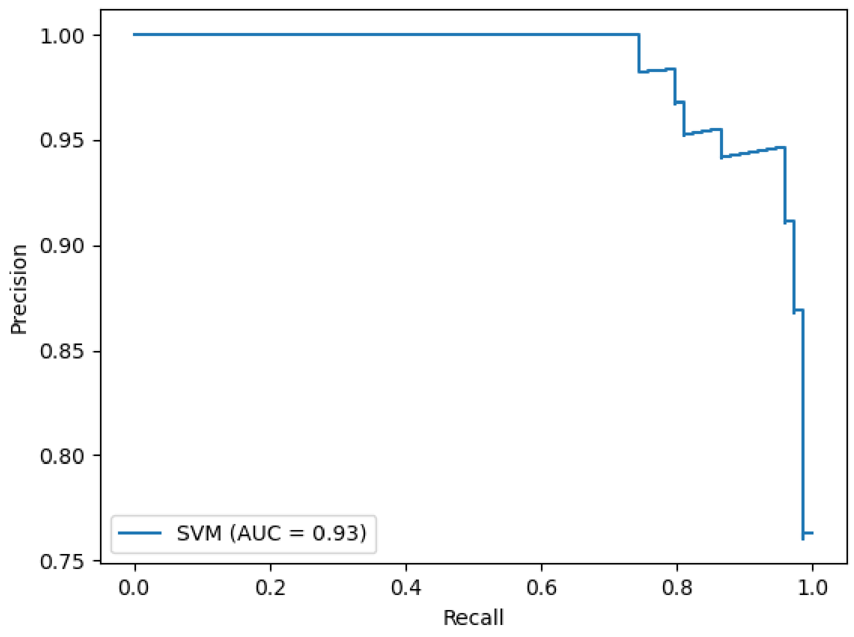

Figure 5.1: Precision-Recall Curve for Best Random State

In Figure 5.1 above, we can see that, for the best random state, our model is able to get a fairly high recall of about 0.75, after which lowering the threshold will gradually result in decreasing overall precision values (again, some small increases are present). At recall = 1, the threshold is set to such a value that precision is a little higher than 0.75.

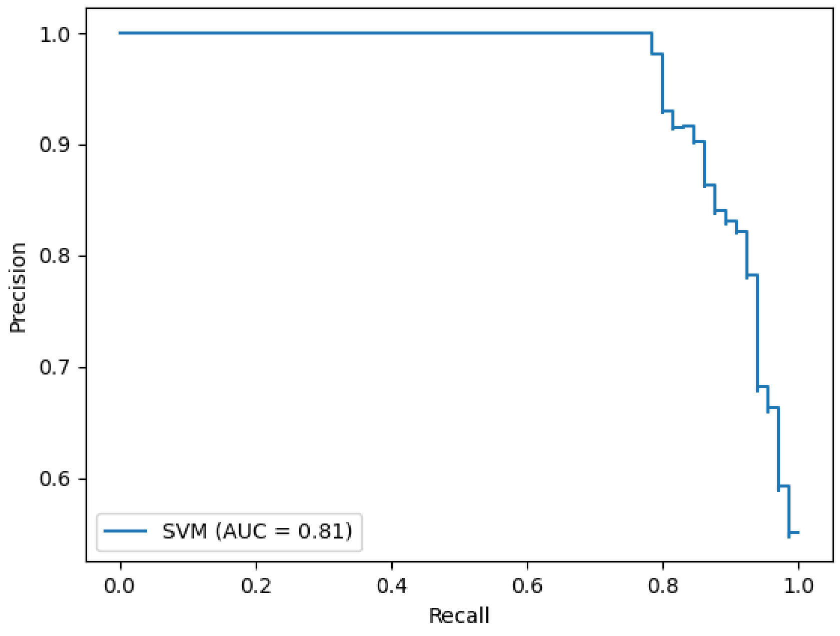

Figure 5.2: Precision-Recall Curve for Worst Random State

In Figure 5.2 above, we can see that, for the worst random state, just like for the best random state, the classifier can get a fairly high recall of 0.75 at precision = 1. Once we start further lowering the threshold values, however, the model experiences a rapid decline in performance. Some small increases in precision are present but very small. For the worst random state, precision is about 0.55 at recall = 1.

5.1.4 Receiver Operating Characteristics

Just like we provided Precision-Recall curves for our model, we provide Receiver Operating Characteristic curves. The curve for our best random state is found in Figure 5.3 below. The figure for our worst random state is found in Figure 5.4.

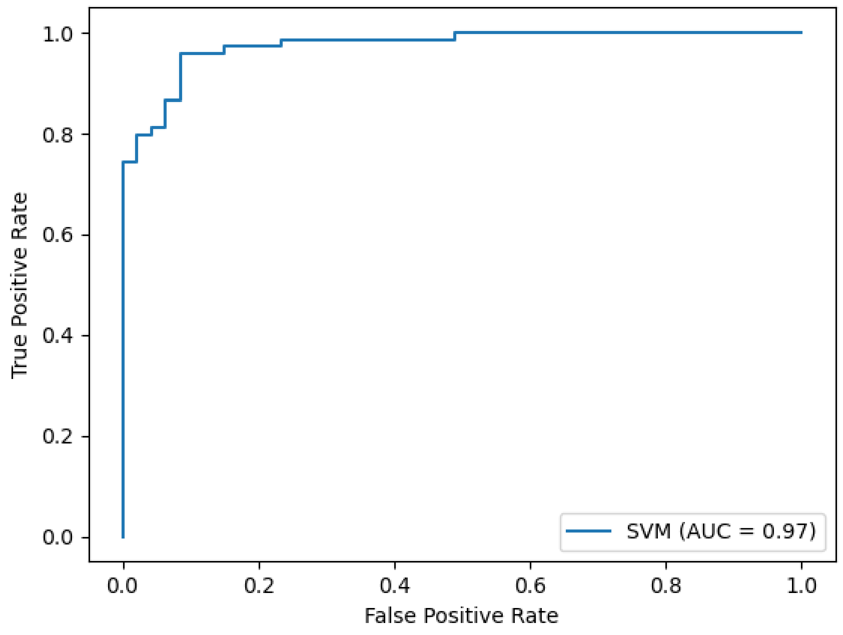

Figure 5.3: Receiver Operating Characteristic for Best Random State

In Figure 5.3 above, we can see the trade-off between true positive rate and false positive rate for various threshold values. We see how an increase in true positive rate leads to an increase in false positives. Overall, for true positive rates between about 0.75 and 0.95, an increase in true positives causes an approximately equal increase in false positive rates.

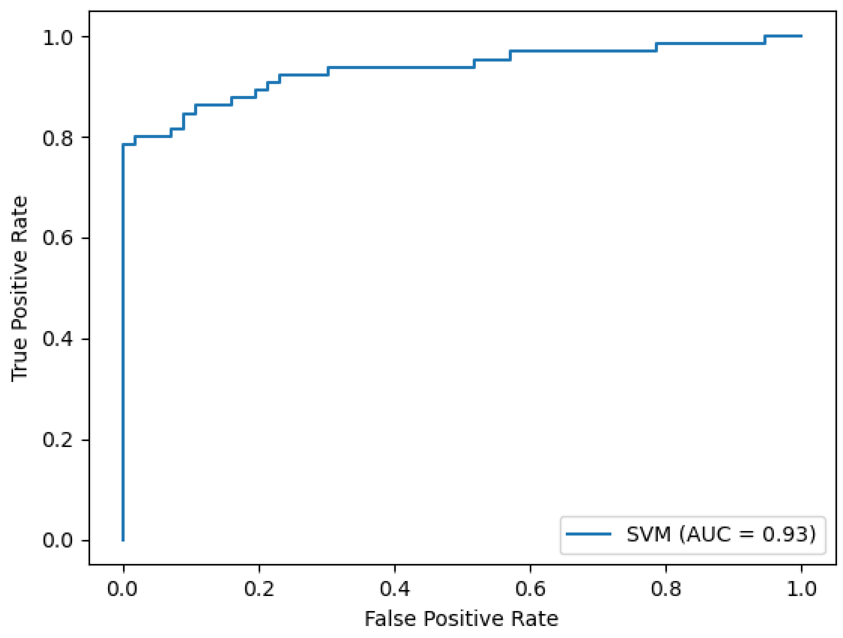

Figure 5.4: Receiver Operating Characteristic for Worst Random State

In Figure 5.4 above, the same curve is shown for our worst random state. Here, we see that increasing the true positive rate will result in major increases in false positive rates.

6 Discussion

We collected and annotated about 600 emails which we used to design, optimise, and evaluate our classification model. In this chapter, we interpret the results of our research, discuss generalisable theory and limitations, and compare our study to prior studies.

6.1 Interpretation

Our research shows that lexical features can serve as an effective input for machine learning algorithms in the problem of distinguishing phishing emails from benign spam. Using support vector machines along with pre-trained word embeddings to vectorise our emails, we create a model that can classify phishing emails from benign spam emails with good performance.

From our precision-recall curves in Section 5.1.3 follows that, up to a certain threshold, we can successfully identify a fair portion (having the value of our maximum recall at $precision = 1$, or about $\frac{3}{4}$ for both the best and worst scoring random states) of our phishing emails from the dataset without incorrectly classifying a benign spam email as being a phishing email. Attempting to identify a greater proportion of phishing emails (i.e. increasing our recall), however, will start returning false positives (i.e. a benign spam email labelled as a phishing email). Manually setting the threshold to a value such that $recall = 1$, meaning all phishing emails are identified as phishing emails, will return many false positives, whereas setting the value such that $precision = 1$ will result in many false negatives. We make a trade-off between precision and recall by using the performance metric of F1-Score. Using the F1-Score, also known as balanced F1-Score, will always set this threshold to a value such that precision and recall are approximately equal if such a value exists.

Receiver operating characteristic curves confirm this interpretation to be correct. We see that we can retrieve about 75% of phishing emails whilst still having a false positive rate of zero. Attempting to also retrieve phishing emails that are more similar to benign spam emails, causes a significant increase in false-positive rate.

Although we made use of a balanced data set (i.e. both classes have approximately the same number of samples), it is interesting to note that our best random state returned an imbalanced testing set when compared to that of our worst random state. The best state contained 47 phishing emails and 74 benign spam emails (or about 38.8% phishing), whereas the worst state contained 56 phishing emails and 65 benign spam emails (or about 46.3% phishing). Both of these sets contain a greater portion of benign spam and a lesser portion of phishing emails, meaning the training sets for these states contained a greater portion of phishing emails than benign spam emails. Though our sample size is not large enough to make any meaningful statements, we hypothesise that our classifier requires a greater amount of benign spam emails than phishing emails in order to attain maximum performance. From this hypothesis, we also hypothesise that benign spam emails have a greater lexical spread than phishing emails do (i.e. the average difference in meaning between two randomly selected benign spam emails is greater than that of two randomly selected phishing emails).

6.2 Generalisable Theory

We get back to our review question of:

How do our found methods contribute to the field of email filtering and text classification?

Our research has two main findings that are relevant to the field of email filtering and text classification. Firstly, we have shown that lexical features can be used for effectively detecting phishing emails through the use of word embeddings. We also proposed an approach to increasing sample size without the need to collect new data. This approach also reduces the impact of overfitting, which is a very significant problem for most classification algorithms, especially for the widely used support vector machine.

6.3 Limitations

There are some limitations related to our the labelling of our data set. Firstly, due to time constraints, we were unable to annotate a large enough set of emails. This is indicated by the fact that there is a significant difference in model performance between different training and testing splits. Secondly, despite the use of our labelling policy, we expect the internal consistency of our research is negatively affected by the fact that all of our samples have only been reviewed and annotated by a single individual. This problem can be solved by letting a greater number of individuals label the same set of samples, after which we can evaluate the internal consistency by using Cronbach's Alpha or similar tests on reliability. Furthermore, our research does not cover the complete set of spam emails. This is due to the fact that we did not annotate emails which we could not review for any reason. This includes emails which could not be decoded and emails that are not written in English. In addition, some types of emails are not accounted for by our model, as they may not be present in our data set.

Another approach to verifying the validity of our model is the evaluation of its performance on a different data set. We are interested in the performance of our model on other email filtering problems.

6.4 Relation to Prior Research

As stated in Chapter 2.3, our work is most closely related to "Work hard, play hard" by Alkhereyf and Rambow (2020), where word embeddings are used in combination with both the support vector machine and the extra-trees algorithm to make a distinction between business emails and personal emails. Just like in our research, the support vector machine yielded best performance in classifying emails. It is interesting to note that Alkhereyf and Rambow made use of network features and meta-info in addition to lexical features. In their research, the addition of these features was able to improve the performance of the models in email classification, and as our problem was similar to theirs, we hypothesise the use of these features will be beneficial in our case as well.

The comparison starts to become interesting when we look back at Chapter 4.3.1, where we compared the performance of various algorithms, both in cases where augmentation was used and where it was not used. As we stated, the performance of the support vector machine is greatly enhanced by the use of data augmentation, which reduces overfitting by increasing data diversity and sample size. Comparing our approach to that of Alkhereyf and Rambow, we can see that, despite the fact that we only label about 600 emails, whereas their paper annotates over 5300, comparable results can be achieved. This shows the implementation of data augmentation is able to boost performance of the support vector machine in a similar way as annotating more samples does, which is great as collecting and annotating data is very costly.

7 Conclusions

To conclude, we provide a brief recap of the results of our studies. We provide a short summary of the answers to our research questions, after which we discuss implications of our research. Furthermore, we propose some ideas for further work.

7.1 Research Questions Revisited

In this section, we revisit our research questions. For clarity, we first provide our main question. Then, our sub-questions are provided and answered, after which we use these answers to formulate an answer to our main question. Our review question is answered in the next section: Implications.

First, our main question:
Which machine learning methods can be used to effectively detect phishing emails?

Now, our three sub-questions:
SQ1: How can we prepare our data for further analysis?
As discussed in Section 3.4, we can further analyse our data after the object of interest is extracted and preprocessed. In our case, the main object of interest was the body of the email. After extraction of the body, we vectorise the data using the GloVe model for word embeddings as described in Chapter 4. These vectors can be used as an input for our model. In addition to using the vectors that vectorisation of our data yields, we use new data that we create from these vectors. This is done using a method we call data augmentation. The method is explained in detail in Section 4.1.

SQ2: What are the most effective classification algorithms for our problem?
Our research uses lexical features in the form of word embeddings. For this approach, the support vector machine yields the best overall performance, based on F1-Score. The highest recall was returned by the logistic regression algorithm, while the random forests algorithm yielded the highest precision. The support vector machine was not the best classifier on either precision or recall, but it finds a good balance between these two. Overall, the support vector machine is the most effective algorithm.

SQ3: How can we optimise hyper-parameters in our model to attain high performance?
Hyper-parameters in our models are heavily interdependent, meaning we cannot optimise them one by one. Instead, we evaluate the performance of a wide range of hyper-parameter settings using grid-search. An analysis of the results of this grid-search provides us with an indication of which hyper-parameters are important, and what their values should approximately be. Using this information, we can repeat this process for values that are close to the optimal solution. Once we are close to the optimal solution, the improvement in performance will no longer be significant. At this point, we consider the hyper-parameters to be set optimally.

Two things are important to note. First, using this method may not lead to the absolute best solution we can find. Instead, it leads us to a local optimum, meaning we either found the best solution (i.e. the global optimum), or we found the best solution contained by our parameter space and the global optimum uses parameter values we did not evaluate. Second, optimising beyond the point where performance improvement is very small might seem attractive, but this does not improve the model in any way, shape, or form. While a 0.1% improvement sounds good, tuning the parameters by very small amounts will likely only result in a better fit to the training and testing data used while performance of the model on new data will not be affected by this change.

7.1.1 Main Research Question

We go back to our main research question: *Which machine learning methods can be used to effectively detect phishing emails?*.
Combining the answers to our three sub-questions, we come to the conclusion that we can effectively detect phishing emails using word embeddings in combination with the support vector machine. To do this, we extract the body of the email, then vectorise it using word embeddings, and lastly, use the support vector machine to classify the emails based on this corresponding vector.

7.2 Implications

To present the implications of our research, we answer our review question: *How do our found methods contribute to the field of email filtering and text classification?*.

Our research has two important implications. Firstly, we can successfully classify emails based on lexical semantics by using word embeddings. Secondly, generating new samples from our actual samples is very effective in improving model performance without collecting and annotating new data.

7.3 Recommendations

Although we have demonstrated the use of word vectors for phishing detection is effective, the approach has not been polished. In further research, the performance of different sets of word vectors needs to be evaluated as our used model may not be optimal. Experiments on the number of dimensions and experiments on the size of the training data should be evaluated. In addition to the GloVe model, the fastText and Word2Vec models can be evaluated.

Another route is exploring the effects of adding more features to our input space. Like in "Work hard, play hard" (Alkhereyf and Rambow, 2020), taking a hybrid approach (i.e. combining lexical and descriptive features) may have positive effects. In addition, synthesising new features from our current features may be possible. Examples of features that may have positive effects on model performance are the angle between email vectors and the unit vector, and the magnitude of our email vector.

Furthermore, experimentation on inclusion and exclusion of certain word types may yield positive results. By using named entity recognition and part of speech tagging to assign different weights to different word types in vectorising our emails, we can more accurately represent the way language is meant to be processed. We hypothesise that nouns and verbs are highly important, while conjunctions are likely of lesser relevance.

References

Abu-Nimeh, S., Nappa, D., Wang, X., and Nair, S. (2007). A comparison of machine learning techniques for phishing detection. In *ACM International Conference Proceeding Series*, volume 269, pages 60–69.

Agrawal, S. (2019). What the heck is Word Embedding.

Alkhereyf, S. and Rambow, O. (2020). Work hard, play hard: Email classification on the avocado and enron corpora. In *Proceedings of TextGraphs@ACL 2017: The 11th Workshop on Graph-Based Methods for Natural Language Processing*, pages 57–65.

Alto, V. (2019). Neural Networks: parameters, hyperparameters and optimization strategies.

Bishop, C. M. (2013). *Pattern Recognition and Machine Learning*. Information science and statistics. Springer (India) Private Limited.

Chandrasekaran, M. and Narayanan, K. (2006). Detection based on structural properties.

Cooke, A., Smith, D., and Booth, A. (2012). Beyond PICO: The SPIDER tool for qualitative evidence synthesis. *Qualitative Health Research*, 22(10):1435–1443.

Fette, I., Sadeh, N., and Tomasic, A. (2007). Learning to detect phishing emails. In *16th International World Wide Web Conference, WWW2007*, pages 649–656.

Freund, Y. and Schapire, R. (1995). *A decision-theoretic generalization of on-line learning and an application to boosting*, volume 904.

Freund, Y. and Schapire, R. E. (1999). Large Margin Classification Using the Perceptron Algorithm. *Machine Learning*, 37(3):277–296.

Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, 63(1):3–42.

Hao, J. and Ho, T. (2019). Machine Learning Made Easy: A Review of Scikit-learn Package in Python Programming Language. *Journal of Educational and Behavioral Statistics*, 44(3):348–361.

Hevner, A., March, S., Park, J., and Ram, S. (2004). Design science in information systems research. *MIS Quarterly: Management Information Systems*, 28(1):75–105.

Huilgol, P. (2019). Accuracy vs. F1-Score.

Karani, D. (2018). Introduction to Word Embedding and Word2Vec.

Kelley, C., Hong, K., Mayhorn, C., and Murphy-Hill, E. (2012). Something smells phishy: Exploring definitions, consequences, and reactions to phishing. In *Proceedings of the Human Factors and Ergonomics Society*, pages 2108–2112.

Khonji, M., Iraqi, Y., and Jones, A. (2013). Phishing detection: A literature survey. *IEEE Communications Surveys and Tutorials*, 15(4):2091–2121.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E. (1995). Robocup: The robot world cup initiative.

Lastdrager, E. (2018). *From fishing to phishing.* PhD thesis, University of Twente, Netherlands.

Martin, S., Nelson, B., Sewani, A., Chen, K., and Joseph, A. D. (2005). Analyzing behavioral features for email classification. In *CEAS*.

Obumneme Dukor, S. (2018). Neural Representation of AND, OR, NOT, XOR and XNOR Logic Gates (Perceptron Algorithm).

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*, 12:2825–2830.

Pennington, J., Socher, R., and Manning, C. (2014a). GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Pennington, J., Socher, R., and Manning, C. D. (2014b). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Python Software Foundation (2020). About Python.

Raschka, S., Patterson, J., and Nolet, C. (2020). Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *Information (Switzerland)*, 11(4).

Rojas, R. (2009). AdaBoost and the Super Bowl of Classifiers A Tutorial Introduction to Adaptive Boosting.

Schryen, G. (2007). *Anti-Spam Measures: Analysis and Design.* Springer Berlin Heidelberg.

Sheldon, A. (2019). Artificial Intelligence Vs Machine Learning: What's the difference?

Tin Kam Ho (1995). Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282 vol.1.

Toolan, F. and Carthy, J. (2010). Feature selection for Spam and Phishing detection. In *General Members Meeting and eCrime Researchers Summit, eCrime 2010*.

Vylomova, E., Rimell, L., Cohn, T., and Baldwin, T. (2015). Take and took, gaggle and goose, book and read: Evaluating the utility of vector differences for lexical relation learning.

Wieringa, R. (2014). *Design Science Methodology for Information Systems and Software Engineering.* Computer science. Springer Berlin Heidelberg.

Appendix

A1 Grid-Search Parameter Space

Table A1.1: Grid-Search Parameter Space

Classifier	**Parameter**	**Parameter Space**
All	Augmentation Factor	0 (No Augmentation), 0.01, 0.015, 0.02, 0.025, 0.03, 0.035, 0.04, 0.045, 0.05
SVM	C	0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.15, 0.2, 0.25, 0.5, 0.75, 1, 1.1, 1.2, 1.25, 1.3, 1.4, 1.5, 1.75, 2, 2.5, 3, 4, 5, 7.5, 10
	Kernel	Linear, Rbf
Random Forests	# Trees	100, 250, 500
	Max Depth	10, 50, 100, 200
Multilayer Perceptron	Alpha	0.1, 0.25, 0.5
	Hidden Layers: 1	Layer Size: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200, 250, 300, 350, 400, 450, 500
	Hidden Layers: 2	Layer 1 Size: 50, 100, 250, 500
		Layer 2 Size: 5, 10, 20, 50
Logistic Regression	C	0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.75, 1
AdaBoost	# Estimators	1, 5, 10, 20, 30, 40, 50, 75, 100
	Learning Rate	0.001, 0.002, 0.003, 0.004, 0.005, 0.01, 0.02, 0.03, 0.04, 0.05, 0.075, 0.1, 0.2, 0.3, 0.4, 0.05, 0.6, 0.7, 0.8, 0.9, 1, 1.25, 1.5, 1.75, 2

A2 Second Grid-Search Parameter Space

Table A2.1: Second Grid-search Parameter Space

Classifier	Parameter	Parameter Space
SVM	C	0.15, 0.5, 0.75, 1, 1.1
	Kernel	Linear, Rbf
	Augmentation Factor	0 (No Augmentation), 0.02, 0.025, 0.04, 0.035, 0.045, 0.05
Random Forests	# Trees	100, 250, 500
	Max Depth	10, 50, 100, 200
	Augmentation Factor	0 (No Augmentation), 0.01, 0.015, 0.035, 0.04
Multilayer Perceptron	Alpha	0.1, 0.25, 0.5
	Hidden Layers: 1	Layer Size: 10, 20, 40, 60
	Hidden Layers: 2	Layer 1 Size: 50, 100, 250
		Layer 2 Size: 10, 20
	Augmentation Factor	0 (No Augmentation), 0.01, 0.03, 0.035, 0.04, 0.045, 0.05
Logistic Regression	C	0.4, 0.5, 0.75, 1
	Augmentation Factor	0.01, 0.015, 0.02
AdaBoost	# Estimators	40, 50, 75, 100
	Learning Rate	0.6, 0.7, 0.8, 1, 1.5
	Augmentation Factor	0.02, 0.025, 0.03, 0.035

A3 Initial Highest Scoring Configurations

Table A3.1: Best Configurations for the Support Vector Machine

Configuration	C	Kernel	Augmentation	F1-Score
1	0.75	Rbf	0.02	0.9189
2	0.75	Rbf	0.03	0.9189
3	0.75	Rbf	0.035	0.9189
4	1	Rbf	0.035	0.9189
5	1	Rbf	0.045	0.9189
6	1.1	Rbf	0.045	0.9189
7	1	Rbf	0.05	0.9189
8	0.5	Rbf	0	0.9123
9	0.75	Rbf	0.025	0.9107
10	0.15	Linear	0.02	0.9091

Table A3.2: Best Configurations for the Random Forests Algorithm

Configuration	# Trees	Max Depth	Augmentation	F1-Score
1	100	10	0.04	0.8929
2	100	200	0.015	0.8909
3	100	50	0.035	0.8909
4	100	200	0	0.8909
5	100	100	0.01	0.8850
6	100	50	0.03	0.8950
7	100	50	0.01	0.8829
8	250	100	0.01	0.8829
9	500	50	0.01	0.8829
10	500	100	0.01	0.8829

Table A3.3: Best Configurations for Logistic Regression

Configuration	C	Augmentation	F1-Score
1	0.4	0.01	0.8850
2	0.5	0.01	0.8850
3	0.75	0.01	0.8850
4	1	0.01	0.8850
5	0.4	0.015	0.8850
6	0.5	0.015	0.8850
7	0.75	0.015	0.8850
8	1	0.015	0.8850
9	0.4	0.02	0.8850
10	0.5	0.02	0.8850

Table A3.4: Best Configurations for the Multilayer Perceptron

Configuration	Alpha	HL1 Size	HL2 Size	Augmentation	F1-Score
1	0.5	100	20	0	0.9159
2	0.25	50	10	0.03	0.9107
3	0.1	250	10	0.05	0.9107
4	0.25	20		0.01	0.9074
5	0.5	60		0.01	0.9074
6	0.5	100	10	0.01	0.9074
7	0.5	100	20	0.045	0.9074
8	0.5	10		0	0.9074
9	0.1	100	10	0.04	0.9027
10	0.5	40		0.035	0.9009

Table A3.5: Best Configurations for Adaptive Boosting

Configuration	Estimators	Learning Rate	Augmentation	F1-Score
1	50	1	0.025	0.9381
2	100	1	0.025	0.9381
3	75	1	0.025	0.9298
4	100	1.5	0.05	0.9286
5	50	1.5	0.03	0.9259
6	50	0.7	0.02	0.9217
7	40	1	0.025	0.9204
8	100	0.8	0.03	0.9204
9	40	0.7	0.035	0.9204
10	40	0.6	0.015	0.9123

A4 Labelling Policy

In order to classify phishing emails from benign spam emails, our categories "phishing emails" and "benign spam emails", need to be properly defined. In this section, we will perform a systematic literature review, which will allow us to come to a clear set of guidelines which we can use in labelling our data.

A4.1 Definition of the Research Question

The goal of this Systematic Literature Review is to come up with a clear policy we can use to distinguish phishing emails from benign spam.

In order to come up with this policy, we provide our definition of phishing. Then, we formulate a set of rules we can use to manually classify our emails.

A4.2 Inclusion and Exclusion Criteria

Not all available literature will be helpful in answering our research question. Therefore, both inclusion and exclusion criteria are defined.

A4.2.1 Inclusion Criteria

An article must provide a clear definition of phishing emails and its characteristics.

A4.2.2 Exclusion Criteria

Phishing emails are occasionally seen a subset of the set: spam emails. Some sources may use terms interchangeably, making no distinction. We will exclude any articles where this is the case.

A4.3 Used Databases

Scopus was used as our main source of articles and publications on this topic.

A4.4 Search Terms and Used Strategy

A proper search strategy is needed in order to find relevant articles. In this review, we will use the SPIDER strategy, which is a modified version of the commonly used PICO strategy (Cooke et al., 2012). The acronym SPIDER stands for: Sample, Phenomenon of Interest, Design, Evaluation, and Research type.

Breaking down our research question using this framework yields the following:

1. Sample: Individual emails
2. Phenomenon of Interest: Phishing
3. Design: Comparison study
4. Evaluation: Correctness, accuracy
5. Research type: Qualitative

Our breakdown is used to create a search matrix. This matrix is found in Table A4.1 below.

Table A4.1: Search Matrix

Constructs	**Related Terms**	**Broader Terms**	**Narrower Terms**
Phishing	Email	Junk	Spear Phishing
Detection	Filtering		
Characteristics	Properties, Features		Aspects

A4.5 Found Articles

Terms in our search matrix were used to construct queries for searching our database. Queries and the corresponding number of hits are found in Table A4.2.

Table A4.2: Found Articles

Search Query	**Number of Results**
Phishing AND email AND detection	323
Phishing AND email AND detection	203
Phishing AND email AND definition	5
Phishing AND email AND filtering	91

These queries result in a set of articles which can aid us in answering our research question. We merge our queries into the following query: "(Phishing AND email) AND (detection OR features OR definition OR filtering)". This returns 419 results. From these 419 results, we select ten articles based on their title. These ten articles are then reviewed based on their abstracts, introduction, and conclusion. We use three of those ten articles as the basis for this literature review. Analysing these articles leads us to the article: "Phishing E-Mail Detection Based on Structural Properties", which we also consider to be relevant. This brings us to a total of four articles, which we use in defining phishing and coming up with a labelling policy. An overview of used articles is provided in Table A4.3 below.

Table A4.3: Articles Used in Defining Phishing

Article	**Author(s)**
Learning to detect phishing emails	Fette et al.
Phishing detection: A literature survey	Khonji et al.
Something smells phishy: Exploring definitions, consequences, and reactions to phishing	Kelley et al.
Phishing E-Mail Detection Based on Structural Properties	Chandrasekaran and Narayanan

A4.6 Conceptual Matrix

After thoroughly reading the selected articles, we extracted the main concepts from each article. These main concepts, along with their presence in the articles, can be found in Table A4.4 below. In Table A4.5, we explain the concepts in more detail.

Table A4.4: Conceptual Matrix

	Deceptive communication	Spoofing	Generalization	Contextual lures	Information theft	Persuasion
Learning to detect phishing emails	*	*				
Phishing detection: A literature survey	*	*			*	*
Something smells phishy: Exploring definitions, consequences, and reactions to phishing	*	*		*		
Phishing E-Mail Detection Based on Structural Properties	*	*	*	*		

Table A4.5: Overview of Concepts

Concept	**Explanation**
Deceptive Communication	Deceptive communication is communication in which the victim is mislead, the truth is hidden, or a false concept is promoted.
Spoofing	Spoofing is a practice in which the criminal disguises him/herself as someone or something else. Disguising oneself as a bank or other reputable company are very common.
Generalization	In generalization, communication is done in a non-personal way. (i.e. they do not address you by name, or any other personal property)
Contextual Lures	In contextual lures, a false sense of urgency, a threat, or any other concern is used to deceive the victim. Common examples include lottery scams and the notorious Nigerian prince email.
Information Theft	Information theft is the practice where criminals send emails in order to steal personal information.
Persuasion	Persuasion is a process in which the criminal attempts to get the victim to perform actions for the attacker's benefit.

A4.7 Integration of Theory

The articles have been analysed, and a concept matrix has been made. Using these concepts, we can formulate a proper definition of phishing. Using this definition of phishing, we create a policy table for labelling our emails.

A4.7.1 Definition of Phishing

As can be seen in Table A4.4, all used articles mention deceptive communication. In essence, this is what most phishing attempts are all about. In phishing, attackers often portray themselves as well-intentioned instances or individuals. We make a distinction between two main ways of doing this.

The first method we found is known as the spoofing attack. In spoofing attacks, the attacker pretends to be part of a trusted instance, such as your bank, PayPal, or even your credit card company. They will then try to obtain your personal information, which they can then abuse for their own financial gain.

The second deceptive method is a method that makes use of the previously-defined contextual lures. In this method, the attacker often does not pretend to be part of an instance your trust. Instead, they present themselves as trustworthy individuals, usually being stuck in situations where they need your help in exchange for wealth, riches, or well-being in some other way, shape, or form. Some examples of this type of phishing attack are the notorious Nigerian prince scam, lottery scams, and overly promising business offers.

In addition to blatant deception, contextual lures are often used in a type of phishing attack we will give the name of "the blackmail". In blackmailing attacks, attackers send coercive emails to their victims. In these emails, they threat to publish revealing information about the victim, e.g. in the form of webcam recordings. Alternatively, they may threaten to inflict harm to the victim or his or her relatives. After the context is constructed, the attacker will offer to renounce the threat in exchange for compensation, usually in the form of money.

From these concepts and methods, we conclude our definition of Phishing is very similar to that in the article: Phishing detection: A literature survey. In this article, the following definition is provided: "Phishing is a type of computer attack that communicates socially engineered messages to humans via electronic communication channels in order to persuade them to perform certain actions for the attacker's benefit." (Khonji et al., 2013). We use this definition, along with the methods described above, as a basis for our labelling policy.

A4.7.2 Labelling Policy

We designed a policy that is used in labelling our emails. Emails can only be assigned one label. Our policy table is found in Table A4.6 below.

Table A4.6: Labelling Policy for Phishing Emails

Situation	**Label**
A third party falsely claims to be part of a trusted institution	Phishing
A third party uses contextual lures (e.g. lottery scams)	Phishing
A third party attempts to persuade the receiver of the email	Phishing
A third party threatens the receiver of the email	Phishing
A third party attempts to obtain personal information	Phishing
None of the above	Benign spam

A4.8 Some Final Notes on Labelling

Despite providing a clear definition of phishing, as well as a policy table that is used in labelling our emails, we still encountered some situations in which our policy table falls short, and we have to make a judgement based on our instinct. These situations include some types of loan offers, some types of advertisements, as well as emails where tricks are used to bypass some spam filters. An example of such a trick is to convert the text of the email to an image or other object, which we decided not to open for the sake of our own security. Additionally, some emails in our data set could not be decoded, are not written in English, or could not be reviewed for any other reason. These emails, as well as the emails where we could not make a decision based on our policy, have been skipped in the labelling process. We further discuss skipping emails in Section 6.3.

A5 Performance Metrics

Performance metrics are variables that we can use to express the performance of a system in a real number. This is done so we can compare different systems and models, which allows us to choose the best option available to use. Two widely used performance metrics are accuracy and F1-Score. These two metrics are explained and compared in this appendix. We use the article: "Accuracy vs. F1-Score" as a basis (Huilgol, 2019).

We make a distinction between four different situations, being the True Positive (TP), False Positive (FP), False Negative (FN), and the True Negative (TN). "True" labels TP and TN indicate a correct prediction, whereas "False" labels FP and FN indicate a false prediction. "Positive" labels TP and FP indicate the model predicted the investigated phenomenon to be present, whereas the "Negative" labels indicate absence. We provide a matrix, known as a "Confusion Matrix" for clarification. This matrix is found in Table A5.1 below.

Table A5.1: Confusion Matrix

		Predicted	
		Positive (1)	Negative (0)
Actual	Positive (1)	True Positive (TP)	False Negative (FN)
	Negative (0)	False Positive (FP)	True Negative (TN)

A5.1 Accuracy

Our first metric, accuracy, is the simplest metric available to us. It expresses the number of correct predictions as a fraction of the total number of predictions. In other words, it indicates the proportion of items the model got correct. It is evaluated by Equation (.1). Accuracy serves as a good performance metric in cases where all instances are equally important and classes are balanced (i.e. both classes are equally present in the data set).

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{.1}$$

A5.2 Precision

A slightly more advanced performance metric is precision. Precision indicates the correct positive cases as a fraction of all predicted positives. This metric only penalizes false positives, meaning false negatives and true negatives have no effect. Precision is evaluated by Equation (.2).

$$Precision = \frac{TP}{TP + FP} \tag{.2}$$

A5.3 Recall

In addition to precision, we use a similar metric known as recall. Recall indicates the correct positives as a fraction of all positive cases. This metric only penalizes false negatives, meaning it is not affected by false positives and true negatives. Recall is evaluated by Equation (.3).

$$Recall = \frac{TP}{TP + FN} \tag{.3}$$

A5.4 F_β-Score

We can make a trade-off between precision and recall by using the F_β-Score metric. F_β-Score evaluates the weighted harmonic mean between precision and recall, such that recall is β times as important as precision. This metric expresses performance as a real number between 0 and 1, where a higher score indicates a better performance. In problems where recall is more important than precision, a high value of β is used. Due to recall being β times as important as precision, precision is $\frac{1}{\beta}$ times as important as recall. F_β-Score is evaluated by Equation .4.

$$F_\beta = (1 + \beta) * \frac{precision * recall}{(\beta^2 * precision) + recall} \tag{.4}$$

A5.4.1 F1-Score

The most traditional version of the F_β-Score is the F1-Score. In the F1-Score, precision and recall are equally important, meaning false positives and false negatives are assigned an equal penalty. Substituting $\beta = 1$ in Equation .4 yields Equation (.5) after simplification.

$$F1 - Score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{.5}$$

Our research uses the F1-Score as opposed to the more general F_β-Score as the $\beta = 1$ implementation of the F_β-Score is most widely used, and we did not encounter a situation where deviating from this standard was deemed necessary.

A5.5 Precision-Recall and ROC Curves

In classifying new samples, classification models express a sample as a single value. This value is then compared to a tune-able parameter called threshold, denoted by λ. In cases where the value of the sample is lower than the model threshold, the model classifies the sample as belonging to class A. In cases where the value of the sample is greater than or equal to the model threshold, the model returns class B for the sample. Tuning the threshold of a model is essential in optimising the model as this allows us to influence when the decision of our model changes.

A5.5.1 Precision-Recall Curves

Precision-recall curves show the tradeoff between precision and recall. This can be used to gain insights into where our model starts to encounter difficulties during classification.

In addition to the curve itself, we can also use the area under the curve as a performance metric. A high area under the curve represents both high recall and high precision, which shows that the classifier is able to separate our classes very well.

An example of a precision-recall curve is found in Figure A5.1 below.

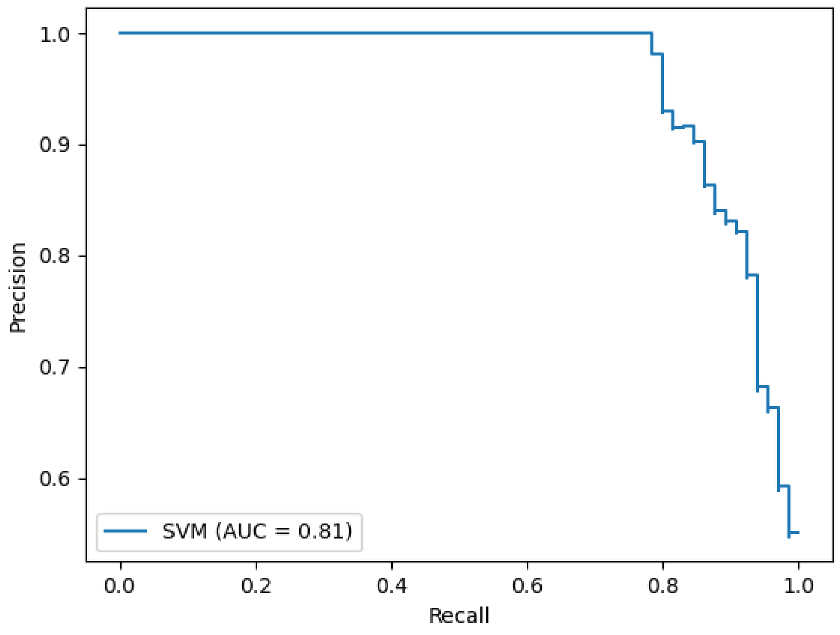

Figure A5.1: Example of a Precision-Recall Curve

A5.5.2 Receiver Operating Characteristics

Receiver operating characteristic curves, commonly referred to as ROC curves, serve a similar purpose as precision-recall curves. Whereas precision-recall curves show the tradeoff between precision and recall, ROC curves show the tradeoff between true positive rate and false positive rate. In addition to the curve itself, we can use the area under the curve as a performance metric, where a high area under the curve represents a model that can classify our samples well.

An example of a receiver operating characteristics curve is found in Figure A5.2 below.

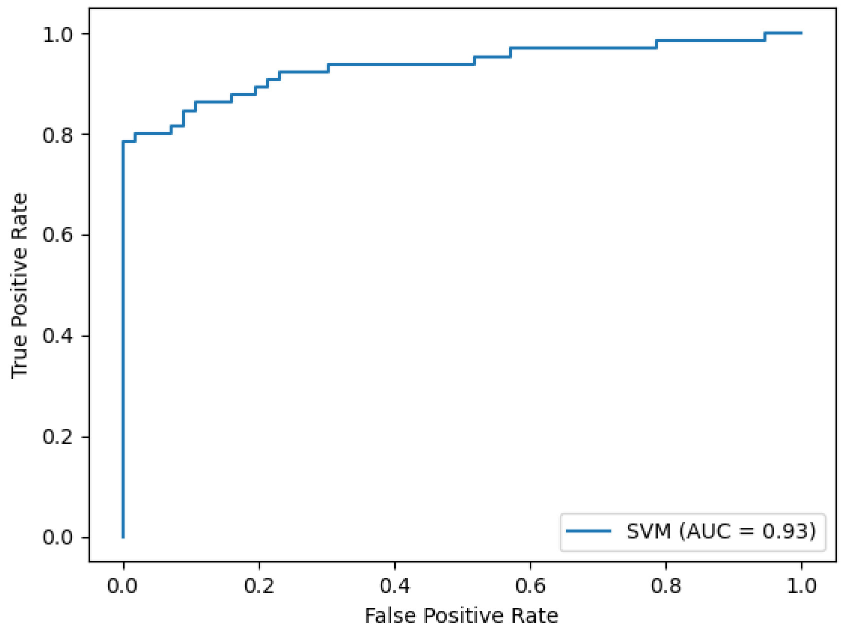

Figure A5.2: Example of a Receiver Operating Characteristic Curve