**UNIVERSITY OF SOUTHAMPTON**
Faculty of Engineering and Physical Sciences
School of Electronics and Computer Science

A project report submitted for the award of
BSc Computer Science

Supervisor: Dr Enrico Gerding
Examiner: Dr Age Chapman

# Machine Learning to Detect Malicious URLs

by Edward J. Gorman

May 12, 2020

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

A project report submitted for the award of BSc Computer Science

by Edward J. Gorman

Malicious URLs are designed to perform a range of cybercriminal activities on unsuspecting users from installing malware to misappropriating personal data. Their presence is a growing problem in the cyber landscape and is a potential threat to everyone. Blacklisting alone is an insufficient method to assuage this ever-growing threat; improved methods of detection are required.

This report provides a comprehensive analysis for the development of a malicious URL detection system using machine learning by providing a holistic overview of previously researched and novel techniques to determine the most effective approach. The methodologies drew upon herein include numerous feature representations (i.e. lexical, host, and content properties) and model configurations (i.e. shallow and deep learning algorithms).

Analysis of the content-based feature representation set performed best relative to other approaches, scoring recall values of 0.995, 0.911, and 0.719 for the SVM, RF, and Pc algorithms, respectively. AUC-ROC curve analysis revealed that the RF algorithm is routinely the best performing model achieving the highest AUC scores for each feature representation, particularly for content-based techniques where an AUC value of 0.95 was observed. The RF algorithm surpassed both online and deep learning techniques and is the recommended approach for the detection of malicious URLs.

## Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

***You must change the statements in the boxes if you do not agree with them.***

We expect you to acknowledge all sources of information (e.g. ideas, algorithms, data) using citations. You must also put quotation marks around any sections of text that you have copied without paraphrasing. If any figures or tables have been taken or modified from another source, you must explain this in the caption and cite the original source.

**I have acknowledged all sources, and identified any content taken from elsewhere.**

If you have used any code (e.g. open-source code), reference designs, or similar resources that have been produced by anyone else, you must list them in the box below. In the report, you must explain what was used and how it relates to the work you have done.

**I have not used any resources produced by anyone else.**

You can consult with module teaching staff/demonstrators, but you should not show anyone else your work (this includes uploading your work to publicly-accessible repositories e.g. Github, unless expressly permitted by the module leader), or help them to do theirs. For individual assignments, we expect you to work on your own. For group assignments, we expect that you work only with your allocated group. You must get permission in writing from the module teaching staff before you seek outside assistance, e.g. a proofreading service, and declare it here.

**I did all the work myself, or with my allocated group, and have not helped anyone else.**

We expect that you have not fabricated, modified or distorted any data, evidence, references, experimental results, or other material used or presented in the report. You must clearly describe your experiments and how the results were obtained, and include all data, source code and/or designs (either in the report, or submitted as a separate file) so that your results could be reproduced.

**The material in the report is genuine, and I have included all my data/code/designs.**

We expect that you have not previously submitted any part of this work for another assessment. You must get permission in writing from the module teaching staff before re-using any of your previously submitted work for this assessment.

**I have not submitted any part of this work for another assessment.**

If your work involved research/studies (including surveys) on human participants, their cells or data, or on animals, you must have been granted ethical approval before the work was carried out, and any experiments must have followed these requirements. You must give details of this in the report, and list the ethical approval reference number(s) in the box below.

**My work did not involve human participants, their cells or data, or animals.**

*ECS Statement of Originality Template, updated August 2018, Alex Weddell aiofficer@ecs.soton.ac.uk*

# Contents

# List of Figures

# List of Tables

# Acknowledgements

Thanks to Dr Enrico Gerding for his valuable and constructive suggestions during the planning and development of this research work.

I would also like to thank my parents for their always-present love, support, and encouragement.

# Chapter 1

# Introduction

Access to content on the Web is mediated by Web browsers (e.g. Chrome, Firefox) using addresses known as Uniform Resource Locators (URLs) which reference resources located anywhere in the world and encompass all types of content such as text files, applications, and multimedia. Web sites (or websites) have become the face of the Internet and are the most common way that people interact with it today. Messages exchanged with safe (i.e. benign) websites perform the actions intended by a user which may include emailing a colleague, viewing banking information, or listening to music. Some websites, however, are considered unsafe (i.e. malicious) because they host content that performs actions to harm and/or confound users. The most common types of malicious sites include:

- phishing attempts – a type of social engineering attack that steals user data or obtains access to protected systems;
- malware installations – software such as viruses or ransomware that cause damage to computer files and systems; and
- spam sites – URLs that trick users into clicking links, abuse page ranking metrics, or contain low-effort/auto-generated content.

Malicious websites have become a growing threat in the cyber landscape, affecting every type of user from individuals to businesses and governments. For phishing attacks alone, it was estimated in 2012 that damage from stolen information ranged between "$61 million to $3 billion" annually [6]. Businesses have experienced a surge in malware attacks rising by 79% between 2017 and 2018, with malware in the form of software backdoors increasing the most [12]. Malicious websites are thus a significant threat to Web users and reliable methods for their detection are required.

Modern-day Web browsers and search engines currently attempt to solve this problem by employing blacklists; i.e. databases of known malicious URLs which can be queried to see if they contain the URL of a website the user is about to access. However, blacklisting is becoming increasingly inadequate due to the large number of malicious sites generated daily, particularly those that comprise slight variations of URLs that have been blocked previously. An analysis of blacklists in 2018 found that only "11% of infected websites were blacklisted", a decrease of 6% from 2017 [19]. Also, maintaining such databases is time consuming and labour intensive, and if a blacklist becomes corrupted it could allow known malicious URLs to escape detection. It is clear that blacklisting alone is not a suitable solution for avoiding malicious URLs.

Another solution to this problem is machine learning: an application from the field of computer science enabling computer systems to perform specific tasks without explicit instructions. This approach has the ability to overcome the deficiencies of traditional blacklists by learning what features of a URL distinguish a malicious from a benign website, and generalise this knowledge such that future malicious URLs may be detected. Many research papers have previously explored this concept with the aim of assessing a single representation of a URL against a selection of similar machine learning models which are trained according to how accurately they predict malicious cases. This report differs by reviewing the full domain of possible feature representations applied to a wide range of learning algorithms and comparing these directly against each other according to their ability to capture all malicious URLs (encouraging a low false positive rate). The main drawback of this approach is that a higher proportion of benign URLs may be incorrectly classified which must be considered as the system will not be useful if it is inconsistent.

This report provides a comprehensive analysis for the development of a malicious URL detection system using machine learning by providing a holistic overview of previously researched and novel techniques to determine the most effective approach. The machine learning system is divided into the feature extraction process which generates a feature representation of a URL, and the algorithm that learns to classify URLs as malicious or benign. Feature representation is further subdivided into lexical-based, host-based and content-based properties of the URL. Machine learning algorithms are also subdivided between shallow learning and deep learning techniques. This paper begins by assessing existing approaches with regards to their feature representation and algorithm choice, followed by a detailed methodology implementing both popular and novel techniques, and concluding with a critical analysis and comparison of these systems to determine the optimal malicious URL detection approach.

# Chapter 2

# Project Management

This chapter details the approach taken to manage all aspects of the research project including planning phases, monitoring strategies, and risk mitigation plans. It details the time line of the project via a Gantt Chart (see Figure 2.1) which was established during the planning stages, as well as constraints that were placed on the project.

## 2.1 Project Scope

The purpose of the project is to develop a malicious URL detection system using machine learning and to evaluate its capacity as a substitute or replacement for the existing blacklist technology. To preserve the aim of this project and prevent scope creep the following goals were formally declared at the start of this research:

- to assess the performance of a selection of machine learning algorithms and parameters for detecting malicious URLs; and
- to determine if reliable predictions can be made using the domain string alone or if additional metadata are required.

A possible extension could involve designing a user-friendly interface in the form of a Web application wherein a user inputs a URL and the service returns the likelihood that the URL is malicious. This could potentially include a user feedback system which helps to improve the ML algorithm by training the model to improve based on results.

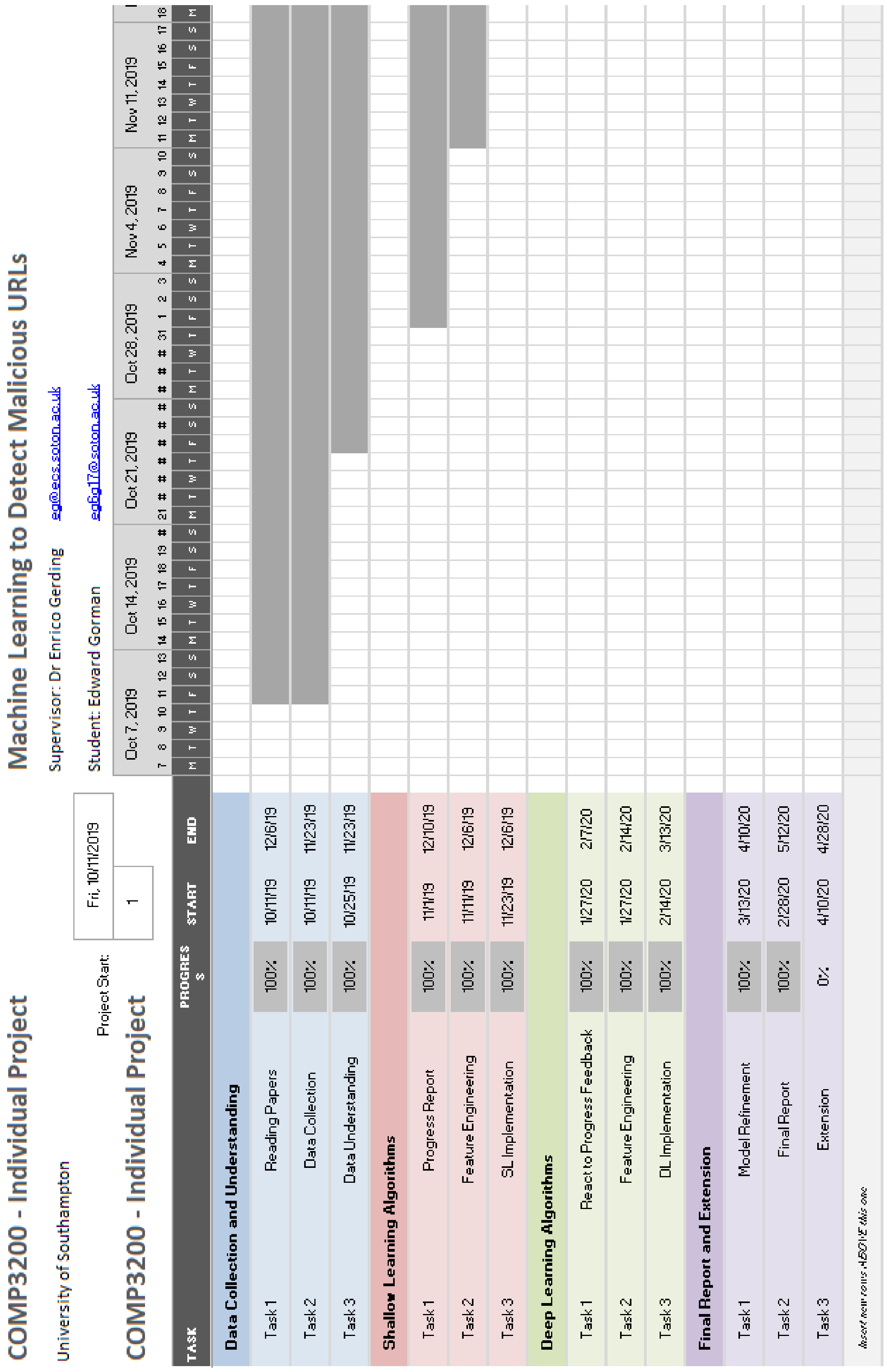
## 2.2 Time Management

The time allocated to complete this project began in September of 2019 and ended in May of 2020 with the work being spread evenly across this time period except for December/January to prepare for the first semester exam season. Approximately 12 hours per week will be spent on this project and any updates/delays that occur will be discussed and confirmed with the project supervisor.

In line with the goals of this project the work has been segmented into four phases. The first two phases are estimated to be completed by semester one and the remaining phases during semester two. The specific details of each phase are as follows:

1. Data collection and understanding
    (a) Read and evaluate similar approaches in related papers
    (b) Accumulate list of known malicious and benign URLs
    (c) Analyse data and identify any bias/constraints present
2. Shallow ML implementation
    (a) Extract lexical-based, host-based, and content-based feature representations
    (b) Implement shallow ML models and measure performance
3. Deep ML implementation
    (a) Extract char-level, word-level, and label-level data representations
    (b) Implement deep ML models and measure performance
4. ML refinement and final report
    (a) Refine models and generate performance graphics
    (b) Write up final report
    (c) Construct user-friendly interface (optional extension)

These phases are illustrated in the project Gantt Chart (see next page) which also details specific time estimates for the length of each task as well as two additional tasks related to the progress report. Progress will be monitored on a weekly basis to ensure the project is on schedule and to keep track of the upcoming steps. An evaluation of the progress made during this project and by the hand-in date is reviewed in Section 6.4.



COMP3200 - Individual Project

University of Southampton

Project Start: Fri, 10/11/2019

1

COMP3200 - Individual Project

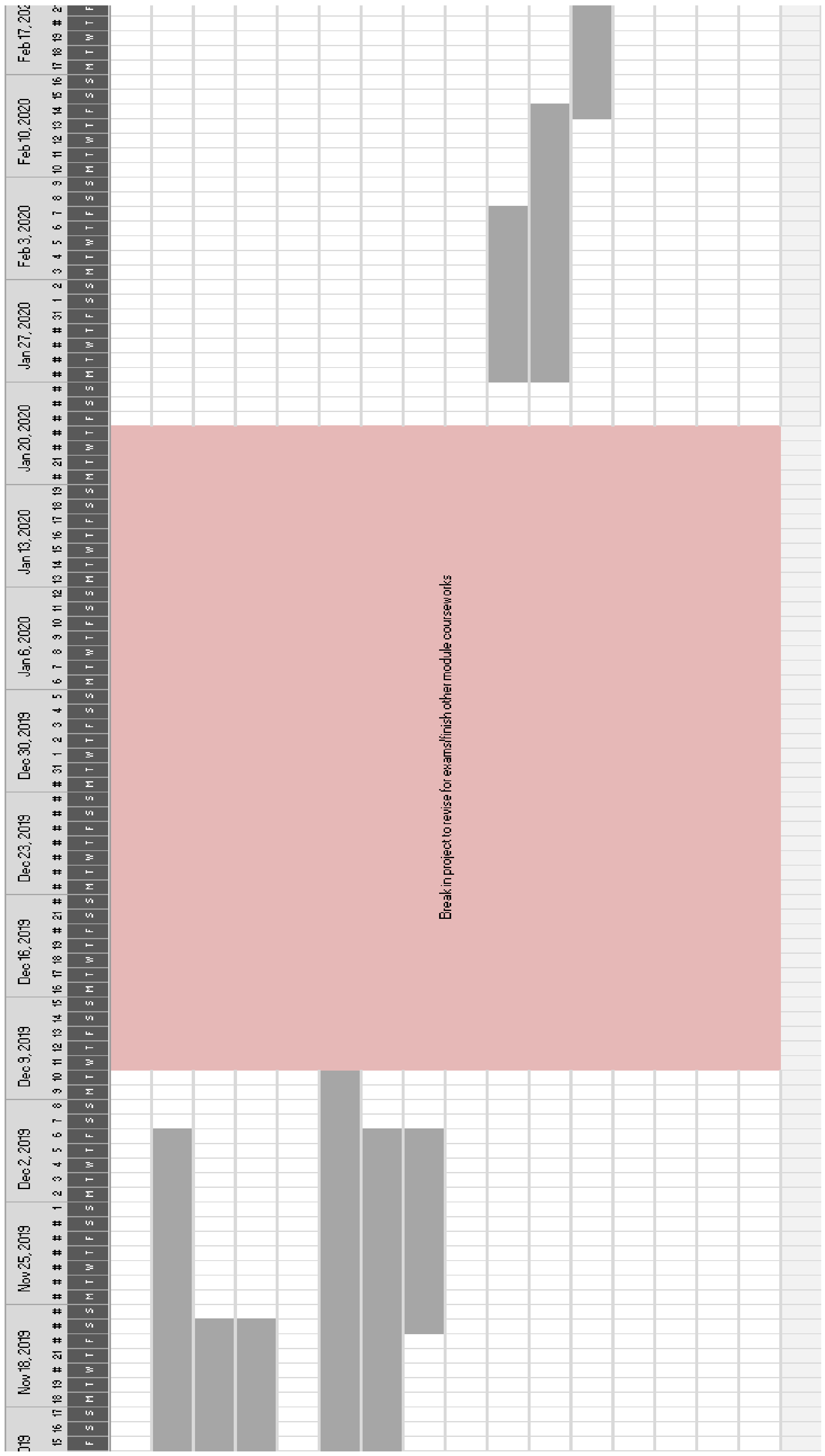
Machine Learning to Detect Malicious URLs

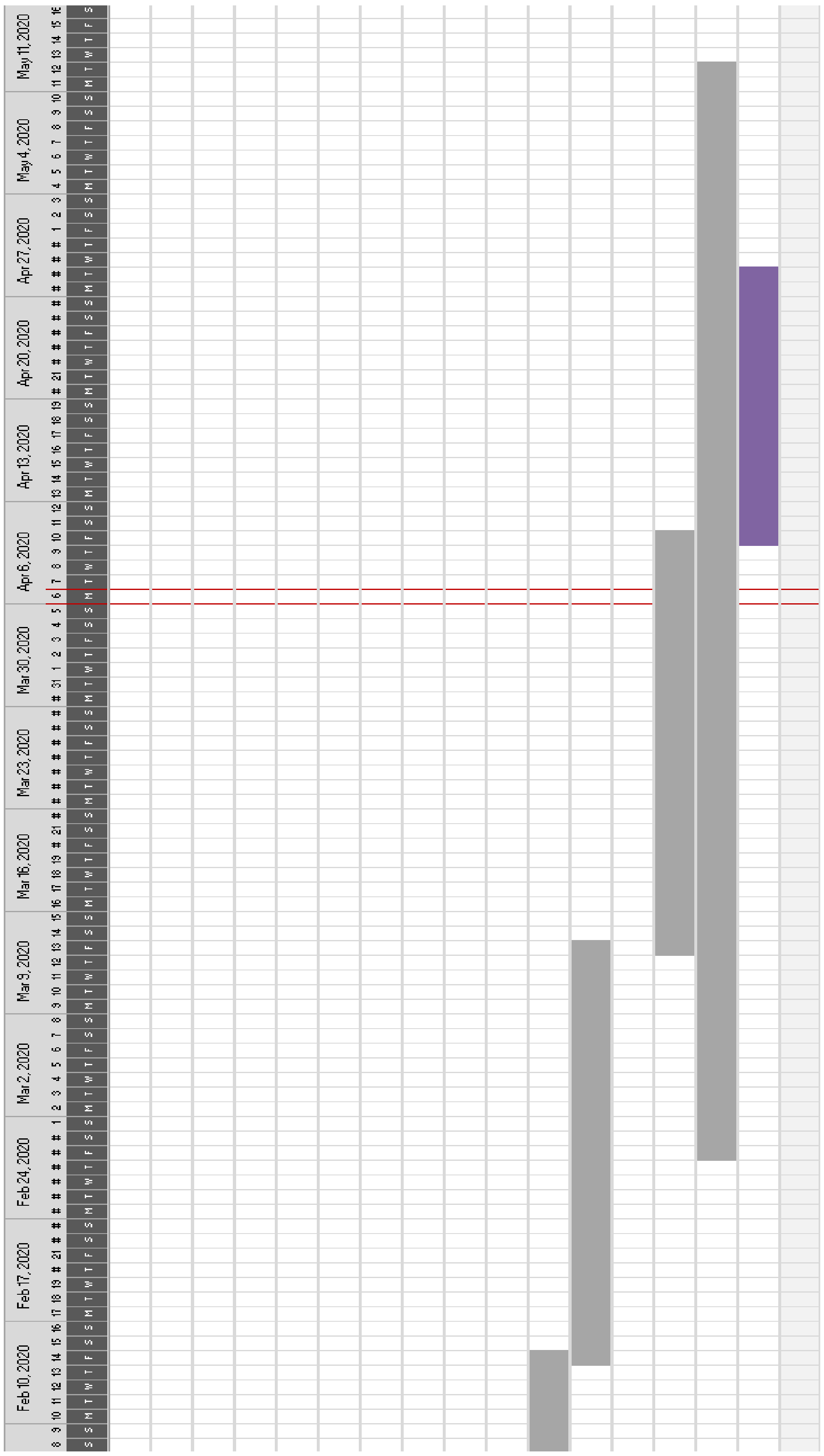
Supervisor: Dr Enrico Gerding eg@ecs.soton.ac.uk

Student: Edward Gorman eg6g17@soton.ac.uk

| TASK | | PROGRESS | START | END |
|---|---|---|---|---|
| **Data Collection and Understanding** | | | | |
| Task 1 | Reading Papers | 100% | 10/11/19 | 12/6/19 |
| Task 2 | Data Collection | 100% | 10/11/19 | 11/23/19 |
| Task 3 | Data Understanding | 100% | 10/25/19 | 11/23/19 |
| **Shallow Learning Algorithms** | | | | |
| Task 1 | Progress Report | 100% | 11/1/19 | 12/10/19 |
| Task 2 | Feature Engineering | 100% | 11/11/19 | 12/6/19 |
| Task 3 | SL Implementation | 100% | 11/23/19 | 12/6/19 |
| **Deep Learning Algorithms** | | | | |
| Task 1 | React to Progress Feedback | 100% | 1/27/20 | 2/7/20 |
| Task 2 | Feature Engineering | 100% | 1/27/20 | 2/14/20 |
| Task 3 | DL Implementation | 100% | 2/14/20 | 3/13/20 |
| **Final Report and Extension** | | | | |
| Task 1 | Model Refinement | 100% | 3/13/20 | 4/10/20 |
| Task 2 | Final Report | 100% | 2/28/20 | 5/12/20 |
| Task 3 | Extension | 0% | 4/10/20 | 4/28/20 |
| *Insert new rows ABOVE this one* | | | | |

Figure 2.1: Project Gantt Chart


Nov 18, 2019
Nov 25, 2019
Dec 2, 2019
Dec 9, 2019
Dec 16, 2019
Dec 23, 2019
Dec 30, 2019
Jan 6, 2020
Jan 13, 2020
Jan 20, 2020
Jan 27, 2020
Feb 3, 2020
Feb 10, 2020
Break in project to revise for exams/finish other module courseworks


Feb 10, 2020
Feb 17, 2020
Feb 24, 2020
Mar 2, 2020
Mar 9, 2020
Mar 16, 2020
Mar 23, 2020
Mar 30, 2020
Apr 6, 2020
Apr 13, 2020
Apr 20, 2020
Apr 27, 2020
May 4, 2020
May 11, 2020

## 2.3 Project Constraints

In order to gain approval for using the data required for the investigation, an Ethics Approval Form for Secondary Data Analysis needed to be completed detailing how the gathered information would be obtained/stored/analysed. This request was approved on the 6th of November 2019 and was valid until the end of the project due date on the 12th of May 2020. If required a request for an extension or additional analysis could be made if the project changed substantially.

Another constraint on the project was the budget supplied by the University of Southampton which was set to £150 per person. The purpose of this budget was to enable the purchasing of components/software/media required to complete the project. To account for this free services were utilised where possible (e.g. university owned textbooks, free IDE software) to reserve this budget in case it was required later on. If required a request for additional funding could be made if there was substantial reasoning and planning for the expenditure.

## 2.4 Risk Assessment

A risk assessment was completed to identify relevant risks to the project and arrange appropriate mitigation strategies depending on the likelihood and severity of the risk. High exposure risks are defined as risks with an exposure greater than 30 and for which action has been taken to avoid or a contingency plan has been constructed to mitigate it. Comparatively, for low exposure risks reasonable steps are taken to avoid them with minimal cost to time and resources (see Table 2.4 for full risk assessment). Using these ratings on a scale of 1 to 10, the risk exposure can be calculated:

$$\text{Risk Exposure} = \text{Likelihood of occurring x Severity of event}$$

Figure 2.2: Equation for calculating Risk Exposure.

During the latter half of this project an exceptional circumstance occurred known as the COVID-19 Pandemic. The actions by the UK Government and University of Southampton prevented meeting with the project supervisor in person and utilising on campus services. To mitigate these physical limitations I coordinated with the project supervisor to complete our biweekly meetings via email and through online video chat if necessary, as well as familiarising myself with the Lyceum Linux Teaching Cluster in the event that access to greater amounts of computing power was required.

| **Risk Description** | **Chance** | **Severity** | **Exposure** | **Mitigation Strategy** |
|---|---|---|---|---|
| Project purpose and goals are not well defined | 3 | 7 | 21 | Avoided by specifically stating the goals of the project and confirming with supervisor in planning stages. Will regularly compare technical progress and development direction regarding project goals. |
| Delay in earlier project causes subsequent delays | 5 | 7 | 35 | Avoid by weekly checking progress made against Gantt chart to see if I am falling behind in areas and if so adjust schedule to account for delay and confirm this with supervisor. |
| Supervisor unavailable or has left university | 3 | 4 | 12 | Will routinely meet up with supervisor every 2 weeks to discuss progress and identify circumstances that may affect communications. If persists will contact senior tutors. |
| Illness or injury preventing work on project | 2 | 8 | 16 | Reduce impact by adjusting project schedule to account for time lost. Also inform supervisor and negotiate new date and requirements for deadlines. |
| Other obligations preventing work on project | 4 | 8 | 32 | Avoid by scheduling other responsibilities where possible and allocating enough slack in time to complete project tasks. |
| Unable to collect data suitable for research | 4 | 7 | 28 | Reduced project delay by searching immediately for data sources and reduced likelihood by pursuing multiple different sources. |
| Failing to receive ethical approval for research | 2 | 9 | 18 | Will reduce the information that can be collected for each URL, as well as how it can be processed. If required can anonymise personal details further. |
| Loss of device, digital files or physical files. | 2 | 9 | 18 | Avoid impact by continuously backing up project files via 3 different back services. Can use university devices to replace laptop or other resource. |
| Unable to employ feature extraction or ML techniques | 4 | 9 | 36 | Make use of Scikit-learn library implementing ML algorithms. Will create testing module to confirm data processing performs functions correctly. |

Table 2.1: Table of Risks and Mitigation Strategies

# Chapter 3

# Related Works

The purpose of this chapter is to discuss related works thematically according to their proposed URL detection system and to highlight the advantages and disadvantages of each approach. This section starts by detailing popular features representations extracted from the URL or associated metadata, then provides a critical analysis of algorithm implementations, and finally summarises the current state of research in this area.

## 3.1 Feature Engineering

Ma et al tackle the problem of detecting phishing URLs using a lexical representation and propose extracting the length of the URL string, number of full stops or '.', and a 'bag of words' (BOW) representation of words found within the URL in an unordered list [11]. They find these lexical techniques alone to be an effective feature set achieving an error percentage of 1.93% on unseen URLs, placing it among the top performing feature representations. Furthermore they favour this approach over Web page content analysis since it is faster/lightweight and is universal in the context of how they are obtained (i.e. URLs are always present but Web page content varies widely). However, they note that lexical analysis may be evaded using redirection services by hiding the URL and potentially negate the effectiveness of this approach, concluding that lexical features should be used in tandem with other feature representations and supplementary to blacklist technologies. These findings are particularly impressive when considering phishing URLs are designed to appear similar to those they pretend to be, suggesting lexical-based techniques would be highly discriminative for detecting any type of malicious URL. However, lexical features may be susceptible to evasion via redirection services and thus other representations must be considered in addition.

Malicious URLs are also known for replacing the host name of a URL with the IP address and port for the Web server to avoid scrutiny over their registered domain name, so looking for anomalous alphanumeric distributions may be symptomatic as noted by [1]. Mamun et al explore the statistical aspects of a lexical representation including the concept for calculating the 'character continuity rate' which represents the number of continuous alphabetical characters in each URL, as well as features related to the frequency of characters/tokens found within the URL [13]. They report that their lexical analysis alone was able to achieve accuracies over 95% for every machine learning algorithm and also when training for a specific type of malicious URL (i.e. spam, phishing, malware, and also defacement). Similarly to [11] they conclude that their system is best suited as an "addon" for blacklist technologies where the URL has not been classified before and is unknown whether the Web page is malicious or not. This paper provides an excellent overview for statistical techniques in the context of lexical URL representations; in particular, their rigorous cross-testing with different algorithms gives confidence behind their conclusions for lexical-based features.

In addition to assessing a lexical representation of malicious URLs, Ma also explores host-based features gathered from characteristics of the Web server hosting the URL including IP address properties (e.g. IP of the domain), Whois properties (i.e. public information stored by Registrar), domain name properties (e.g. time-to-live for DNS records), and geographic properties (e.g. server location) [11]. They report a similar predictive ability in comparison to lexical techniques with an error percentage of 2.26%, attributing this capability to factors that malicious sites may share including "a particular IP prefix, ISP, [and] country". They selected these host-base features for their ability to be collected automatically via scripts which is important when considering the system in an operational context where a new URL may be input by many users and the data must be collected rapidly. These findings have considerable influence with regards to this problem since host-based features are universal for each domain, avoid interacting with any potentially malicious files, and these properties won't be considered by maleficent individuals and therefore are less likely to be evaded. However this makes the assumption that all malicious websites operate with similar practices or that benign websites operate significantly differently, which may not be realistic.

Thomas et al review their own set of host-based features in the context of a spam filtering service assessing properties such as geolocation, domain name services properties (e.g. mail server domains and IPs), and routing data for each IP encountered (i.e. collecting the autonomous system number prefix for each node encountered) [20]. They train a classifier on each feature to understand which are "the most influential" finding that geolocation performed the best of the host-based features achieving 81% accuracy, followed by routing information at 71%, and domain name properties at 72%. Their findings may suggest that spam services rapidly alternate between IP addresses and server locations which ma because this tactic avoids existing blacklist technologies from blocking them. They go on to elaborate that attackers may also use "mutable content" to circumvent becoming blacklisted by presenting a benign page when classified, and later altering this to include spam content or a redirect to a new landing page. These factors combined may encumber host-based approaches, particularly by large-scale maleficent organisations who are experienced and adapt quickly to transfer their sites. Overall, Thomas et al have shown that practical results may vary and are heavily susceptible to new emergent tactics that counter blacklisting techniques which will also be true of machine learning approaches.

Web page content is another source of feature extraction from the corresponding URL. Choi et al discuss deriving statistical based metrics from the Hyper Text Markup Language (HTML) code for possible indications of malicious intent revealing that one attack strategy injects malicious code into Web pages, thus "hiding the exploits" in otherwise benign pages [3]. They suggest extracting the following features: frequency of HTML tags, iframe tags (known for embedding content from other Web pages), hyperlinks (which may also be malicious), frequency of suspicious Javascript (JS) functions, and total number number of suspicious JS functions. In their experiments this combination of features produced a predictive model with an accuracy of 80%. Their findings fall short of what other feature representations have achieved demonstrating this approach may not be as effective, potentially due to evasion tactics via text obfuscation which Choi et al mention may have "mislead webpage content features", with the solution involving functionality capable of translating the obfuscated text to a more human-readable format for subsequent text analysis. Overall this set of techniques does not achieve a similar predictive ability to that of other feature representations, and when considering this technique requires interaction with a potentially malicious document it suggests this approach may not be suitable.

Another content-based approach involves applying image analysis techniques to screenshots of Web pages as discussed in [2, 1]. The reasoning behind this is that the HTML code isn't enough to understand what the page will look like to the user especially when one considers that most of the web page styling is written in a separate CSS file. Bannur et al's approach involves identifying key points within each image using SIFT (Scale Invariant Feature Transform) and then comparing these points to a repository of common web logos to generate a ranking of closeness [1]. If an exact match is found and the URL does not match the domain of the logo then this is a likely indication for a phishing site.

They report that this approach has an error rate of ~8% and did not “help to improve classification” when used in addition with other feature extraction techniques. This approach becomes further undesirable when considering image analysis takes great amounts of time and that retrieving the screenshot requires loading the Web page fully, allowing potentially malicious scripts to execute upon loading. Therefore, image analysis of the Web page does not appear to be reliable nor effective solution for distinguishing malicious from benign URLs and should not be considered for a detection system.

## 3.2 Model Survey

Kazemian and Ahmed approach this problem with a feature set consisting of lexical and content-based properties, and investigate how the following batch learning algorithms (i.e. algorithms that train on all data at one time) compare in terms of prediction accuracy and their AUC-ROC curves (these performance metrics are explained in Section 4.5): K-Nearest Neighbors (KNN), Radial Basis Function Support Vector Machine (RBF SVM), Linear Support Vector Machine (Linear SVM), and Naive Bayes [8]. They found that KNN and Naive Bayes recorded the lowest accuracies of 91% and 85% respectively, with KNN’s AUC score reporting a value of 0.66 (where 0.5 represents a random prediction function). Comparatively the RBF SVM achieved an accuracy of 97% and AUC of 0.91, while the Linear SVM reported an accuracy of 92% but a higher AUC score of 0.93. This suggests that while the RBF SVM was more accurate in its predictions, the Linear SVM was closer to the optimal detection model which is represented by an AUC score of 1. The batch learning approach is advantageous because once a model is trained it can be deployed to a system and require no additional modifications, and as evidenced by Kazemian can achieve satisfactory results. However, if we wanted to train the model on a new set of data in order to adapt to changing trends then the model must be trained anew, creating downtime within the system during this and potentially longer if the process fails for whatever reason. Despite this SVMs appear to be adept at learning from lexical and content-based feature representations and therefore are strong candidates for the learning mechanism in a malicious detection system.

In contrast, Zhao and Hoi investigate a wide array of online learning algorithms (i.e. algorithms that train as new data arrives) and their ability to learn from a combination of lexical and host-based features. These models include: Perceptron, Passive Aggressive, Confidence Weighted, and Cont-sensitive Online Active Learning (CSOAL, their proposed learning algorithm which differs from normal online algorithms by its method for updating model weightings) [21]. They find that applying these algorithms yields a consistently high accuracy above 96% for each model, demonstrating the capability of this type of algorithm. Furthermore the ability to routinely update models to tweak or improve performance is beneficial making them "highly efficient and scalable for web-scale applications". The online learning approach overcomes batch learning techniques where new data can be added routinely, as can be imagined in such a system where malicious URLs are discovered daily. However, the training sample required by Zhao for the algorithms to learn effective feature representations required a sample size of 1 million, making this approach potentially infeasible for host and content-based feature sets. Overall, online algorithms intuitively fit what a malicious URL detection system would require for the machine learning component, being supported by Zhao et al's results but potentially limited by the quantity of data available.

A final type of learning algorithm is deep learning (most commonly referring to neural network models) which is a technique investigated by Le et al in the context of malicious URL detection [10]. One aim for these techniques is to avoid the feature extraction process by learning effective feature representations automatically from the raw data, sidestepping the requirement for a domain specific expert to perform the feature extraction process. Le discusses using Convolutional Neural Networks (CNN) to extract structural information within the character and word encodings, surpassing lexical feature extraction techniques which are unable to account for unseen words whereas character level CNNs can adapt easily. One drawback for this set of techniques is that they require substantially more samples to learn from in comparison to batch and online learning techniques, restricting the data sources that can be learned from to lexical based. Le et al are able to produce a malicious detection model with an AUC score of 0.99 when learning from a combination of character and words representations of the lexical data, achieving performance akin to that of the ideal classifier and demonstrating the effectiveness for this approach. Deep learning is an area that has not been explored in much depth with respect to malicious URL detection and hence may provide exceptional performance; however, it is limited by the data it can learn from and the time available to train such a system which is extensive.

## 3.3 Summary

It is evident that the concept of applying machine learning techniques to detect malicious URLs is well explored from both a feature representation and algorithm implementation perspective. It is notable that these papers rarely compare the full range of possible feature representations and algorithm implementations, thus they cannot confirm whether their approach is the most optimal for the given problem. Furthermore many assess their detection systems in terms of accuracy rather than metrics such as recall (which measure how effectively the algorithm captures all malicious URLs) and hence do not significantly penalise cases where malicious URLs are incorrectly classified as benign.

# Chapter 4

# Methodology

This chapter explains the processes used during this investigation detailing specific methodologies to ensure the repeatability of the results and validify the conclusions. It begins with a formal definition of the problem setting related to malicious URL detection, followed by how each type of data was gathered, how data was transformed for each feature representation, how the machine learning algorithms learned from these feature sets, and which performance metrics were selected to measure the ability of each algorithm to detect malicious URLs.

## 4.1 Problem Formulation

Malicious URL detection is commonly formalised as a supervised binary classification problem as described in [10] and [16]. 'Binary' classification simply refers to data points classified into one of two groups. 'Supervised' models are trained on datasets labelled with a ground truth (i.e. what you are trying to predict). The problem setting for malicious URL detection in this mode is:

1. Given a dataset with T URLs $(u_1, y_1)$,. . . ,$(u_t, y_t)$ where $u_t$ for t=1,. . . ,T represents a set of URLs, and $y_t \in$ 1, -1 where 1=malicious and 0=benign.
2. Extract a feature representation from the URL such that $u_t \rightarrow x_t$ where $x_t \in \mathbb{R}^{\mathrm{d}}$ is a d-dimensional vector representing the URL.
3. Apply a prediction function using an ML model to predict which class a given URL belongs: $f : \mathbb{R}^{\mathrm{d}} \rightarrow \mathbb{R}$.

## 4.2 Data Gathering

A large collection of malicious and benign URLs were accumulated for this investigation from several blacklist (list of known malicious URLs) and whitelist (list of known benign URLs) sources, to be combined into one complete CSV file delimited by tabs and encoded in UTF-8 (see Appendix A for full list of sources). Each entry in this file consists of the URL string followed by a label which is either '1' for malicious or '0' for benign. In total there are 1,855,369 URLs (after removing repeated URLs and improperly formatted fields) of which 27.5% are malicious and 72.5% are benign, demonstrating a significant bias towards benign URLs. The restraints passed on data gathering is that it must not identify individuals nor extract their gender/race/religion in accordance with the ethics approval as mentioned in Section 2.3.

Host-based features were extracted by an automated script on each URL to generate a new CSV file containing the additional data. This was performed using a Python script that executed a Whois request to the Whois database and deriving the following features: location, registrars, server count, creation date, update date, expiration date, and connection latency. Similarly, content-based features were extracted by another automated Python script which sends a GET request to the web server and extracts the following: content type, is page redirection, length of content and content of the web page (up to 100,000 characters). All data fields were moderately processed during collection to account for missing information or malformed requests, replacing fields with Not Applicable (NA) values for category based information (e.g. country location) and a numerical value dependent on the field (i.e. 0, median value, max value).

The data were cleaned removing any duplicate entries and those containing malformed fields. Most machine learning algorithms can learn effective feature representations on a dataset of approximately 20,000 samples (as demonstrated in [3, 8]) which determined how long the automated Python scripts collected samples for host-based and content-based feature sets. Other algorithms (e.g. neural networks) require considerably more samples to perform the learning process (see [10]) and will use the full list of accumulated URLs from which lexical features may be derived. The data is then divided randomly between a training and testing set with a 4:1 distribution respectively. The purpose of constructing separate data sets is to test models on unseen data and verify they are basing their predictions on the characteristics of URLs rather than what they have seen before during training.

## 4.3 Feature Extraction

The feature extraction process can begin once the associated URL metadata has been gathered during Section 4.2. The selection of features listed below are a compilation of previously published and novel techniques, with the novel techniques from this paper being labelled with (NOVEL). Features are categorised by the source from which they originated including lexical-based, host-based, and content-based properties of the URL.

### 4.3.1 Lexical Properties

The lexical properties of a URL refer to the URL string and the individual characters that constitute it. The features extracted are predominantly statistical properties of the text but also includes a natural language processing technique since URLs can be akin to human text. These include:

1. Length of URL - The number of characters that make up the URL. This can be calculated by measuring the length of the URL string or the number of characters that constitute it.
2. Numer of labels - The number of sections delimited by a full stop. This can be calculated by counting the number of '.' characters that appear within the URL string.
3. Average length of labels (NOVEL) - The average length of each section delimited by a full stop. This can be calculated by measuring the length of each section delimited by '.' and dividing by the number of '.' characters.
4. Number of normal characters - The number of alphabetic characters (i.e. a-z, A-Z) that make up the URL. This can be calculated by counting each letter of the alphabet that appears in the URL.
5. Number of special characters - The number of punctuation characters (e.g.. ./-_?) that make up the URL. This can be calculated by counting each punctuation character that appears in the URL.
6. Number of numerical characters - The number of numerical characters (i.e. 0-9) that make up the URL. This can be calculated by counting each numerical character that appears in the URL.

7. Bag of Words - A vector representation of the 200 most common words present within a URL. This vocabulary was generated from the training data set consisting of the most common words (i.e. tokens separated by '-' and '.' ) found in the URLs (see Appendix B for full vocabulary).

The deep learning algorithms reviewed in this research use raw lexical data as input into their neural network models. This data is represented at three different levels:

1. Character level - Each character of the URL becomes an input neuron in the first layer of the neural network. For example: google.com/web-page.
2. Word level - Each word of the URL (delimited by '-', '.', '/') becomes an input neuron in the first layer of the neural network. For example: google.com/web-page.
3. Label level (NOVEL) - Each label of the URL (delimited by '.', '/') becomes an input neuron in the first layer of the neural network. For example: google.com/web-page.

### 4.3.2 Host Properties

The host properties of a URL refer to the characteristics of the Web server that hosts the Web address. This information originates from the metadata collected via Whois requests, as discussed in Section 4.2. These include:

1. Registrant location - The country of the user who owns the domain. This is extracted from the Whois request relating to the geographical location of the owner of the domain.
2. Number of servers (NOVEL) - The number of servers associated with the given domain. This counts the number of name servers that handle queries directed towards the domain.
3. Registrar (NOVEL) - The registrar responsible for the domain. This creates a vector representation of all Registrars where each dimension is a unique Registrar (akin to a bag of words approach).
4. Creation Date (NOVEL) - The date the domain was created. This measures how recent the domain was created by determining the age of the domain in number of months.

5. Modification Date (NOVEL) - The date the domain was modified. This measures when the domain was last modified by determining the time since the domain was last modified in number of months.

6. Expiration Date (NOVEL) - The date the domain will expire. This measures when the domain will expire (or has since expired) by determining the time to expiration in number of months.

7. Connection Latency - The delay between a Web request and the Web server response. This is determined using several ping requests and calculating the average return time in milliseconds.

### 4.3.3 Content Properties

The content properties of a URL refer to the characteristics of the document to which the URL is addressed to. Web pages are assessed by considering their HyperText Markup Language (HTML) and JavaScript (JS) properties which are the dominant languages used for creating Web pages. These include:

1. Is a redirect (NOVEL) - Whether the URL redirects the user. This occurs when a URL address immediately redirects the user to another Web page or domain and is represented as True or False.

2. Type of document - The type of document the URL is addressed to. This is extracted from the metadata of content request which represents the file type of the document.

3. Length of document - The number of characters that make up the Web page. This can be calculated by measuring the length of the content returned or the number of characters that constitute it.

4. HTML Bag of Words - A vector representation of blacklisted HTML tags found within the Web page. This vocabulary was generated from a list of common malicious tags found in the Web page (see Appendix C).

5. HTML average word length - The average length of each word in the HTML file. This can be calculated by measuring the length of each word in the HTML section of the Web page and dividing by the number of words.

6. JS Bag of Words - A vector representation of blacklisted JS tags found within the Web page. This vocabulary was generated from a list of common malicious functions found in the Web page (see Appendix C).

7. JS average word length - The average length of each word between '<script>' tags in the HTML file. This can be calculated by measuring the length of each word in the JS section of the Web page and dividing by the number of words.

## 4.4 Model Configuration

This section details the learning algorithms of the machine learning models and how they were developed including the methodology for the following processes:

- Additional data pre-processing - any extra data processing steps required before the algorithm can learn from the features.
- Hyperparameter tuning - the process of finding the optimal settings that are fixed before training begins (e.g. kernel type, number of estimators). This was implemented by experimentally finding the best hyperparameter combination after randomly testing 10% of all permutations.
- Model training - the process of finding the optimal settings that are altered during training (e.g. neuron weightings, threshold values). This process is algorithm-dependent and is implemented automatically using the Python module Scikit-learn and their 'fit' function for training models.

Machine learning can be divided between shallow learning (SL) algorithms and deep learning (DL) algorithms with the former referring to techniques that require structured input features from which to learn, while the latter refers to techniques that use unstructured input data.

### 4.4.1 Shallow Learning Algorithms

A Support Vector Machine (SVM) is a popular SL algorithm that uses a batch learning (algorithm is trained on data at one time) approach for classification tasks. These work by separating classifications in a m-dimensional feature space by a hyperplane such that only one classification resides on each side of the hyperplane, and when queried with a new instance its position within the feature space relative to the hyperplane determines to which class it belongs. This hyperplane is calculated by maximising the margin between the closest instances from different classes (also known as the support vectors) however there are variations of SVMs which allow classes to overlap to prevent overfitting (i.e. when the algorithm is trained too closely to the training set of data). This technique relies heavily on the spatial relationship of points within the feature space and so requires additional data pre-processing to standardise each dimension by subtracting the mean from each instance and scaling it to within one standard deviation. The algorithm hyperparameters that undergo tuning include: C (regularlisation parameter), kernel type, degree of polynomial kernel (if polynomial kernel used), and gamma (kernel coefficient). Training the model optimises the parameters of the hyperplane which define its shape within the feature space.

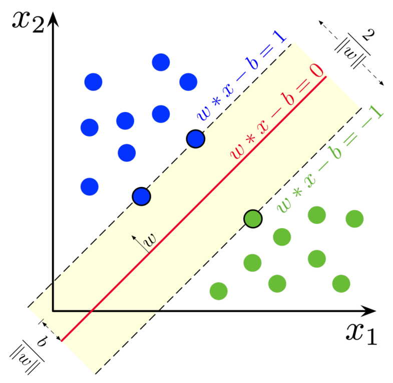


FIGURE 4.1: Diagram of a SVM.
This illustrates how a 1-dimensional plane separates two classes in a 2-dimensional space. Sourced from [9].

The next SL algorithm is Random Forest (RF) which also uses batch learning for classification tasks. This technique makes use of decision trees where at each node a question is asked about one feature of the data (e.g. is URL length <30) with the aim of separating classes such that a majority of one class travel down one branch and a majority of the other class travel down the other (i.e. keeping malicious and benign cases separate). An RF uses a large number of decision trees where each tree is given a random subset of features and a random subset of data from which to ask questions, ensuring variety between decision trees by preventing every tree asking the same question which divides the classes best. When analysing a new data instance this ensemble of classifiers output the most popular classification according to each decision tree, using the collective knowledge of the group rather than an individual. Unlike SVM this technique does not require additional data pre-processing since the threshold value that each decision tree selects is dependent on the value range of the chosen feature, so scaling the data would have no effect. The algorithm hyperparameters that undergo tuning include: number of estimators (or decision trees), max features, max tree depth, minimum samples per leaf, and minimum samples per split. Training the model optimises the collection of decision trees that constitute the RF and improve the ability of the ensemble to distinguish malicious from benign cases.

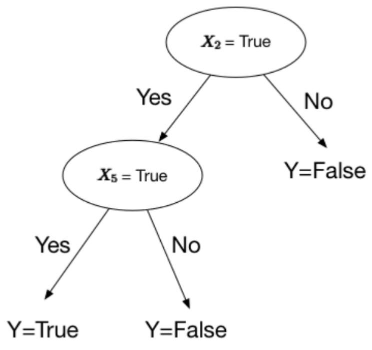


FIGURE 4.2: Diagram of a Decision Tree.
This illustrates how a single decision tree classifies instances based on their feature values. Sourced from [18].

The final SL algorithm is Perceptron (Pc) which uses online learning (algorithm is continuously trained as new data arrives) for classification tasks. This method consists of an m-dimensional input vector where each dimension represents a single feature. Each dimension is multiplied against a unique weighting and aggregated to produce a single value, which if sufficient according to some activation function (e.g. surpasses a fixed threshold) will output a value of 1 (or otherwise 0). This algorithm learns by adjusting its weightings for each input dimension each time a datum instance is passed through the Perceptron such that it nudges the values towards settings that would lead to a correct output prediction. This learning process demonstrates why this algorithm is suitable for online learning: it is able to train itself on new instances at any time. Similar to SVM this technique requires additional data processing to both standardise and scale the data since the input neurons must be within a certain range for the activation function to threshold correctly. The hyperparameters of this technique that undergo tuning include: penalty/regularlisation technique, alpha (constant that multiplies regularlisation term), max iterations over the training data, and early stopping (whether to stop training when the score stops improving). Training the model optimises the weightings applied to the input neurons such that the output neuron will predict the correct classification based off the original input instance.

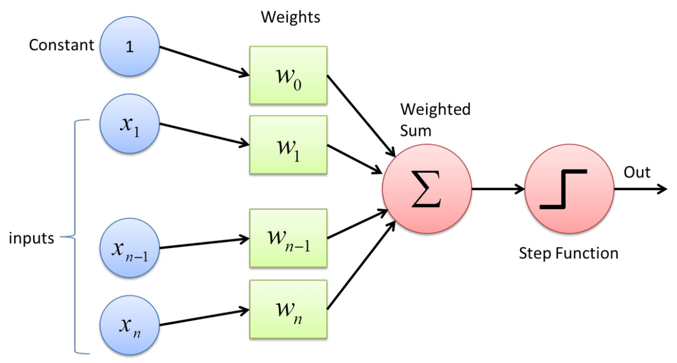


FIGURE 4.3: Diagram of a Perceptron.
This illustrates how an m-dimensional input vector representing the features are aggregated and output a result. Sourced from [17].

### 4.4.2 Deep Learning Algorithms

The DL algorithm explored in this research is the Convolutional Neural Network (CNN). Neural networks are essentially sequential layers of Perceptrons (discussed in Section 4.4.1) connected in parallel starting with some input space (e.g. the characters that comprise a URL) and ending with an output neuron where the value indicates indicates the predicted class. Convolutional Neural Networks (CNN) are a type of neural network that apply convolutional filters across the m-dimensional input (or any preceding layer) to extract patterns/structure present within the data. In this context the networks learn lexical patterns/structure within the raw data that distinguish between malicious and benign cases. The size of these patterns is dependent on the size of each kernel which in this research use kernel sizes of 3, 4, 5 and 6. These are followed by pooling layers to remove noise and reduce the dimensions of the output space by choosing the max value from the neighbouring region. Afterwards the convolution layers are concatenated together and pass through two fully connected neuron layers intersected by a dropout layer to regularlise the network, before reaching the final output neuron with a sigmoid activation function. The hyperparameters of this technique that undergo tuning include: the number of neurons in the fully connected layers, the activation function of the fully connected layers, and the dropout rate. Training neuron weightings occurs via the Adaptive Moment Estimation (Adam) optimiser which builds upon previous gradient descent algorithms by computing an adaptive learning rate from the first and second-order moments of the gradients, accelerating the rate at which the CNN improves. The diagram on the next page (see Figure 4.4) illustrates the architecture of this proposed CNN model which was initially based off [10].

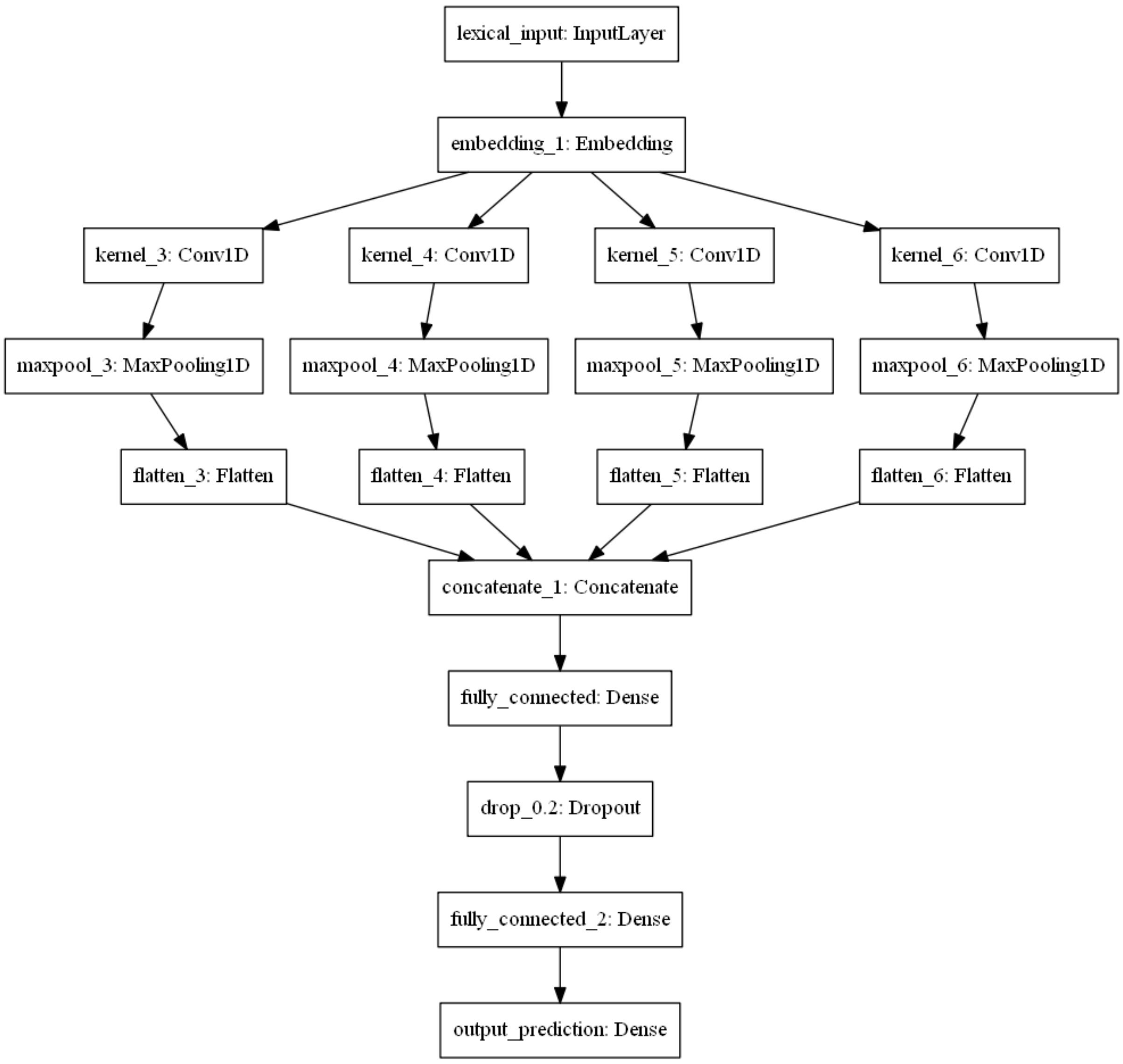


FIGURE 4.4: Diagram of Convolutional Neural Network.
This illustrates the layers and makeup of the discussed CNN architecture.

## 4.5 Performance Metrics

To assess the capability of each model's prediction function performance metrics were used. These are statistical methods for empirically measuring the ability of the model to predict whether a URL is malicious or benign. As stated in the problem formulation (see Section 4.1) the data is already labelled with the ground truth and thus can be compared directly against the results of the prediction function to assess a model's performance.

One performance metric is accuracy, calculated using the equation below:

$$\text{Accuracy} = \frac{\text{number of correct predictions}}{\text{total number of predictions}}$$

Figure 4.5: Equation for calculating Accuracy from predictions.

This is a simple way of demonstrating how successful the prediction model is for classification problems; however, it assumes an even distribution of each class in the data set. For example, given a data set where one class constitutes 90% of the overall data set while another class is the remaining 10%, a dumb classifier which always predicts the class with the greater number of instances would be 90% accurate while having learnt nothing. Consequently, accuracy is considered a poor performance metric for evaluating imbalanced data sets. The next performance metric requires the definition for the following concepts:

$$\begin{gathered}\text{True Positive (TP)} = \text{correct prediction of the positive class}\\ \text{True Negative (TN)} = \text{correct prediction of the negative class}\\ \text{False Positive (FP)} = \text{incorrect prediction of the positive class}\\ \text{False Negative (FN)} = \text{incorrect prediction of the negative class}\end{gathered}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP + FP}} \qquad \text{Recall} = \frac{\text{TP}}{\text{TP + FN}}$$

Figure 4.6: Equation for calculating Precision and Recall from predictions.

Precision is a measure of how well the model predicts positive cases, while recall is a measure of how well the model predicts all positive cases. These metrics improve upon accuracy by considering the cases when the prediction function was incorrect in their equations, thus a dumb classifier would not score as highly. Using the dumb classifier example where the 10% represents the positive class and the 90% represents the negative class we calculate that precision is equal to 0.1 and recall is equal to 1.0. This demonstrates how precision is not affected by the imbalanced dataset since the score is low (hence not precise) while recall appears to be flawed similar to accuracy; however, this is by design as its purpose is to represent how many of the positive class cases were correctly predicted which the dumb classifier is still successful at in this example.

The F1 score metric intends to construct a better representation of a model's accuracy by calculating the harmonic mean between precision and recall using the equation below:

$$\text{F1 score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

FIGURE 4.7: Equation for calculating F1 score from predictions.

This metric uses the values of precision and recall to determine its final value which is only high if both metrics are also high. Using the dumb classifier example once more we calculate the F1 score is equal to 0.182 which is a significantly better representation in comparison to accuracy because this low score reflects how the model has learned nothing and is simply predicting the most common class every time. Hence F1 score is the preferred performance metric for reporting the accuracy of a model.

A final metric that helps to visualise the performance of a model's prediction function is an AUC-ROC curve which plots the true positive rate (TPR) against the false positive rate (FPR). The resultant curve represents the Receiving Operator Characteristic (ROC) of the model at various classification thresholds equal to the number of predictions. At each new threshold the TPR/FPR is calculated from how many instances are classified correctly or incorrectly at this new decision threshold. The further the ROC curve is above the identity line (representing a random model) the better the predictive ability of the model. This principle is reflected by the Area Under the ROC Curve (AUC) score of the plot where a greater score indicates a better prediction model. These graphs are also 'insensitive to changes in class distribution' which is useful for biased data sets [4].

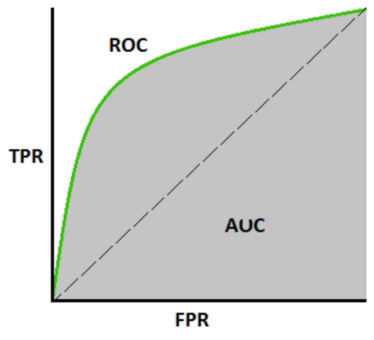


FIGURE 4.8: Example AUC-ROC plot.
The diagonal represents a random classifier and the green line represents a good classifier. Sourced from [14].

The performance metric that was used to assess features and algorithms in this report was recall, which was decided to encourage models to detect all malicious URLs rather than focusing on the accuracy of those predictions. AUC-ROC curves will be used in addition to illustrate all possible classification thresholds for an algorithm and demonstrate their trade-off between correct and incorrect predictions.

# Chapter 5

# Research Findings

| **Feature Representation** | | **Algorithm Recall** | | |
|---|---|---|---|---|
| **Set** | **Technique** | **SVM** | **RF** | **Pc** |
| Lexical | *All techniques combined* | 0.722 | 0.715 | 0.407 |
| | Length of URL | 0.638 | 0.627 | 0.033 |
| | Number of labels | 0.595 | 0.584 | 0.597 |
| | Average length of labels (novel) | 0.817 | 0.721 | 0.262 |
| | Number of normal characters | 0.649 | 0.592 | 0.878 |
| | Number of special characters | 0.594 | 0.667 | 0.691 |
| | Number of numerical characters | 0.252 | 0.271 | 0.290 |
| | Bag of words | 0.647 | 0.621 | 0.483 |
| Host | *All techniques combined* | 0.217 | 0.205 | 0.196 |
| | Server location | 0.117 | 0.133 | 0.117 |
| | Number of servers (novel) | 0.000 | 0.354 | 0.015 |
| | Registrars (novel) | 0.059 | 0.069 | 0.000 |
| | Creation date (novel) | 0.731 | 0.686 | 0.120 |
| | Modification date (novel) | 0.000 | 0.358 | 0.000 |
| | Expiration date (novel) | 0.299 | 0.312 | 0.000 |
| | Connection latency | 0.000 | 0.121 | 0.000 |
| Content | *All techniques combined* | 0.995 | 0.911 | 0.719 |
| | Is a redirect link (novel) | 0.000 | 0.000 | 0.000 |
| | Type of document | 0.000 | 0.000 | 0.000 |
| | Length of document | 0.997 | 0.364 | 0.009 |
| | HTML malicious bag of words | 0.992 | 0.939 | 0.823 |
| | HTML average word length (novel) | 0.000 | 0.167 | 0.661 |
| | JS malicious bag of words | 0.998 | 0.630 | 0.088 |
| | JS average word length (novel) | 0.885 | 0.212 | 0.213 |

| **Feature Representation** | | **Algorithm Recall** |
|---|---|---|
| **Set** | **Technique** | **CNN** |
| Lexical | Character level | 0.699 |
| | Word level | 0.638 |
| | Label level (novel) | 0.616 |

Table 5.1: Table of Results for Model Performance.

(Algorithms: SVM = Support Vector Machine, RF = Random Forest, Pc = Perceptron, CNN = Convolutional Neural Network)

# Chapter 6

# Evaluation

This chapter discusses how well the models were able to distinguish malicious from benign URLs by evaluating their performance metrics. It begins by assessing feature representations of URLs independent of their algorithm implementation, followed by an evaluation of each algorithm's performance, a critical comparison to results achieved in previously published papers, and finally the overall progress of the research project.

## 6.1 Recall Score Analysis

The features investigated in this project were assessed on both an individual and a group level in terms of their recall score, as is detailed in Table 5.1.

Models using all lexical-based techniques as their feature representation performed consistently well achieving recall scores of 0.722 and 0.715 for the SVM and RF algorithms, respectively, with Pc being the exception and scoring only 0.407. The majority of these techniques scored recall values between 0.6 and 0.8, with the novel feature 'average number of labels' being the most discriminative of the set reporting a score of 0.817 and 0.721 for SVM and RF, respectively, possibly indicating that malicious URLs use more/-less characters per label. In contrast, the 'number of numerical characters' technique performed the worst reporting a score of ~0.27 suggesting that the frequency of numerical characters does not discriminate well between malicious and benign URLs. The remaining techniques achieved recall scores of ~0.63 demonstrating moderately successful performance but failing to consistently detect all malicious cases.

The next set of features assessed were the host-based techniques which performed poorest in comparison to other feature representations with low recall scores of ~0.21 for all techniques combined. Some values from this set are reported as 0.000; this occurred when the model classified all cases as either all malicious or all benign, achieving recall scores of 1.000 and 0.000, respectively (but to avoid confusion those that achieved 1.000 are represented as 0.000). This low range of values suggests host-based features are ineffective techniques, with the notable exception for the novel technique ‘creation date’ which achieved values of 0.731 and 0.686 for SVM and RF models, respectively. This may be explained by the fact that malicious URLs have “short lifespan[s]” as discussed in [14] and so the number of months since the creation date for malicious URLs will be significantly less on average than benign URLs. Despite this, host-based techniques were generally ineffective in this project.

The content-based feature set appears to have performed the best relative to lexical and host-based techniques, scoring extremely high recall values of 0.995, 0.911, and 0.719 for SVM, RF, and Pc, respectively. On analysis, this set also contains recall scores of 0.000 which are due to the same reasons mentioned in host-based analysis. The bag of words technique applied to both HTML and JS text reported the best performance for this feature set suggesting the vocabulary (listed in Appendix C) is able to distinguish effectively sites that contain malicious functionality from those that do not based off the presence of these key words. The novel techniques that calculated the average length of HTML and JS words was less successful, suggesting text obfuscation within long strings is not as indicative for malicious intent as mentioned by Choi et al [3]. Overall, content-based analysis performed surprisingly well suggesting it may be beneficial to be included in a malicious URL detection system.

The final set of techniques listed in Table 5.1 are the different representations of the raw lexical data using in deep learning algorithms including the character, word, and label level techniques. The character-level input performed the best achieving a score of 0.699 suggesting it had the best ability to detect all malicious URLs. All scores ranged between ~0.61 and ~0.7 which is a similar range for the majority of lexical-based features, but is still lower than all lexical techniques combined for the SVM and RF models (despite having access to 1.1% of the data available for deep learning algorithms).

## 6.2 AUC-ROC Curve Analysis

The true positive and false positive rates have been plotted on AUC-ROC curves to illustrate how each model performs at varying classification thresholds.

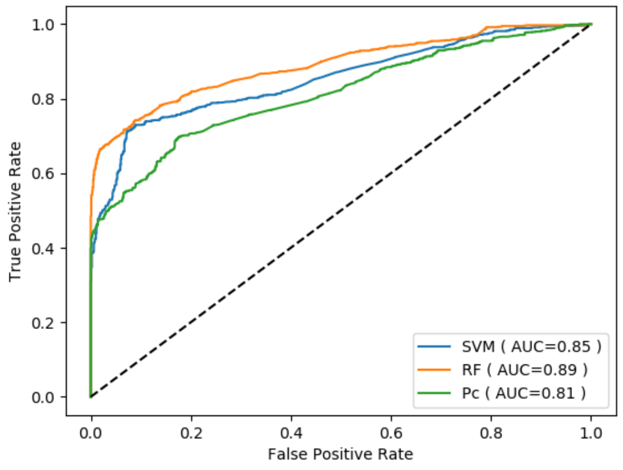


FIGURE 6.1: AUC-ROC Curve of SVM, RF and Pc on All Lexical Techniques.

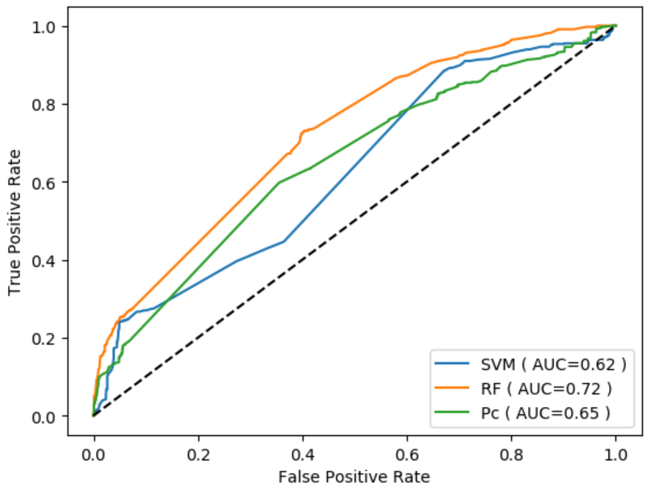


FIGURE 6.2: AUC-ROC Curve of SVM, RF and Pc on All Host Techniques.

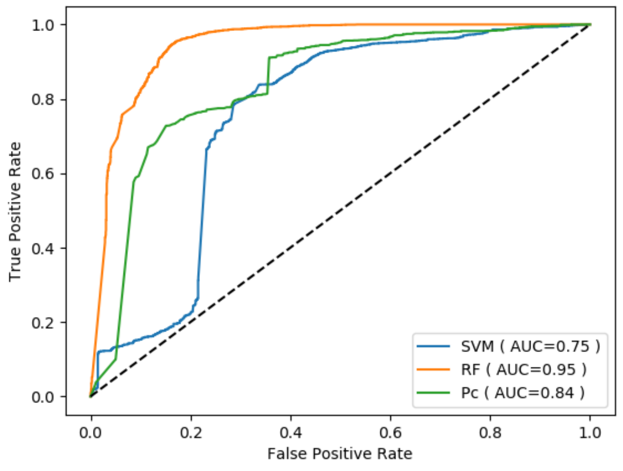


FIGURE 6.3: AUC-ROC Curve of SVM, RF and Pc on All Content Techniques.

When comparing the performance of shallow learning algorithms (see Figures 6.1, 6.2, 6.3) it is evident that RF is routinely the best performing model achieving the highest AUC scores for each feature representation, particularly for content-based techniques which reports an AUC value of 0.95. The SVM model appears to be the next best model over Pc when observing the recall scores presented in Table 5.1; however, the AUC-ROC diagrams would suggest that the Pc model is similar if not better in performance when compared to the SVM models, as is demonstrated by Pc having a greater AUC score for both host and content-based feature sets.

Each shallow learning algorithm in Figure 6.1 confirms that lexical-based features are an effective feature representation by achieving a consistent ROC curve with AUC scores averaging ~0.85. Host-based features were less effective as shown by curves that are considerably lower than that of lexical techniques and lower AUC scores of ~0.66. Algorithms applied to content-based techniques were less consistent as is shown in Figure 6.3, where the AUC value ranges between 0.95 to 0.75. These sudden jagged/horizontal sections are likely due to the SVM and Pc models predicting many cases with the same probability of being malicious, thus when the classification threshold passes these values they are suddenly all classified as false positives and consequently the true positive rate stops increasing.

The CNN algorithm shown in Figure 6.4 achieved similar results with the character-level model performing the best and reporting an AUC score of 0.84. This is followed by the word-level architecture which scored 0.81, and finally the label-level CNN scoring 0.8. These curves are similar to that of shallow learning applied to lexical based techniques (see Figure 6.1) suggesting there is little differentiating the two approaches.

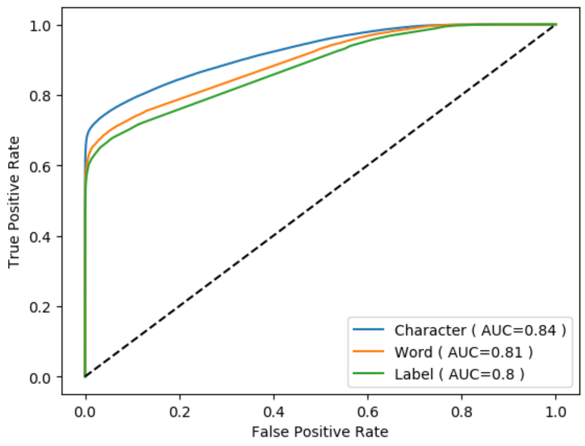


Figure 6.4: AUC-ROC Curve of CNN for each Lexical Encoding.

## 6.3 Related Work Comparison

It is challenging to compare the results collected in this research directly to that referenced in Chapter 3 since most sources evaluated their models in terms of accuracy/error percentage rather than recall. Despite this we can compare the relative abilities of approaches and comment where techniques may be best applied.

This research found that lexical-based techniques were moderately successful but not as discriminative for malicious URLs as suggested by both Ma and Mamun, failing to achieve their extremely low error percentage and high accuracy rates, respectively, despite investigating 6/7 of their lexical techniques [11, 13]. This may suggest lexical-based approaches are better suited for achieving high accuracy ratings rather than producing a high true positive/low false negative rate and thus making them less suitable for capturing all malicious URLs. Host-based techniques were surprisingly unsuitable, reporting the lowest recall scores of the feature representation. This is in contrast to Thomas et al's findings who reported an accuracy range of 70%-80% for their host-based features, while the feature closest to this success was the novel technique 'creation date' with a recall score of ~0.7. Figure 6.2 further confirms the truncated performance for these techniques found in this research, suggesting that (similar to lexical techniques) host-based features are not well designed for their ability to capture all malicious URLs. Content-based techniques ranged significantly between each algorithm implementation (as shown in Figure 6.3) however it may represent the best feature set for the problem of detecting all malicious URLs, allowing SVM and RF models to achieve recall scores greater than 0.9. This is an improvement over Choi et al's findings which produced a predictive model with 80% accuracy, despite also using the bag of words approach that was so successful in this research.

When comparing how the algorithms in this research performed against those in related works, it is notable that the Pc model demonstrated a significantly poorer ability to learn effective representations, and in particular for lexical-based techniques which Zhao and Hoi who reported accuracies of 96% [21]. These findings may be due to the training size of 20,000 compared to their 1,000,000 samples, which was decided so that comparing host and content-based techniques would be more fair since they were limited to 20,000 each. Therefore, given more data, this approach could achieve a performance similar to that demonstrated by Zhao, but when compared against SVM and RF it is evident that Pc is the inferior technique. The SVM algorithm implemented by Kazemian and Ahmed reported an AUC score of 0.93 for their best performing model on lexical-based features, which could not be matched as this investigation only managed an AUC score of 0.85 [8].

Furthermore the SVM models for host and content-based feature sets reported mid to low range performance when compared to Pc and RF, demonstrating a substandard ability to detect malicious URLs. Finally, the character and word-level CNN proposed by Le et al that managed to achieve AUC scores of 0.98 and 0.97 (when trained on 1 million samples each), respectively, also exceeded this paper's implementation with values of 0.84 and 0.81, respectively [10]. This may have been caused by differing hidden layers following the convolutional layers (see Figure 4.4) which were generated during the hyperparamter tuning stage to find the optimal arrangement of neurons and hidden layers.

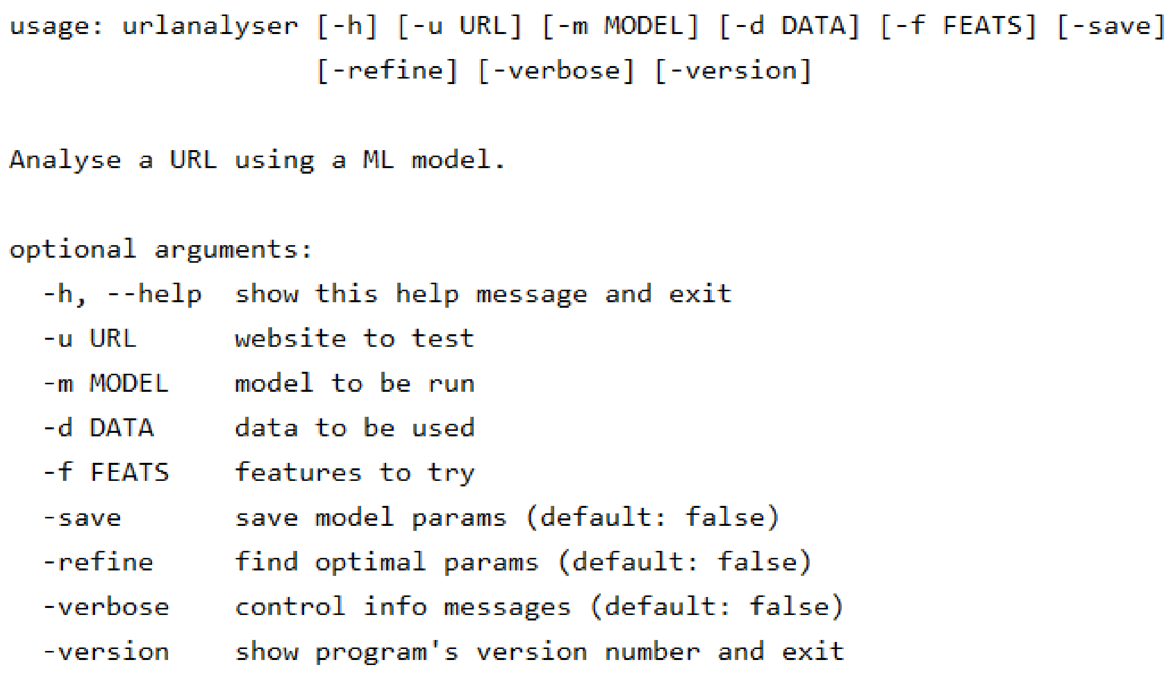
## 6.4 Project Evaluation

The project was on schedule for the entire duration of the research task (see Gantt Chart in Figure 2.1) and achieved the goals set out in Section 2.1.

The first goal was achieved via the implementation of multiple machine learning algorithms including SVM, RF, Perceptron, and CNN. These algorithms covered a wide range of learning techniques including batch learning, online learning, and deep learning with each algorithm undergoing hyperparameter tuning to ensure their settings were optimised correctly to learn the best feature representations from the data and produce optimal performance. These models were assessed in terms of the performance metric recall (as a measure of how many positives cases were correctly classified) and via AUC-ROC plots to understand how varying the classification thresholds affected their true positive and false positive rates.

The second goal was accomplished by not only investigating the lexical properties of the URL string, but host-based and content-properties in addition to determine how effective they were and if they could be useful in addition to (or in place of) lexical techniques. A total of 27 techniques were pursued of which one-third were novel, with the remaining being selected from related works. This created a wide range of techniques which were assessed on individual and group levels to assess their eligibility as feature representations, with their scores also being assessed in terms of the recall metric (as seen in Table 5.1).

The optional extension to develop a user interface was not successfully completed; however, a command line interface was created during the project to assist in the development of models by automating the functionality for training and testing models based on input. Furthermore this system is capable of returning whether an input URL is malicious or benign based on the selected model (best performing by default). If no URL is given it will test the model on data stored locally. This functionality is demonstrated in the



```
usage: urlanalyser [-h] [-u URL] [-m MODEL] [-d DATA] [-f FEATS] [-save]
                   [-refine] [-verbose] [-version]

Analyse a URL using a ML model.

optional arguments:
  -h, --help  show this help message and exit
  -u URL      website to test
  -m MODEL    model to be run
  -d DATA     data to be used
  -f FEATS    features to try
  -save       save model params (default: false)
  -refine     find optimal params (default: false)
  -verbose    control info messages (default: false)
  -version    show program's version number and exit
```

Figure 6.5: Demo of Command Line Interface for URLAnalyser Module.

screenshot (see Figure 6.5) illustrating which commands are present in the ‘urlanalyser’ module. The list of possible values for each argument are shown below:

- URL: google.co.uk, https://google.co.uk, https://www.google.co.uk
- MODEL: svm, rf, pc, cnn
- DATA: lexical, host, content
  - Note: cnn only accepts lexical data.
- FEATS: all, 1, 2, 3, 4, 5, 6, 7
  - Note: index corresponds to listing in Table 5.1.
  - Note: cnn only accepts range 1 to 3.

The word count for this report is 9724 as was calculated using the Perl script ‘Texcount’ as found here: `https://app.uio.no/ifi/texcount/DOC/TeXcount_3_0_0_24.pdf`. When this script is run on the ‘Project.tex’ file it returns the subcounts for each included file, and the files that constitute the body of the document are aggregated to produce the final word count.

# Chapter 7

# Conclusions

This chapter summarises the progress and conclusions made during the project, as well as potential avenues for further research.

## 7.1 Project Summary

This report provided a holistic overview of potential machine learning approaches taking into consideration both feature representations and algorithm implementations to find the most optimal combination for detecting malicious URLs. It finds that lexical techniques alone are not sufficient to capture a majority of malicious URL cases, hence content-based features may be considered as additional metadata since they proved themselves extremely effective at discriminating between malicious and benign URLs. The Random Forest algorithm was consistently the best performing model across every feature set and is the recommended learning algorithm from this research. It is recommended that this system be implemented alongside a blacklist component which is queried before the model is activated since the purpose of this model is to predict unknown URLs and the blacklist could identify known malicious sites with high confidence.

## 7.2 Future Research

While this project has met its goals, it could be extended to include more feature representations of a URL including the visual analysis aspect as discussed by [1] since content-based analysis is already being performed, so it could be activated in parallel. However, this approach must balance the risks associated with loading a potentially malicious URL safely, possibly into some sandbox environment which is isolated from the system running the virtual instance.

Another potential direction for further research is a more explorative analysis of deep learning techniques including Long Short Term Memory (LSTM) networks which include feedback connections within the network. Some research has been carried out by Pham et al who explored how character-level CNNs and LSTM compare for the problem of malicious URL detection and achieved a recall score of 0.98 when combining these systems [15].

A final area for future investigations could include context-based analysis of URL origination (e.g. in an email) which could help discriminate against phishing attacks if certain attributes of the sender do not match that of the URL (e.g. the sender's email domain does not match the URL domain). The text associated with the post may also be evaluated, potentially involving a Term Frequency Inverse Document Frequency (TFIDF) approach to identify which terms are more/less relevant in comparison to other posts in a corpus which may be indicative of malicious intent.

# Appendix A

# List of URL Sources

List of sources for malicious and benign URLs:

- https://www.phishtank.com/developer_info.php
- https://openphish.com/
- http://www.joewein.de/sw/blacklist.htm
- http://www.malwaredomains.com/wordpress/?page_id=66
- https://www.malwarepatrol.net/
- http://www.malwaredomainlist.com/
- https://majestic.com/reports/majestic-million
- http://www.hosts-file.net/

# Appendix B

# Word Level Vocabulary

Vocabulary (size 200) of most common words for lexical BOW.

000webhostapp, 10, 11, 12, 1252899642, 13inboxlight, 13inboxlightaspxn, 1774256418, 20, 2011, 239, 24, 25, 27, 28, 29, 2c, 2f, 6892, 91, abuse, ac, account, admin, adobe, alibaba, amazon, amp, and, aol, app, apple, archive, asp, aspx, aspxn, at, au, auth, battle, be, bin, biz, blog, blogspot, br, by, ca, cache, cc, cgi, ch, cl, cmd, cn, co, com, components, content, cp, css, cz, d3, data, de, default, dhl, doc, docs, document, download, dropbox, edu, email, en, es, eu, excel, exe, facebook, fav, fid, file, files, fr, free, gate, go, google, gov, gr, home, htm, html, http, https, hu, id, ii, images, img, in, includes, index, info, installer, ip, it, jdownloader, jp, jpg, js, kr, libraries, log, login, login_submit, ly, mail, mailbox, me, media, mobile, modules, moncompte, mx, my, name, net, new, news, nl, no, of, office, online, org, page, pages, panel, paypal, pdf, people, php, pl, plugins, post, public, rand, ref, remax, ro, ru, se, search, secure, security, service, session, signin, site, sites, system, templates, the, themes, tk, to, top, true, tv, ua, uk, update, upload, uploads, url, us, usa, user, userid, verification, verify, view, vn, watch, web, webmail, wiki, wikipedia, wordpress, work, wp, www, xn, xyz, yahoo, youtube, za, zip

# Appendix C

# HTML and JS Vocabulary

HTML blacklist vocabulary (size 22) sourced from [7].

applet, body, bgsound, base, basefont, embed, frame, frameset, head, html, id, iframe, ilayer, layer, link, meta, name, object, script, style, title, xml

JS blacklist vocabulary (size 17) sourced from [5].

getCookie, setCookie, indexOf, charAt, split, fromCharCode, charCodeAt, setInterval, setTimeout, eval, writeIn, write, appendChild, innerHTML, assign, replace, unescape

# Bibliography

[1] Sushma Bannur, Lawrence Saul, and Stefan Savage. Judging a site by its content: learning the textual, structural, and visual features of malicious web pages. 10 2011.

[2] Kuan-Ta Chen, Jau-Yuan Chen, Chun-Rong Huang, and Chu-Song Chen. Fighting phishing with discriminative keypoint features. *IEEE Internet Computing*, 13(3):56–63, 2009.

[3] Hyunsang Choi, Bin B. Zhu, and Heejo Lee. Detecting malicious web links and identifying their attack types. In *Proceedings of the 2nd USENIX Conference on Web Application Development*, page 11. USENIX Association, 2011.

[4] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[5] Shashank Gupta and BB Gupta. Cssxc: Context-sensitive sanitization framework for web applications against xss vulnerabilities in cloud environments. *Procedia Computer Science*, 85:198–205, 2016.

[6] Jason Hong. The state of phishing attacks. *Communications of the ACM*, 55(1):74–81, 2012.

[7] Joomla. Html tag filtering in joomla. Retrieved 17 November 2019 from: https://www.siteground.com/kb/html_tag_filtering_in_joomla/.

[8] Hassan B Kazemian and Shafi Ahmed. Comparisons of machine learning techniques for detecting malicious webpages. *Expert Systems with Applications*, 42(3):1166–1177, 2015.

[9] Larhmam. Svm margin diagram, October 2018. Retrieved 17 April 2020 from: https://commons.wikimedia.org/wiki/File:SVM_margin.png.

[10] Hung Le, Quang Pham, Doyen Sahoo, and Steven CH Hoi. Urlnet: Learning a url representation with deep learning for malicious url detection. *arXiv preprint arXiv:1802.03162*, 2018.

[11] Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. Beyond blacklists: Learning to detect malicious web sites from suspicious urls. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, page 1245–1254, New York, NY, USA, 2009. Association for Computing Machinery.

[12] Malwarebytes-Labs. State of malware, 2019. Retrieved 20 November 2019 from https://resources.malwarebytes.com/files/2019/01/Malwarebytes-Labs-2019-State-of-Malware-Report-2.pdf.

[13] Mohammad Mamun, Muhammad Rathore, Arash Habibi Lashkari, Natalia Stakhanova, and Ali Ghorbani. Detecting malicious urls using lexical analysis. volume 9955, pages 467–482, 09 2016.

[14] Sarang Narkhede. Understanding auc - roc curve. Retrieved 10 February 2020 from: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.

[15] Thuy Thi Thanh Pham, Van Nam Hoang, and Thanh Ngoc Ha. Exploring efficiency of character-level convolution neuron network and long short term memory on malicious url detection. In *Proceedings of the 2018 VII International Conference on Network, Communication and Computing*, ICNCC 2018, page 82–86, New York, NY, USA, 2018. Association for Computing Machinery.

[16] Doyen Sahoo, Chenghao Liu, and Steven CH Hoi. Malicious url detection using machine learning: A survey. *arXiv preprint arXiv:1701.07179*, 2017.

[17] Sagar Sharma. What the hell is perceptron?, September 2017. Retrieved 17 April 2020 from: https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53.

[18] Brandon Skerritt. What is a decision tree in machine learning?, October 2018. Retrieved 27 April 2020 from: https://hackernoon.com/what-is-a-decision-tree-in-machine-learning-15ce51dc445d.

[19] Sucuri-Inc. Hacked website report, 2018. Retrieved 20 November 2019 from Sucuri: https://sucuri.net/reports/19-sucuri-2018-hacked-report.pdf.

[20] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song. Design and evaluation of a real-time url spam filtering service. In *2011 IEEE Symposium on Security and Privacy*, pages 447–462, 2011.

[21] Peilin Zhao and Steven CH Hoi. Cost-sensitive online active learning with application to malicious url detection. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 919–927, 2013.