

Bsc (Hons) Computer Science With Professional Experience

Final Year Project Dissertation

Academic year 2016/2017

Project to distinguish phishing websites from benevolent ones

Author: Sophie Brown

Supervisor: Dr C.H.Bryant

## Abstract

Phishing is defined as “Fraud perpetrated on the Internet; *spec.* the impersonation of reputable companies in order to induce individuals to reveal personal information online.” (Murray and Burchfield, 2000). This project attempts to identify characteristics of phishing websites that can be used to detect them using supervised data mining. A mix of attributes provided by a human expert and a computer are used to generate a model which classifies websites as either phishing or benevolent.

Attributes provided manually are whether or not the website makes any offers in exchange for a user’s personal information, and how similar the look and feel of the website is to other, known benevolent websites. Attributes computed automatically are whether or not the website redirected the user elsewhere, the number of spelling errors, and how close the edit distance – the number of single-character edits between one string and another - of the URL is to a list of known benevolent websites.

Furthermore, a standalone application has been developed, allowing a user to input a URL, and the values of the human-provided attributes for the corresponding website. The application then gathers the remaining attributes and the model then classifies the website as either benign or phishing.

The project has shown promising results, with a predictive accuracy of almost 90% using cross-validation (Kohavi, 1995). Therefore, the attributes selected by this project appear to be highly relevant for detecting phishing websites. Additionally, manual user testing has found that the application works well for detecting phishing websites in practice.

## Table of Contents

Abstract.....	2
Chapter 1. Project Introduction.....	5
Section 1.1. Motivation.....	5
Section 1.2. Project Aim.....	5
Section 1.3. Project Objectives.....	5
Section 1.4. Planned approach.....	6
Section 1.5. Summary of project plan.....	7
Chapter 2. Literature review.....	10
Section 2.1. Related works.....	10
Section 2.2. Technology review.....	11
Subsection 2.2.1. Methodology review.....	11
Subsection 2.2.2. Languages and libraries review.....	12
Subsection 2.2.3. Supporting technologies review.....	13
Chapter 3. Requirements and design.....	15
Section 3.1. The Web Crawler.....	15

Subsection 3.1.1. Web crawler requirements.....	15
Subsection 3.1.2. Web Crawler Design.....	15
Section 3.2. The Data Mining model.....	17
Section 3.3. The user interface.....	18
Subsection 3.3.1. User interface requirements.....	18
Subsection 3.3.2. User interface design.....	19
Chapter 4. Web crawler implementation and testing.....	21
Section 4.1. Writing the code.....	21
Subsection 4.1.1. URL similarity to list of benevolent sites.....	21
Subsection 4.1.2. Redirection detection.....	22
Subsection 4.1.3. Spelling errors counting.....	23
Section 4.2. Writing unit tests.....	24
Subsection 4.2.1. URL similarity unit tests.....	24
Subsection 4.2.2. Redirection detection unit tests.....	25
Subsection 4.2.3. Spelling errors unit tests.....	25
Chapter 5. The data mining model – choosing an algorithm.....	26
Section 5.1. Gathering the data.....	26
Section 5.2. Preliminary data mining.....	26
Subsection 5.2.1. Initial experiments.....	27
Section 5.3. Further data mining.....	29
Subsection 5.3.1. Experiments with new algorithms.....	30
Section 5.4. Statistical analysis of algorithms.....	32
Chapter 6. User interface implementation and testing.....	35
Section 6.1. User interface implementation.....	35
Section 6.2. System integration.....	36
Section 6.3. User interface testing.....	36
Chapter 7. Critical evaluation and conclusion.....	41
Section 7.1. Critical evaluation of the project.....	41
Subsection 7.1.1. Review of the project’s achievements against its objectives.....	41
Subsection 7.1.2. Review of project plan.....	42
Subsection 7.1.3. Evaluation of system strengths and weaknesses.....	43
Subsection 7.1.4. Lessons learnt during the course of the project.....	43
Section 7.2. Project conclusion.....	44
Subsection 7.2.1. Comparison of results with existing literature.....	44

Subsection 7.2.2. Potential improvements to the project.....	44
Subsection 7.2.3. Review of other issues.....	45
References.....	45
Appendices.....	49
Appendix A. The project log book.....	49
Appendix B. The project plan.....	91
Appendix C. The project progress report.....	94
Appendix D. The ARFF dataset headers.....	99

# **Chapter 1. Project Introduction**

## **Section 1.1. Motivation**

This project was born out of a desire to understand how the process of data mining works in large-scale academic and industrial applications, as previous experience within the field of machine learning has been on small-scale case studies and in-class tutorials. Furthermore, a greater understanding of internet security was sought, specifically dealing with social engineering and the use of data mining methods for detecting these kinds of human-centric security attacks.

## **Section 1.2. Project Aim**

The aim of this project, as given in the project plan, is “to design, develop and implement a data mining application used for determining if a website is a phishing site, using a mixture of data gathered by a web crawler and data provided by a human, and to investigate whether the selected attributes are good indicators of whether a given website is a phishing site.” In other words, to select attributes of phishing websites, build a data mining model from these attributes, and then to test this model so as to ensure that the attributes selected do in fact generate a useful model, thereby implying that the selected features are useful for detection of phishing websites.

It should be noted at this point that the wording of the project aims effectively split the goal of the project into two closely related, but ultimately different, points. The first part of the aim is concerned with gathering attributes and building a data mining model from these, whereas the second - and more difficult part of the project – is concerned with ensuring that the model created by the first part of the project is fit for purpose. As such, it could be said that the project has two aims, and any one of these two will suffice for the purposes of considering the project to be complete. However, what has been done with the aim for this project is to hold the project to a higher standard than is typically expected – to not only create and deliver a data mining model for the purposes of detecting phishing websites, but to deliver one of high quality. What this means in practical terms, and an outline for achieving this, is described in the project objectives.

## **Section 1.3. Project Objectives**

The project objectives were composed with the intent of being able to easily fit into a time plan. As such, they each deal with achieving a specific part of the project. Furthermore, there are some optional objectives that deal with adding user interaction to the final data mining model, allowing a user to nominate their own websites and receive a prediction.

- To write a web crawler capable of recording the URL similarity, number of spelling errors and whether a redirection was attempted.
- To write unit tests to measure the performance of the crawler on test websites.
- To write a user interface consisting of a form that allows a user to submit a URL, similarity to an existing website, and whether or not the website in question offers something in exchange for personal information.
- To link the user interface to a back end, consisting of the crawler and an algorithm implemented within WEKA, such that the crawler will gather the remaining attributes and the algorithm will run on the resulting data.
- To configure the crawler to gather an approximate 1:5 ratio of phishing and non-phishing sites.

- To provide the human attributes in at least 90% of the training data.
- To evaluate algorithms selected for this type of dataset, and select one to generate a model.
- To estimate the predictive performance of this model.

The objectives as written give a coherent structure to the project, allowing a time plan to easily be written that follows these objectives. Furthermore, since the project is essentially a mix of software engineering and data mining, the planned approach to the project deserves some special thought. As such, it is the focus of the next section.

#### **Section 1.4. Planned approach**

Since this project does not make use of a publically available dataset, the first stage of this project is to identify the attributes that will compose the dataset that will be used throughout. It was decided early on to use a collaborative learning framework, in which some of the attributes are easily computable and hence provided by the web crawler mentioned in the objectives, and some are easy for a human to identify and therefore provided by a human. Throughout this dissertation, the terms “computed attributes” and “human attributes” will be used to distinguish between the two kinds of attributes. The attributes that were decided upon were selected in order to test, through the course of the project, how good these attributes are in terms of predicting phishing websites, as required by the second half of the project aim.

The human and computed attributes themselves were hinted at above, but it is worth taking the time here to formally define them, as they are a central part of the project.

The computed attributes take the form of the following:

- During the course of the project, a list of sites known to be benevolent is compiled. When classifying a website, its URL is checked for similarity with this list, using a metric known as edit distance – the number of character additions, removals or deletions required to transform one string into another. This similarity is recorded as one of the attributes.
- Whether or not a redirection occurred while trying to connect to the website is also recorded as an attribute.
- The number of spelling errors present on the crawled web page is recorded, and used as one of the attributes.

The human attributes are much more subjective, depending initially – during the data gathering phase - on my own judgement, and later – during the deployment phase - on the judgement of the individual user of the resultant application.

These attributes, then, are as follows:

- Whether or not the website makes any offers in exchange for personal information. This could be an e-commerce site making deals on the products it sells, or it could be a banking website demanding personal bank details to supposedly verify your account.
- How similar the website looks in terms of design and theme to a website that the user knows to be benevolent. This could mean something as vague as design decisions being largely the same across websites dealing in certain areas – for example, banking websites are likely to have vaguely similar designs – or as clear as copied HTML and CSS code.

The structure of the accumulated data will be WEKA's ARFF format. The ARFF headers can be found in Appendix 1.

The first part of the aim requires data to be gathered, a model to be generated and this model to be tested. Once a high quality model is attained, then a user interface is added in order to allow users to receive predictions for other, independent websites.

The data is gathered with the aid of a web crawler to automate the gathering of the computed attributes, developed specifically for this purpose. Once a suitable number of examples have been generated, a human expert then goes through this dataset and provides the human attributes.

Once a dataset is gathered and populated, the next step is to discover a data mining algorithm that will generate a high quality model. There are many different means of measuring the success of a model – low false positives, low cost of a false positive or a false negative – but for this project the measure that was used is the predictive accuracy of the model, because while cost is not a great concern for this project, making accurate predictions is. If the project were to be sold in a commercial environment, cost would become more of a factor as the system grew more popular.

Once a model is generated, it needs to be thoroughly tested in order to ensure the model gives the high predictive accuracy necessary to achieve the second part of the project aim. This was achieved using stratified ten-fold cross-validation, to ensure that as much of the data as possible was used for training during each fold, while maintaining a reasonably sized test set. In this way, a balance was struck between the size of the training and test sets.

Once a model has been built, trained and tested, it is combined with a user interface and the web crawler. The user interface allows a user to enter values for the human attributes, while the web crawler collects the values for the computed attributes. This is then associated with the dataset used to build the model, and as such the model can then classify the new instance. The results of this classification are then shown to the user, along with the predictive accuracy.

This section has detailed the approach taken to developing this application, which is a significant part of the project plan. A summary of the plan is laid out in the next section.

### **Section 1.5. Summary of project plan**

The project plan as submitted contains the following sections:

- Project Title,
- Author,
- Supervisor,
- Date of submission,
- Aim,
- Background,
- Objectives,
- Methodology,
- Resources,

- Time plan.

The project title, author, supervisor, aim, background and objectives have all been covered in previous sections of this dissertation, so for the sake of expediency, these shall not be repeated.

A full description of the project methodology and the processes used to arrive at it is described in Chapter 2, Section 2.2.1.

This leaves the resources necessary for the project and the time plan. The resources necessary for the project include a personal computer, with the following necessary software packages installed:

1. WEKA, a data mining tool written in Java. (Witten et al., 2005) WEKA will be used for trying out algorithms on test data as part of the experiments to create a machine learning model that gives a high predictive accuracy.
2. Java, for writing the web crawler that will gather the training data and the user interface, as well as using WEKA's API to access its functions within the user interface.
3. Eclipse, an industry-standard Java IDE (Integrated Development Environment). All Java development will be done in Eclipse.
4. VirtualBox. The web crawler will deliberately trawl the web looking for dangerous websites, and therefore needs to be run in an environment where the risk of viruses causing damage to the computer is minimal. A virtualized operating system fills this role, as a compromised or otherwise broken virtual operating system can be easily reimaged with no loss of resources.

Furthermore, in order to survey the available literature to be able to write about the project background and context in terms of existing research, said literature must be made available through the university library. Books, journal articles, websites and conference papers have all been used throughout this project, and referenced wherever they have been used.

This leads on to the time plan, which has been revised since the original plan submission. Subsection 7.1.2 goes into more detail on what these changes are, and lays out the reasons behind them. The revised project time plan is shown below.



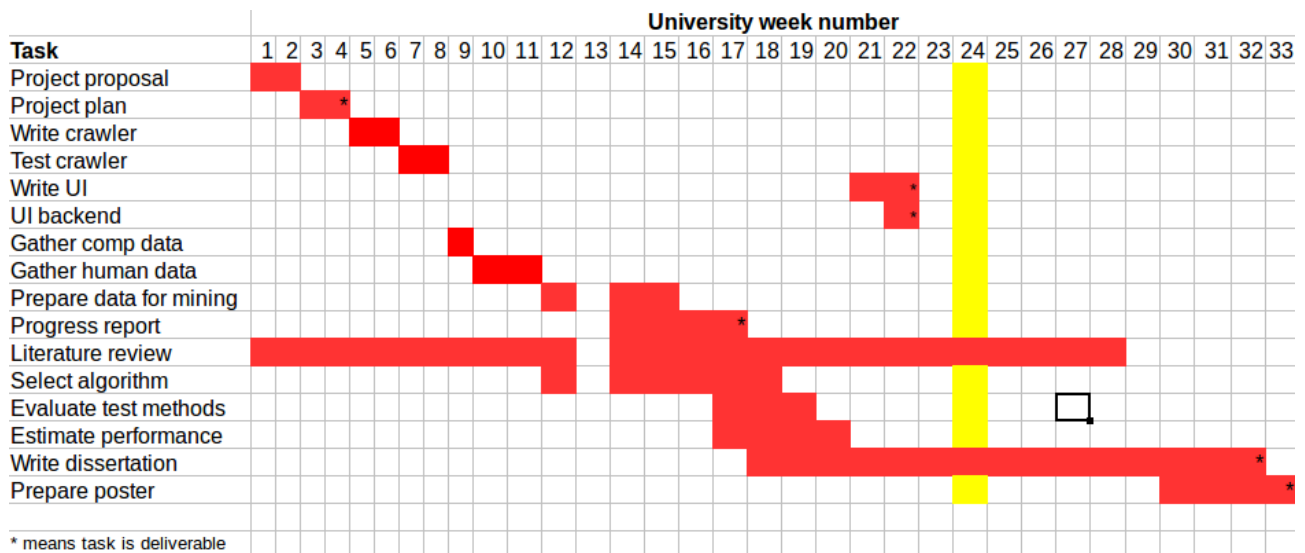


Figure 1.5. The updated project timeplan

This dissertation explains the process taken to build the web crawler, gather the training data, refine it, discover an algorithm that gives maximum predictive accuracy on this refined data, and finally how this algorithm was incorporated into a desktop application to allow users to classify their own websites. Before that, however, there is an explanation of the various technologies used during the project, as well as the project's context in terms of existing research.

## **Chapter 2. Literature review**

### **Section 2.1. Related works**

Since the beginning of human history, people have been violent towards each other. This desire, to improve oneself by diminishing others, has taken many different forms across history, from physical violence such as war, murder and assault, to emotional violence such as domestic abuse and bullying, to economic violence such as robbery and fraud. Phishing is a kind of fraud in which the scammer requests some personal information from the would-be victim, claiming to be part of some reputable organization. Then, when the victim provides this information, the scammer uses it for their own purposes, from stealing money in bank accounts to taking out loans in other people's names.

Phishing, as well as other forms of scam, have become much more prevalent with the rise of the internet – phishing attacks increased by 250% in the first quarter of 2016 alone (Phishing Activity Trends Report, 2016). Furthermore, in April 2016, an estimated 2.3 billion US dollars had been lost to a phishing scam in which the attackers posed as the CEO of the victim's workplace. (FBI, 2016)

Because of the rapid rate of phishing attacks, there is considerable interest in reliable ways to detect them, and this has led to phishing becoming a very well-studied problem, especially in the area of machine learning. (He et al., 2011) (Hadi et al, 2016) (Xiang et al., 2011) (Wenyin et al., 2005) (Kim and Huh, 2011) Phishing is of interest to the machine learning community because machine learning, as a discipline, primarily deals with non-exact problems where designing explicit algorithms is infeasible, and detecting phishing falls squarely into this category. Phishing depends on appearing to be a respectable and legitimate organization to a human user, and computers have no concept of "respectable" or "legitimate" - these concepts vary even among humans.

Many different methods have been adopted in the hopes of solving the problem of phishing websites. Major approaches involve analyzing URLs for features typical of a phishing website (Hajgude and Ragha, 2013) (Basnet and Doleck, 2015) or examining the website itself using image recognition techniques (Zhang et al., 2011) (Wenyin et al., 2005). A third approach, slightly more general in its scope of analysis but narrower in its methods, is the use of supervised classification techniques to analyze URLs, websites, and emails (Lakshmi and Vijaya, 2012) (Hadi et al, 2016) (Abu-Nimeh et al., 2007).

Out of these methods, the one chosen to focus on for the purposes of this project is the use of supervised classification techniques specifically to distinguish phishing websites from non-phishing, or benevolent ones. Work has been done in this area; Lakshmi and Vijaya (2012) have used multi-layer perceptrons, which is a neural network learner, decision tree induction using J48 - an open source implementation of C4.5 (Quinlan, 1992) - and bayesian analysis using Naive Bayes (Bayes, 1763) in order to classify websites as either phishing or benign.

Furthermore, Lakshmi and Vijaya (2012) have used many different website features as attributes for the prediction model – 17 different features based around the website's HTML, DOM (Document Object Model) tree and third-party services such as blacklisting – whereas this project uses only 5, which are detailed in Section 1.4. The attributes selected for this work are primarily based on data that can be gathered automatically using a customized web crawler, as well as data provided by a human expert simply looking at the website.

Another set of studies that heavily influences this work is that of Akanbi et al (2014) who carried out data mining using J48 (Quinlan, 1992), Random Forest (Breiman, 2001), K-nearest neighbour (Aha, Kibler and Albert, 1991), Support Vector Machines (Microsoft Research, 1998) and Naive Bayes (Bayes, 1763). This work was split into three studies, each roughly corresponding to phases of the CRISP-DM, or Cross Industry Standard Practice for Data Mining, methodology (Smart Vision – Europe, 2016) – the first corresponds to the data understanding and preparation phases of the methodology, the second focuses on evaluating this prepared data on the classifiers detailed above, as in the modelling and evaluation phases of CRISP-DM, and the third decides which algorithm gives the superior result in context of predicting phishing websites, ready to potentially be deployed into the real world.

This project uses Akanbi et al (2014) as a guideline, and follows a similar structure in the general sense, then builds on it by including a deployment to users by means of a front-end desktop application. The work done by Akanbi et al also inspires the choice of classifiers for this work, as well as work done by Islam and Abawajy (2013), as their use of K-nearest neighbour further supports its inclusion as a candidate algorithm for this project.

Furthermore, Xiang et al (2011) has had an influence on the choice of algorithms for this work, as they use K-NN (Cover and Hart, 1967), Random Forest (Breiman, 2001), Support Vector Machines (Microsoft Research, 1998), J48 (Quinlan, 1992), and an improvement to a previous Linear Regression classifier (Zhang et al., 2011). While linear regression functions work solely on mathematical attributes and therefore are unsuitable for this project, the use of the other algorithms within this paper lend further credence to the idea of using them within the context of this final year project.

## **Section 2.2. Technology review**

This section will look at the design and implementation of the project in terms of the existing body of methods, tools and techniques. In other words, a description of the methodology used within the project is first. Then, there is a review of the programming languages available and a rationale for the use of Java. Further review is then given to the frameworks and libraries available within the Java programming environment for use on this project, and then a review of the non-programming software packages that were used during the course of the project is given.

To summarize, the previous section contained a survey of the literature and theory, and this section contains a survey of the practice and technology.

### **Subsection 2.2.1. Methodology review**

This project is at its core a data mining project. The aim is to deliver a machine learning model capable of predicting whether or not a given website is a phishing site. However, software development is a significant part of the project too. The web crawler as mentioned in the objectives is software that will need to be developed, as well as the user interface. As such, a combined methodology is appropriate, one that draws on both data mining techniques and software engineering methods.

A common methodology in the field of data mining is the CRISP-DM methodology. (Smart Vision – Europe, 2016) In terms of this project, the business understanding and data understanding phases of CRISP-DM have already been carried out simply by having a project plan approved, and therefore the remainder of the project will incorporate the remaining phases.

These phases may be carried out in a cyclic manner reminiscent of the agile software engineering philosophy, which is what will be used for the software development parts of the project. Specifically the project will make use of a scaled-down variant of scrum (scrumalliance.org, 2016) in which ‘sprints’, scrum’s term for an increment of the system development, occur every few days to a week, as opposed to the 2-4 week sprints common in industry.

### Subsection 2.2.2. Languages and libraries review

This project was thought up with the intent of showcasing the understanding of various technologies acquired throughout the course of an undergraduate degree in Computer Science, and as such a broad range of technologies should be used, so as to demonstrate proficiency in multiple areas. This can be seen immediately from a high-level overview, since the project is a hybrid of data mining and software engineering. However, this does not mean that technologies that fit a particular role are merely chosen at random. Each library, language, tool and technique that has been used during the course of the project has been carefully examined to ensure not only its suitability for the task but also how well the given technology integrates with the rest of the project.

This can be seen with the choice of programming language. From the outset, the project was planned to have three distinct subsystems that provide necessary functionality – the web crawler and data gatherer to gather training data, the data mining package to create a model, and the user interface that allows users to create their own predictions. All three of these subsystems must integrate seamlessly with each other, so for ease of development the entire systems should be written in the same language.

This means the language of choice must be a general purpose language, with third party libraries and extensions that give all of this functionality. The language studied throughout the course in the greatest depth is Java, which is known for its large variety of third party libraries. Therefore, Java became a candidate language for this project.

In order to come up with new candidate languages, as well as compare the available candidates against each other, each subsystem must be examined in terms of the correct language for the job.

The first subsystem that will be written is the web crawler. When dealing with websites, a good choice of language is PHP (Php.net, 2017), which is designed specifically for writing websites. PHP also integrates seamlessly with HTML and CSS, allowing a solid web-based user interface to be written. Hence, PHP becomes a candidate language.

Next, there is the data mining model to consider. For this, a language that deals well with data models and statistics is likely to be a good choice, and chief among these languages is R (R-project.org, 2017). However, while R would be very suitable for the data mining subsystem, and does offer a third party library for building user interfaces in the form of Shiny (Shiny.rstudio.com, 2017), it does not offer any web crawling libraries.

Finally, there is the user interface to consider. All three of the candidate languages – PHP, Java and R – can build user interfaces, and the assumption is made that user interfaces can be built with the same ease for each language. As such, the contest comes down to which language can adequately support the remaining two subsystems. Languages which do not offer the necessary functionality are rejected.

As mentioned above, R does not possess any third party libraries that can do proper web crawling. Hence, R is disqualified as a candidate language. PHP and Java both have web crawling capabilities, so they remain. Considering the data mining model, Java does have a library that can do this – WEKA (Witten et al, 2011), which was used in the second year AIDM module, and as such meets the secondary objective of showcasing expertise developed during this degree course. PHP, on the other hand, does not have any functionality for doing data mining, and as such, Java was chosen as the language to build the system in.

Using Java as the language and WEKA as the data mining library means that libraries for the web crawler and UI can now be selected. For the web crawler, three libraries were examined – crawler4j (GitHub, 2015), Nutch (Nutch.apache.org, 2009) and Heritrix (Webarchive.jira.com, 2010). However, Heritrix has not been updated since 2010, and numerous changes to the web have occurred since then, HTTP 2.0 and HTML5 chief among them. This leaves Nutch and crawler4j as candidates. Of the two, crawler4j is multithreaded, whereas Nutch is single-threaded. This means that crawler4j can crawl multiple websites at once, cutting down on the time needed to crawl large numbers of websites, as will be needed for this project. As such, crawler4j was chosen as the library for the web crawler.

This leaves only the UI library, of which many are available – awt and its extension, swing, have been built into the core Java libraries since 1997. However, they have recently been replaced by the more recent JavaFX GUI libraries. As such, awt and swing are rejected as a GUI library. This puts JavaFX as a candidate library, but not the only one. GWT is a GUI library that compiles the written java code to javascript, effectively allowing web applications to be written in standard Java. Hence, the decision that must be made is between GWT and JavaFX. Of these two, JavaFX was chosen, as it is the simpler and lighter – GWT offers many pieces of functionality that would not be used on this project, and the ability to compile to javascript is not necessary – the user interface can be a simple desktop application.

Now that these libraries have been identified, the next step is to select other tools available that will be useful when working on this project, which is the topic of the next subsection.

### Subsection 2.2.3. Supporting technologies review

When writing applications that do not make use of third party libraries, the process for building Java applications is very simple; all that is necessary is to compile the code. However, things get more complicated when third party libraries are involved. The project depends on the third party libraries, and those libraries may depend on other libraries that the programmer is unaware of. As such, a dependency management tool is required, that will somehow manage this complex hierarchy of dependencies.

The tool that was used for this project is Maven (Porter, Zyl and Lamy, 2017), because the other common build tool, known as Ant (MacNeill and Bodewig, 2017) requires an extension in order to be able to do dependency management, which means that the one thing the project truly needs from a build tool is not a core part of the Ant specification.

Another technology that will be used is not part of the core of the project and therefore not strictly required, but important for security considerations. The websites that the computed attributes are gathered from also need to be visited manually to record the human attributes, and since a portion of them will be dangerous sites, security measures will need to be employed. However, in order to determine which sites are dangerous and which are not, the easiest way is to simply let the

dangerous code execute. In order to do that, it is necessary to run it in a sandbox system. The sandboxes that will give any possible malware space to activate, and hence reveal its dangerous nature, are virtual operating systems. As such, a virtualization software system will be used to view the websites.

There are two main competing virtual operating system packages – VirtualBox (Virtualbox.org, 2007) and VMWare (Vmware.com, 1999). In terms of providing a disposable operating environment and allowing websites to be thoroughly examined, both services are approximately equal in terms of how they meet the requirements for this project– both are free, and offer approximately the same performance levels – however, VirtualBox is easier to install and configure, running “straight out of the box”. Therefore, VirtualBox was selected as the virtualization software for this project.

## **Chapter 3. Requirements and design**

With all of the technology now in place, the next thing to do is to gather together system requirements, and create high-level designs for each of the three subsystems.

### **Section 3.1. The Web Crawler**

The web crawler has been designed to gather the computed attributes for a set of websites very rapidly. To get a picture of how a web crawler works, imagine the internet as one colossal graph, where web pages are vertices, and a page linking to another page denotes an edge. Web crawlers work by designating one or more vertices within this graph, doing some processing - in this case gathering the computed attributes - following edges from these vertices, and repeating with the new pages.

This functionality is inherited from the crawler4j library, so it does not need to be reimplemented. All that needs to be done is to implement the processing mentioned above.

#### **Subsection 3.1.1. Web crawler requirements**

The fundamental purpose of the web crawler is to gather the computed attributes from many websites in an automated manner. Hence, it must be capable of processing data from the websites visited, and following links to other websites in order to repeat the cycle.

Therefore, the web crawler will include methods to connect to a website, to compare its URL against a list of known benevolent URLs, to detect whether or not a redirection occurred and to count the number of spelling errors on the web page. Lastly, the crawler needs to be capable of writing these pieces of data to an ARFF file, for processing within WEKA. The actual crawling capability – the reason for using this construct in the first place – is inherited from crawler4j, so there is no need to reimplement this.

In effect, then, the web crawler has some very simple requirements – visit a website, gather the computed attributes, and write them in ARFF format to a file.

#### **Subsection 3.1.2. Web Crawler Design**

Now that the crawler requirements are fleshed out, the design part can take place. It is logical to divide up the gathering functionality into three separate functions, one for each attribute. Then, in the WebCrawler.visit() method inherited from crawler4j's WebCrawler class, this data will be accumulated and another method will be called, one that writes the collected data to a file.

Crawler4j requires the use of a controller class to set up and configure the crawler. This deals with things such as setting the depth of the crawl, what to do about robots.txt files – they mark pages on a website that should not be visited by a crawler, but can be ignored – as well as a crawler4j specific CrawlController class.

Furthermore, the data that the crawler gathers is likely to be passed around to numerous other classes, and as such it was packaged into a Data Transfer Object, or DTO, to make changes to the structure of the gathered data easier to make – for example if the human attributes needed to be added, or some attributes needed changing.

Lastly, the actual data gathering was decoupled into its own class that is then called from the crawler, so that in future, if data gathering is required in a non-crawling context, this change is easy to account for. This technically means that this subsystem is a combination of a web crawler and data gatherer, but for simplicity it is referred to as a web crawler unless otherwise required.

As such, it is now possible to write the UML for these 4 classes that constitute the web crawling package that will lay out the methods that must be implemented. The actual implementation will not be discussed at this time, as it will be discussed in chapters 4 and 6.

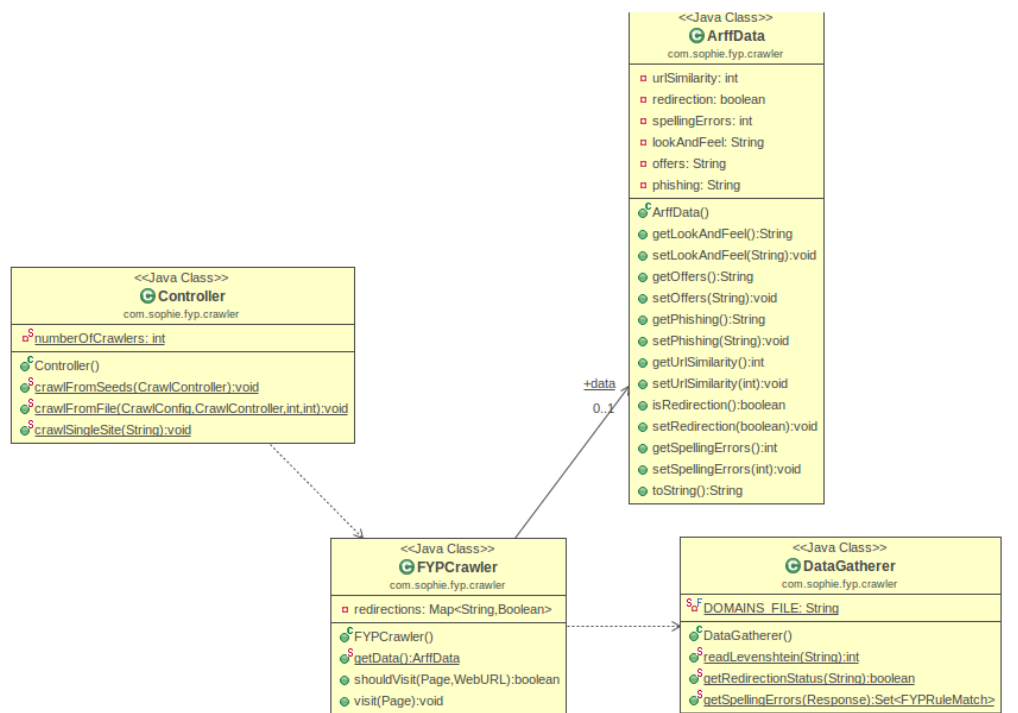


Figure 3.1.2a. the UML for the crawler package

The web crawling method suffices for easily gathering data from websites that are closely networked and reference each other, but the problem that arises is that phishing websites are very seldom referenced by non-phishing sites. Therefore it is unlikely that, starting from a benevolent website, the crawler will reach a phishing site of its own accord.

However, there are other ways of finding phishing sites. A common vector for phishing attacks is email – spammers send emails to targets pretending to require details for a legitimate organization. Studies have been done on how to build defenses against these emails (Ma et al., 2009) (Bergholz et al., 2010) and therefore repositories of phishing emails sent to honeypot addresses exist for public use.

The repository that has been used for this project is SPAM archive (Untroubled.org, 1998), and in order to obtain a list of phishing URLs, the text of emails was downloaded from this repository. Then, a BASH script was written that extracted the URLs from the text of the email, writing them out to a file. As such, the output of this script is a file containing all of the URLs present within the



emails that have been downloaded from this spam repository. The code of this script is given in Code 3.1.2b.

```
#!/bin/bash

for file in spamUrls/*; do

    url=$(grep -Eo 'https?://[^\s >]+' "$file")
    echo $file + ": " + $url
    echo $url >> results.txt
done

sed -i 's/\s\+/\n/g' results.txt
sed -i 's/<\/.*\/' results.txt
```

*Code 3.1.2b. the BASH script written to parse URLs from spam emails*

Once this collection of URLs was assembled, the crawler was then configured to visit each URL within this file in turn, instead of utilizing the interconnected nature of the internet to reach new sites. This is reflected in the UML given in Figure 3.1.2a, which shows a Controller class defining `crawlFromSeeds()` and `crawlFromFile()` methods.

### **Section 3.2. The Data Mining model**

Once the crawler described in the last section was completed and implemented, it was then used to gather data that will form the basis of the training set. A complete explanation of the process used to choose an algorithm for use during this project is given in Chapter 5, however this section will explain the need for a data mining model, how it will be used within the final software product, and review the metrics used for determining an algorithm to be the best for this situation.

On the surface, the need for a data mining model is obvious – the project is at its heart a data mining project, after all – so this section will detail what makes a ‘good’ model, and what other features are required in order to integrate smoothly with the final software product and WEKA.

The data mining model, as used in the final software, will be trained and tested on the gathered data. Users will provide their own data, and the model will then classify their specified websites.

Ideally, the model will then give a probability of the prediction being true or not, but only probability-based algorithms such as Naive Bayes offer this information. Therefore, it will only be possible to provide this information if the algorithm that is chosen provides it.

Furthermore, it is likely that users will want to visualize the data mining model, so as to see for themselves a justification of how the prediction was arrived at. WEKA offers a means of drawing data models for decision tree algorithms, but this has the same problem as the probability based algorithms – only a subset of the available classifiers implement this. Hence, these two features will be added if possible, but it is entirely possible that neither will be available.

With regards to the metrics used to identify a ‘good’ algorithm and separate these ‘good’ algorithms from ‘bad’ algorithms, the measure that will be used is the predictive accuracy, as that is the property that should be optimized in order to achieve the second half of the project aims – the

project requires accurate predictions. Getting a prediction wrong does not carry any severe penalty, since the project is only being used on a small scale, so measures such as counting cost of false positives versus false negatives do not really apply. If the system were to be scaled up, then the cost of generating an incorrect prediction would increase correspondingly.

Finally, there is the issue of testing. Traditionally, once data mining models have been trained using a training dataset, they are then tested using a separate, independent test dataset. However, the datasets that will be gathered using the web crawler will be small – the combined dataset resulted in 600 examples – and so another method is required.

Cross-validation (Kohavi, 1995) is a means of performing training and testing on the same dataset, by dividing the dataset into parts and training using all but one of the parts, testing using the remaining part, then repeating until all of the dataset has been used for testing. The average of these results is then calculated and given as the predictive accuracy for the model as a whole. Lastly, a model is trained again from the entirety of the data. Because of the lack of data available for this project, this is the method that will be used.

Now that these fundamental features of the data mining model have been decided upon, this chapter shall be concluded with a discussion of the requirements and design of the user interface.

### **Section 3.3. The user interface**

The user interface will provide users with the opportunity to nominate their own websites, and receive a prediction on whether or not the system believes the given website is a phishing website. The user will provide a URL, and values for the human attributes, and the user interface will then use the data gatherer described in the crawler package to collect the computed attributes for the given URL. Once the human and computed attributes are known, the data mining model described in the preceding section will make a prediction using this data. This prediction is then relayed back to the user.

#### **Subsection 3.3.1. User interface requirements**

Fundamentally, the user interface will need to provide a means for the user to enter a URL and the human attributes, and receive a prediction back. But there are also other things it could do. For example, if the data mining model allows it, then the user interface could either display the probability of a given prediction being correct, or allow for visualizing the model.

Furthermore, the act of processing a particular URL could take a substantial amount of time, during which the software appears to hang. Therefore, it would be more user friendly to add a progress bar and allow users to see exactly what the software is doing at a given moment during processing.

Lastly, but perhaps most importantly of all, there is the issue of accessibility to consider. Designing and implementing a user interface based solely on the requirements described here would mean making assumptions about the capabilities of the users of the software – the default font size may be insufficient to allow users with poor eyesight to adequately use the software, for example. Another example could be how the text colour and background colours combine to make the visual theme of the application – again, the default colour combination may not be suitable in terms of allowing users with colourblindness to make effective use of the software.

In order to work around these issues, it is important to embed accessibility concessions into the user interface right from this early stage. As such, the interface will contain controls to allow the user to change the font size, and a choice of several ‘themes’ - a combination of background colour and text colour, thereby changing the overall look of the software – as well as a comprehensive user manual.

With the issue of accessibility covered adequately, it is now time to consider how this user interface might be designed, and to present wireframe diagrams and flowcharts detailing typical use of the system as a whole.

### Subsection 3.3.2. User interface design

To begin the design of the interface, it is useful to go back to the list of requirements. The most essential part is that a user should be able to input a URL, choose values for the human attributes from a selection box, and start the process of classifying the website. Therefore, the interface will need a text entry field and two choice boxes, one for each of the human attributes.

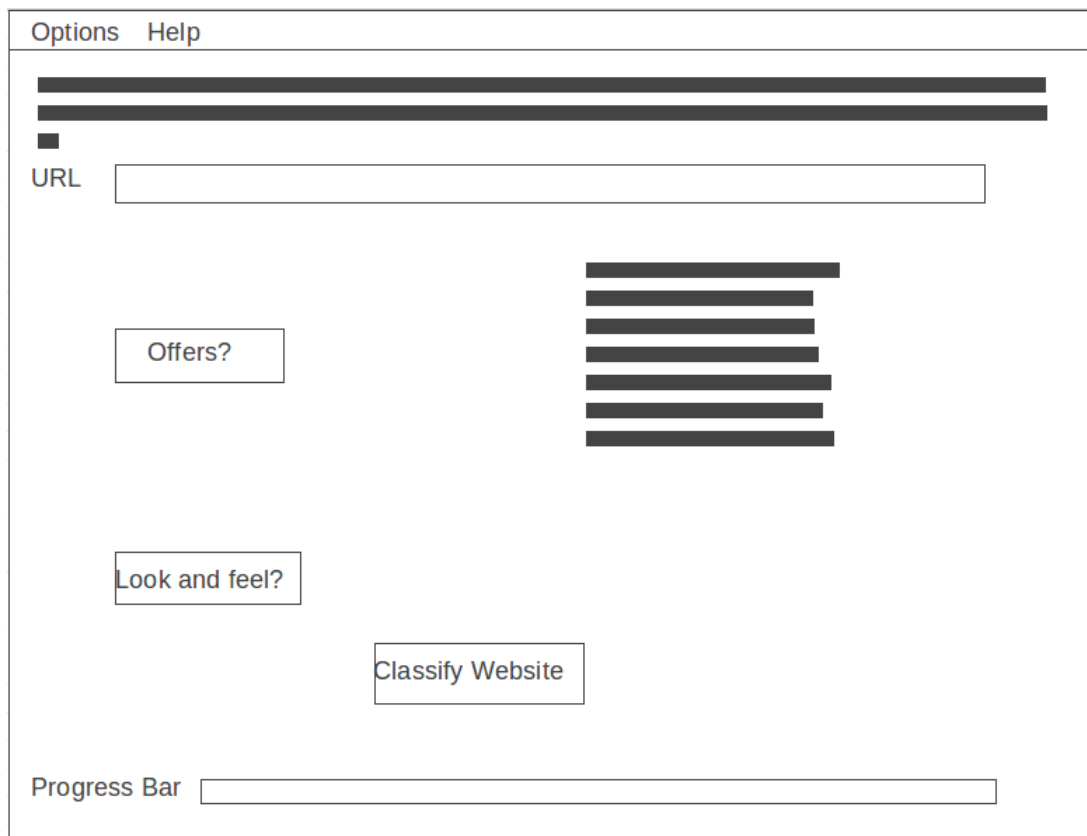
There will need to be a description of what the software does and how to use it in a prominent place, too. Hence, this should be above the URL box. Along very similar lines, the input boxes will need labels, explaining what is done with their input, and for the choice boxes, what the available options even mean. These should be brief, and positioned to the left of the input fields, like a web-based form.

The user will need to click a button to tell the software to classify the website. Therefore, there will need to be a button below the input fields, carefully labeled in order to explain what it does in a concise manner. Leading on from this, the prediction will need its own label, in order to convey the result to the user in plain English, as opposed to simply ‘true’ or ‘false’.

The progress bar should be at the bottom, near to the ‘classify website’ button, so that the user’s eye is naturally drawn to it after they click the button to start the software. In this way, they can receive live updates on what the software is doing during classification, and not get annoyed at the software if classification takes a long time and the software appears to freeze during that time.

Lastly, the accessibility concerns need dealing with. These are important to disabled users, but a default theme and font size must be chosen. The options to choose these things will be in an options menu, and the user manual will also be accessible through this menu. In order to not bombard users with too many choices, and to reduce the workload during the implementation phase, four font sizes will be made available, and three themes. The exact nature of these things will be detailed in Chapter 6.

Now that some of the specifics have been fleshed out, a preliminary wireframe diagram of how the interface will look can be created, and is shown in Figure 3.3.2.



*Figure 3.3.2. The wireframe model for the user interface*

However, the user interface is the last subsystem to be implemented chronologically. Therefore, the focus of the next chapter shall be the implementation and testing of the web crawler and data gathering components.

## **Chapter 4. Web crawler implementation and testing**

When implementing a web crawler to do a specific task – in this case, gather the computed attributes and write the data to a file in ARFF format – it is important to recall the technologies that were decided upon in Chapter 2 of this dissertation. Java was the chosen programming language, and it contains built-in facilities to write data to a file. Hence, no additional code needs to be written for this, either specifically for this project or in third-party libraries.

The web crawler software that was chosen is crawler4j, which gives crawling functionality to classes derived from WebCrawler. Hence, all that is necessary is to write the code to gather the computed attributes within the visit() method.

### **Section 4.1. Writing the code**

There are three computed attributes that need to be implemented, and each one comes with its own challenges and constraints. Hence, each gets its own subsection, to adequately explore and explain the issues faced when implementing this feature. Firstly, the ‘URL similarity’ attribute is considered.

#### **Subsection 4.1.1. URL similarity to list of benevolent sites**

Early in the project, a list of websites that are known to be benevolent is compiled. This list is, by definition, incomplete – if a complete and comprehensive list of benevolent sites could be compiled, this defeats the purpose of the entire project. Therefore, the list is created incrementally, and contains only the most popular sites that phishers are likely to try and imitate, such as Amazon, Google, Ebay, or banking websites. This list is added to manually throughout the course of the project, if new candidates for comparison come up.

The reason for making this comparison is that phishing works by imitating a company and using the victim’s trust in that company to get them to trust the phisher, and so they might copy the company website, and give it a URL similar to the URL of the real company, so ‘twitter.com’ might become ‘tvvitter.com’.

Hence, the question becomes “how do we compare one string against another, in terms of similarity?” Java offers built-in functionality for comparing two strings, but only for finding out if they are identical or not.

Luckily, there are several ways of implementing a similarity metric, also known as “fuzzy string matching”. Perhaps the simplest is the Damerau-Levenshtein Distance (Damerau, 1964) which is the number of single-character additions, deletions, transpositions or edits necessary to change one string into another. Another fuzzy matching algorithm is SoundEx (Odell, 1956), based on the principle of comparing words based on how they sound. Unfortunately, as URLs are not dictionary words, this algorithm is not suitable. Hence, the Damerau-Levenshtein distance is the metric that was used to implement fuzzy matching of URLs, in order to compare their similarities.

Once an algorithm was decided upon, the implementation became a relatively simple matter. Damerau-Levenshtein expresses output in the form of a “distance measure” between the two strings passed as input, and the smallest distance is desired. Hence, a ‘minDifference’ value was initialized, defaulting to the maximum 32-bit integer value, ( $2^{32} - 1$ ), or 2,147,483,647. On visiting a website, the URL would be obtained and compared to each URL in the list, one at a time. Each comparison

produces a “distance value” as defined above by Damerau-Levenshtein. If any of these values are less than the current minimum, then it becomes the new minimum. In this way, the value that will be used as the attribute – the distance value for the most similar URL and therefore the one a phishing site is most likely to be trying to emulate – can be extracted.

The source code for this method is given in Code 4.1.1. It describes how URLs are extracted from a file and compared against the current value in sequence to obtain the minimum value.

```
public static int readLevenshtein( String url) throws IOException
{
    BufferedReader reader = new BufferedReader(new
FileReader(urls_FILE));
    int lowestDistance = Integer.MAX_VALUE;
    String line = reader.readLine();
    while(line != null)
    {

        int levenshteinDistance =
StringUtils.getLevenshteinDistance(url, line);
        if(levenshteinDistance < lowestDistance)
        {
            lowestDistance = levenshteinDistance;

        }
        if(lowestDistance ==0)
        {
            break;
        }
        line = reader.readLine();

    }
    reader.close();
    return lowestDistance;
}
```

*Code 4.1.1. source code for the URL similarity method*

#### Subsection 4.1.2. Redirection detection

The second of the three computed attributes is much simpler conceptually, but required the most work in order to get right. The idea is that if a URL linked to by a hyperlink within a web page or email claims to lead somewhere, and a redirection to another web page occurs, then that is a potential notifier of a phishing site, since the people who created that hyperlink intended for it to lead to a different place than would be otherwise apparent from the hyperlink itself.

There exists a setting within crawler4j’s WebCrawler class, which is the superclass of the web crawler that is being created for this project, to automatically follow any redirections or not. Unfortunately, this setting does not allow for detection of redirection, which is the real objective for this attribute. Initially, what was done to detect redirections is to set crawler4j to not follow any redirections, and if the initial connection gave an error code of the form 3XX – the code block that denotes redirections – get the URL that was redirected to, connect to that, and gather the other two attributes.

Unfortunately, this method did not initially work, because the page parsing library, Jsoup (Hedley, 2017), was the one that actually needed setting to not follow redirections, since this was what was being used to detect redirections since both standard Java and crawler4j do not offer this functionality.

Later on in the project, due to some differences between the environment when running unit tests and when crawling live sites, this method started to fail for the unit tests. The problem was that the web crawler crawls a page, adds any hyperlinks on that page to a list of pages to crawl, then starts crawling them. Due to the multithreaded nature of crawler4j, these activities could be interleaved, causing problems that do not exist when testing on a single website. Therefore, a new method of detecting redirections on connection to a website needed to be implemented.

The solution was to maintain a Java HashMap mapping URLs to redirection statuses, try connecting once with redirection following turned off, store the result in the map, then – at a later, unknown stage due to tasks executing concurrently – if the redirection status is true, try to connect again to the website using the redirected URL.

The net result of this is that redirection status is detected, but redirections are then followed – exactly what is required for this project. Once the website is connected to, the third and final computed attribute can be collected – the number of spelling errors on the page.

#### Subsection 4.1.3. Spelling errors counting

The most pressing thing to think about when considering how to count the number of spelling errors on a web page is what library to use for it. English spelling and grammar rules are notoriously complicated even among other Germanic languages, and implementing a spell-checker in code could well be beyond the scope of a Final Year Project by itself. As such, a pre-written spell checking library was required for this section.

The one that was chosen is the Java API for LanguageTool (Wiki.languagetool.org, 2017), which is a spelling and grammar checking extension for LibreOffice and OpenOffice. It operates in terms of rules that can be enabled and disabled for a customized level of checking. This works very well for this project, as only the rules relating to spelling checks can be turned on, and any other rules that the tool implements can be safely ignored.

There is one problem, however. LanguageTool uses dictionary definitions of words as the basis for misspellings, and many words that are not in dictionaries appear many times on a webpage – for example, “firstyearmatters” is not a misspelling, but it is not in the dictionary, and it appears many times on the firstyearmatters web page. As such, in an attempt to mitigate these false positives, a list of all the rule matches was gathered, and put into a set to remove any duplicates.

Unfortunately, the default implementation of the RuleMatch class defines object equality differently to how it is defined for this project – each match is given its own unique ID – so what was needed was a subclass of RuleMatch that redefines the equals() method used to test for equality, to define two instances of FYPRuleMatch as equal if the list of suggested replacements contained all the same elements. Two different misspelled words will have different lists of suggested corrections, so two words that are misspelled the same will flag as equal, and will therefore be only allowed into the set once.

This final set of matches is then returned, and its size used as the final computed attribute. The source code for this explanation is given in Code 4.1.3.

```
public static Set<FYPRuleMatch> getSpellingErrors(Response response) throws
IOException
{
    Document doc = response.parse();

    String text = doc.text();
    JLanguageTool jlang = new JLanguageTool(new BritishEnglish());
    for (Rule rule : jlang.getAllRules()) {
        if (!rule.isDictionaryBasedSpellingRule()) {
            jlang.disableRule(rule.getId());
        }
    }
    List<RuleMatch> matches = jlang.check(text);
    Set<FYPRuleMatch> fypMatches = new HashSet<>();
    for (RuleMatch match: matches)
    {

        fypMatches.add(new FYPRuleMatch(match));
    }

    return fypMatches;
}
```

*Code 4.1.3. Implementation of a spelling errors counter using LanguageTool*

Now that the implementation of the collection of all three of the computed attributes has been discussed, the next step is to merge everything together, and to call all of these methods from within the crawler, as well as write out these values in ARFF format to a file. Once this file has been populated with examples, the data mining can begin.

But before this, it is important to make sure that the crawler's methods all work correctly, so unit tests were written to check for any errors. These tests are the subject of the next section.

## **Section 4.2. Writing unit tests**

There are three attributes that need to be tested, and therefore six unit tests is the logical place to start. Three for testing for a positive outcome, and three for testing for a negative one. However, the reality ended up being slightly different, due to multiple potential failure conditions – a website connection could fail for any number of reasons, and impossible values need to be tested for to confirm there are no bugs in the crawling code.

### **Subsection 4.2.1. URL similarity unit tests**

The result of the URL similarity method is an integer value representing the edit distance of the given URL to the URL in the stored file that is most similar to it. This number cannot be negative under any circumstances – calculation terminates early if the minimum value undergoing calculation is 0. As such, testing that the result is not negative is a useful test to do, as well as a positive test to check that values are equal or above 0. There is no meaningful distinction to be made on whether this value is 0 or greater – merely a boundary that the acceptable results should not cross.



What is important to remember here is that no connection to the website undergoing crawling must be established in order to collect this attribute; only the URL is necessary. Hence, there are no other tests that can be written for this functionality.

#### Subsection 4.2.2. Redirection detection unit tests

Unlike the URL similarity attribute described above, connecting to the website is necessary in order to discern whether or not the website causes a redirection. As such, a test where connection fails due to a URL that does not link to a valid website was written, to account for this possibility.

Of the tests that do rely on a successful connection to a real website, both a positive and a negative test were written. The negative test, showing that a website that does not cause a redirection is indeed flagged as such, linked to ebay.com. For the positive test, a URL shortening service was used to take the same ebay URL and turn it into a different one, thereby adding a redirection to the process of connecting to ebay's website.

#### Subsection 4.2.3. Spelling errors unit tests

Like the previous attribute, counting the number of spelling errors depends on a valid connection to a site that actually exists. As such, that test could be copied over from the redirection test suite and into the suite for this attribute. Unfortunately, once a connection to a valid website can be assumed, the method is all but guaranteed to return a valid value for the number of spelling errors – 0 is within the acceptable range, and the number cannot be negative. As such, a simple positive test was written, testing if the number of spelling errors is greater than or equal to 0.

Now that the unit tests have been covered, the next stage in the project is to discuss the creation, training and testing of the data mining model, and the process by which an appropriate algorithm was selected for inclusion in the final product.

## **Chapter 5. The data mining model – choosing an algorithm**

The data mining model is the centrepiece of the entire project, and deserves considerable attention paid to it. Firstly, there will be a short section on how the data used to train and test the model was gathered using the web crawler covered in chapters 3 and 4 as well as a human expert. After this, the preliminary data mining performed using simple algorithms that were studied during the AIDM module is described, as well as experiments done to the parameters of these algorithms in an attempt to hone and optimize the predictive accuracies.

This is followed by further mining done using algorithms found from surveying the related literature. Once results are obtained from these algorithms, then experiments are done to vary algorithm parameters and try to refine the predictive accuracies, in an attempt to find the most appropriate algorithm. Finally, the results of these experiments are considered within the context of the rest of the project, and a final choice of algorithm is arrived at using statistical analysis.

### **Section 5.1. Gathering the data**

In the previous chapters, the design, implementation and testing of the web crawler was discussed. The class attribute is labelled “isPhishing” and hence there are two classes within the data to gather – yes and no. A website can either be phishing or benevolent. With that in mind, the crawler Howas set to work, crawling benevolent data and gathering phishing data from the file of phishing URLs mentioned in Subsection 3.1.2.

The result of this data gathering was an ARFF file containing 600 examples, but with only the computed attributes filled in. A human expert then went through this file, manually visiting each website and using their own judgement to provide the human attributes.

Despite the project objectives saying that the ratio of benevolent to phishing sites should be approximately 5:1, the final dataset contained a ratio closer to 3:1. This was done because initially, several smaller datasets were compiled in order to test the effect of different ratio on the predictive accuracy. However, in the end it was decided to merge the collected datasets together, as a larger training dataset gives a model that is more representative of the universe of examples.

Once this data was gathered, the next step was to try mining it. Initially, only tentative efforts at generating a suitable model were made, using simple algorithms such as the ones learnt in the AIDM module in the second year.

### **Section 5.2. Preliminary data mining**

This section will detail the algorithms that were used right after data gathering, their predictive accuracies on the collected dataset using WEKA’s default values for any parameters, and any experiments that were done to vary the parameters in an attempt to push the predictive accuracies higher.

Four algorithms were used at this stage. These are ZeroR, OneR (Holte, 1993), Naive Bayes (Bayes, 1763) and K-nearest neighbour (Aha, Kibler and Albert, 1991).

ZeroR is a very simple algorithm – it will always predict the majority class. For example, if a dataset contains 200 websites that are phishing and 400 that are benevolent, then ZeroR will always predict benevolent, with a 66.6% accuracy rating.

OneR builds a one-level decision tree, considering a single attribute. As an example, if the attribute 'offers' is selected, then the rule list from that decision tree could be IF offers = 'yes' THEN phishing = 'yes', IF offers = 'no' THEN phishing = 'no'. Numeric values are discretized into a set of intervals to fit this structure.

Naive Bayes uses Bayesian probability theory to generate a posteriori probabilities for each class value, then predicts the one with the highest probability. It is called naive because it considers each attribute to have equal importance when generating the final prediction, when this is very often not the case, i.e when the value of one attribute depends on the value for another attribute.

K-nearest neighbour is what's called a 'lazy learner' because it does not generate a model. Instead, to classify a new instance, it takes the K 'nearest' other instances, according to the euclidean distance measure, and takes a majority vote on their class values to predict the new instance.

#### Subsection 5.2.1. Initial experiments

At this stage, two experiments were done. The first was to run all of these algorithms, with their default parameters, on the collected dataset and analyze the results. The second is to vary some parameter of the algorithm, specifically the K part of K-nearest neighbour, in the hopes that a greater number of other instances in the vote leads to a better solution.

In order to do the first experiment, a hypothesis is needed. For this stage, the hypothesis shall be that the more complex algorithms such as naive bayes and K-nearest neighbour will outperform the overly simplistic algorithms, OneR and ZeroR. Therefore, the prediction shall be that Naive Bayes shall outperform K-nearest neighbour, and OneR shall outperform ZeroR. All parameters were at their default values within WEKA.

ZeroR simply predicts the majority class value, which in this dataset is no.

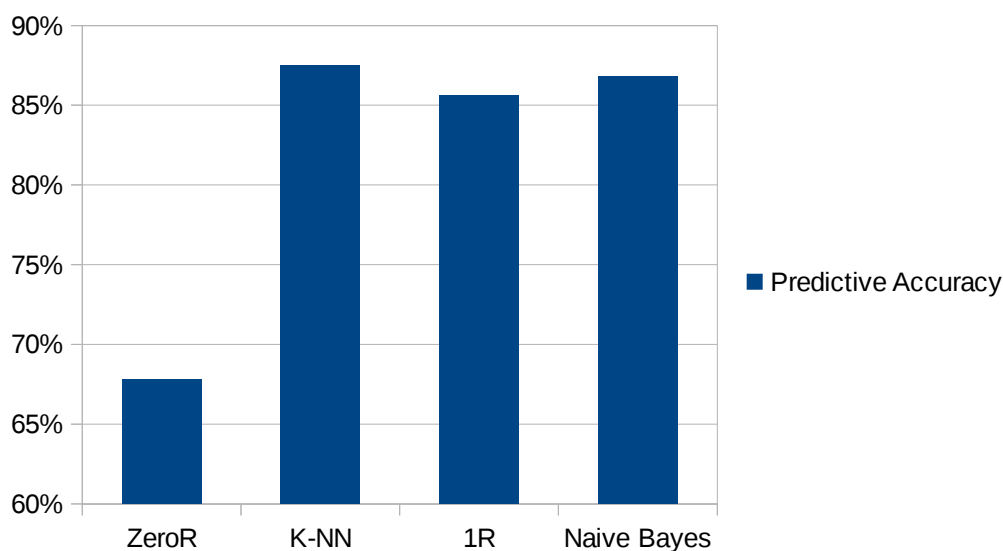
OneR generates a one-level decision tree, choosing a single attribute and predicting based on that. This tree can be converted into a list of rules, so the model that is generated is:

```
IF offers = yes THEN isPhishing = yes
IF offers = no THEN isPhishing = no
IF offers = ? THEN isPhishing = no
```

Naive Bayes uses the mean and standard deviation for the distributions of all attribute values to generate a prediction, and predicts isPhishing = yes with 32% probability, and isPhishing = no with 68% probability.

K-nearest neighbour does not generate a model.

From these models, particularly the OneR model, it can be seen that the 'offers' attribute appears to hold greater weight than the others when predicting phishing websites.

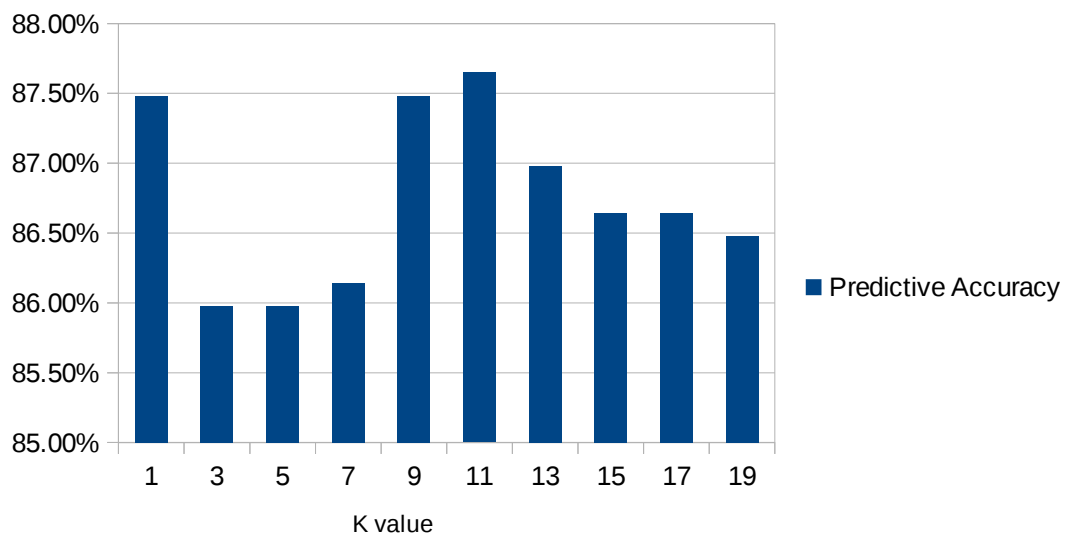


*Graph 5.2.1a. The predictive accuracy for the preliminary data mining*

Looking at these results in terms of the original hypothesis, it can be seen that OneR has, in fact, beaten ZeroR. This is to be expected, because of ZeroR's simplistic nature. What is more interesting is the fact that K-nearest neighbour has a higher predictive accuracy rating than Naive Bayes, and in fact has the highest predictive accuracy out of all of the algorithms, as this is the opposite of the hypothesis. This could be because of Naive Bayes's tendency to treat each attribute as equally important when this is likely not the case, or perhaps there is a property of K-nearest neighbour that is particularly suited to this dataset.

Nonetheless, it is clear from these preliminary results that K-nearest neighbour is a promising choice of algorithm. However, its default value for K is 1, which means that the algorithm is making predictions based purely on the single "closest" (in terms of euclidean distance) value. Therefore, another experiment can be devised based on varying the value of K, in an attempt to optimize the predictive accuracy further.

In this new experiment, it is only K-nearest neighbour that will be considered. The predictive accuracy is likely to increase up to a certain maximum, as noise in the data is compensated for at higher values of K. Furthermore, the predictive accuracy is likely to drop at first, because there is likely to be noise in the data that gets overly compensated for at low, non-1 values of K. For full control over the experiment, all variables will be the same throughout, except for the value of K.



*Graph 5.2.1b. The predictive accuracy for varying values of K in K-nearest neighbour*

These results confirm the second part of the hypothesis – the predictive accuracy does dip at first before rising again – but not really the first part. The predictive accuracy peaks at slightly above the accuracy for  $K = 1$ , but just as rapidly declines again. Therefore, the conclusion can be that noise is not as much of an issue in this dataset as had been first suspected, because if it was then the predictive accuracy would increase after the initial dip, as noise is compensated for by votes of a progressively greater size.

However, despite the hypothesis being disproven, the true goal of the experiment – to find a value for  $K$  that gives a higher predictive accuracy than the default of 1 – has been achieved with a real predictive accuracy of 87.65%, and this new value is the one that will be compared against the later algorithms.

87.65% is a very good starting value. In an attempt to push this even higher, research papers in this area were sought after and read, in the hope that the algorithms used in these other studies will lead to a higher accuracy. These algorithms, and the experiments derived from them, are the topic of the next section.

### **Section 5.3. Further data mining**

In the previous section, a brief description of how each algorithm works was given before the experiments, as a way of providing context for the discussion and experiments. The algorithms that were found and used on the dataset at this stage are considerably more complicated than the simple ones used before, and therefore the descriptions may be vague or incomplete. Nonetheless, enough information shall be given to inform the discussion and experiments performed using these algorithms.

The algorithms that were found include Random Forest (Breiman, 2001), Support Vector Machine (Microsoft Research, 1998), J48 (Quinlan, 1992) and Bayesian Network. A brief description of these algorithms follows.

Random Forest is based on the notion of a ‘forest’ of randomly generated decision trees, trained on the problem at hand. In order to generate a prediction, each of these trees generates its own prediction, and the predictions are averaged together to get the final result.

Support Vector Machine is a heavily mathematical algorithm that considers each instance as a point in space, then attempts to draw dividing lines, grouping each instance into similar clusters that are then used as the basis of classification. WEKA implements John Platt’s Sequential Minimal Optimization enhancement to the base algorithm.

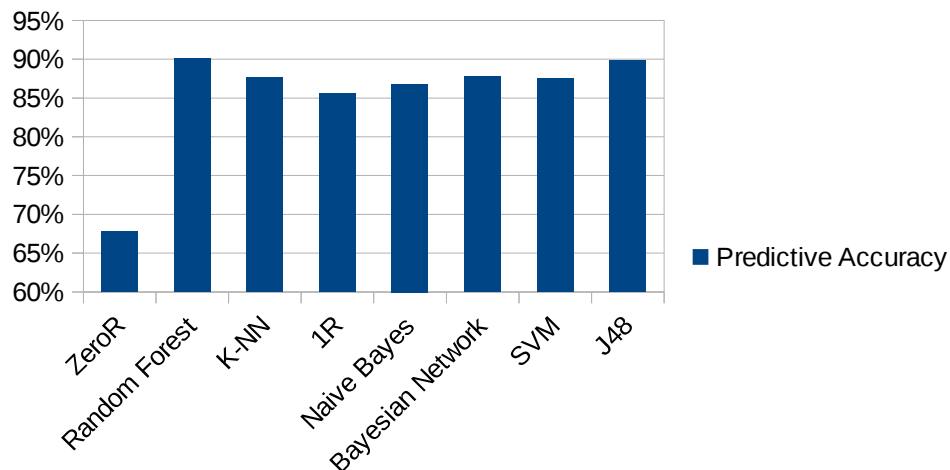
J48 is a complex decision tree learner, and an open source implementation of the earlier C4.5 algorithm, which is in turn an extension of the ID3 algorithm (Quinlan, 1986). C4.5 uses ID3’s concept of entropy to build a decision tree, but goes one step further with the addition of information gain to further refine the method by which the tree is split, and reduce the problem of overfitting.

A Bayesian Network is a probability-based algorithm to generate a directed acyclic graph to determine probabilities of class values given the attribute values. There are many implementations available – the default implemented within WEKA, and the one used during these experiments, uses a simple estimator for estimating learned probability tables within each node of the graph.

#### Subsection 5.3.1. Experiments with new algorithms

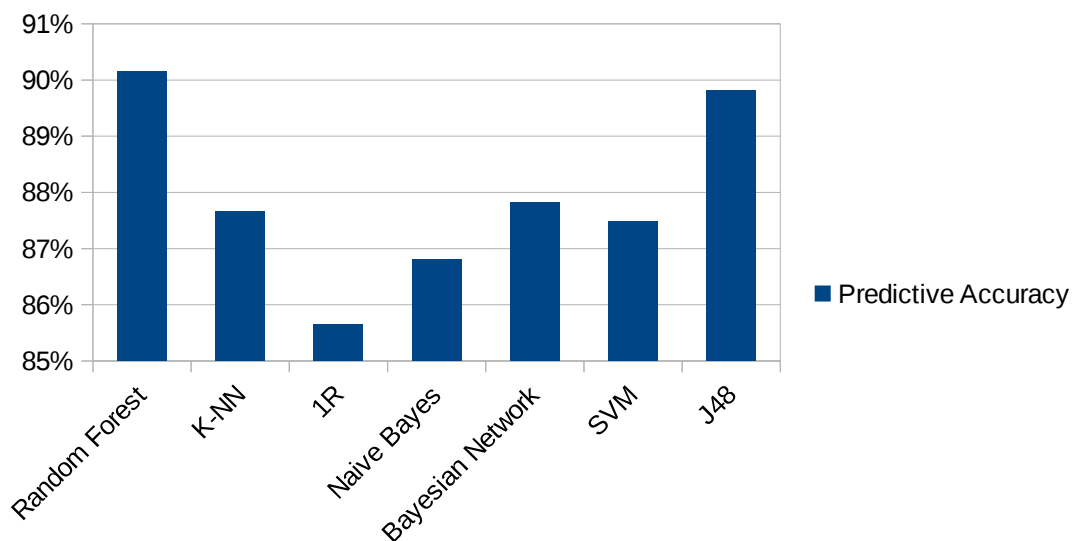
At this point, the results that these algorithms give on the gathered dataset can now be written up as an experiment. Since the goal is specifically to find an improvement to the algorithms used in the preliminary data mining stage, and in the previous stage more complicated, refined algorithms gave improved results over the more rudimentary ones, the hypothesis for this experiment will be that the more complex algorithms given here do in fact improve on K-nearest neighbour. Precisely pinpointing these new algorithms in terms of their complexity is much more difficult than it was for the earlier ones however, and so the hypothesis cannot be further refined.

What can be done is to look at the results, shown in Graph 5.3.1, and analyse them for clues on complexity. All results are using the default values, except for K-nearest neighbour which is using the optimal value discovered in Subsection 5.2.1.



*Graph 5.3.1a. The predictive accuracy of all algorithms.*

Now that we have the additional data points from using more algorithms on the dataset, it is clear that the value generated from ZeroR is an outlier, and therefore can be disregarded in order to focus on the remaining values, and hopefully optimize them further.



*Graph 5.3.1b. A more detailed graph, without the ZeroR outlier.*

This graph gives a much clearer picture of what is going on. The simpler 1R algorithm trails behind, and even K-nearest neighbour – the algorithm that gave the highest predictive accuracy of the original, simple algorithms – has been beaten by all of the newer algorithms. The reasons for this could be to do with their complexity – perhaps the simpler algorithms overfitted to the training data, and therefore performed poorly during the testing phase whereas the more advanced algorithms had measures in place to prevent this – but this explanation offers no understanding of why the more advanced algorithms ranked in the order they did.

What can be gleaned from these results – without an in-depth understanding of the advanced algorithms – is the two decision tree-based learners, J48 and Random Forest have achieved the top two predictive accuracies, with Random Forest at the very top. Since Random Forest gets its name from the ‘forest’ of randomly generated decision trees it creates, then perhaps varying the number of trees in the forest can optimize this result even further, pushing the predictive accuracy above even 91%?

At this stage, however, it is worth taking a step back and thinking about the system from a user’s perspective – the experiments detailed in this chapter to optimize predictive accuracies have given very small increases – under a tenth of a percent in the case of the K-nearest neighbour optimization experiment. As such, perhaps using an algorithm with a lower accuracy rating will be better, if said algorithm has other properties that would be useful to the user.

In Section 3.2 of this dissertation, the concept of visualizing the model to the user was briefly discussed, as a means of justifying the predictions generated on unseen data. WEKA offers a built-in method for doing this in the Drawable interface that certain algorithms implement, but the issue with that is that only certain algorithms implement it. Random Forest, the current highest scorer in terms of predictive accuracy, does not. However, its closest contender, J48, does – this makes sense, as J48 generates a decision tree and Random Forest generates a whole host of them – and therefore, if the predictive accuracy penalty is small enough so as to be inconsequential, then using J48 in the final model instead of Random Forest would in fact be better, due to the ability to visualize the tree.

In order to determine if the predictive accuracy tradeoff is worth the visualization ability, a statistical approach is necessary to discover if the differences between the two algorithm’s predictive accuracies is statistically significant.

#### **Section 5.4. Statistical analysis of algorithms**

The results generated from the earlier experiments came from just one iteration of stratified ten-fold cross-validation (Kohavi, 1995), which was done by dividing the dataset up into 10 separate parts. Therefore, it is entirely possible for different results to come from partitioning the dataset in different places. In order to truly judge the algorithms in terms of statistical significance, something called a paired t-test is done on the results mentioned above.

A paired t-test is a kind of statistical test that determines if two sets of measured values differ from each other in a statistically significant way, assuming that the two sets are normally distributed and independent. This makes it suitable for analysing the results generated from the above experiments, and judging if the difference between two algorithms is statistically significant.

Specifically, a decision on whether J48 and Random Forest differ in predictive accuracy in a statistically significant way is sought. If the two algorithms are not significantly different, then the difference between the values generated in the above experiments is considered to be the product of random chance, whereas if the difference is statistically significant, then J48 is considered to be a lower performer than Random Forest.

Fortunately, WEKA offers the means to perform a paired t-test through the use of its experimenter interface. It also allows for multiple comparisons against a single algorithm to be done in one go, allowing for all of the algorithms to be compared against Random Forest in a statistical way.



Figure 5.4 shows the results of this paired t-test, as shown in the WEKA experimenter interface.

Test output									
Tester: weka.experiment.PairedCorrectedTTester -G 4,5,6 -D 1 -R 2 -S 0.05 -result-matrix "weka.experiment.ResultMatr									
Analysing: Percent_correct									
Datasets: 1									
Resultsets: 8									
Confidence: 0.05 (two tailed)									
Sorted by: -									
Date: 04/04/17 16:19									
Dataset	(4) trees.Ra	(1) rules	(2) rules	(3) trees	(5) lazy.	(6) funct	(7) bayes	(8) bayes	
websites	(100) 89.83	67.78 *	85.54 *	89.38	87.08 *	87.48	87.75	87.06	
	(v/ /*)	(0/0/1)	(0/0/1)	(0/1/0)	(0/0/1)	(0/1/0)	(0/1/0)	(0/1/0)	
Key:									
(1) rules.ZeroR '' 48055541465867954									
(2) rules.OneR '-B 6' -3459427003147861443									
(3) trees.J48 '-C 0.25 -M 2' -217733168393644444									
(4) trees.RandomForest '-P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1' 1116839470751428698									
(5) lazy.IBk '-K 1 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A \"weka.core.EuclideanDistance -R first-last\"									
(6) functions.SMO '-C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K \"functions.supportVector.PolyKernel -E 1.0 -C 250007\"									
(7) bayes.BayesNet '-D -Q bayes.net.search.local.K2 -- -P 1 -S BAYES -E bayes.net.estimate.SimpleEstimator -- -A 0.5' 74									
(8) bayes.NaiveBayes '' 5995231201785697655									

Figure 5.4a. paired t-test results against Random Forest

Some of the results have an asterisk by them, and some do not. The asterisk denotes that the difference in the predictive accuracy is statistically significant, i.e. not merely the product of random chance. For example, the result for K-nearest neighbour, denoted by the number 5 in the table, has an asterisk whereas Support Vector Machine, denoted by the number 6, does not. Therefore, K-nearest neighbour statistically gives poorer results on this dataset than Random Forest, but the difference in results between Random Forest and Support Vector Machine can be chalked up to random chance.

From this table of results, it is possible to see that the confidence level is set to 0.05, but is two-tailed. This means that the confidence level is 5%, but the results could be higher or lower than the baseline – Random Forest – so there is 90% certainty of statistical significance, which is more than sufficient for this project.

Since J48 does not have an asterisk next to its results, this means that the results it gave during this experiment are not statistically significantly different from the ones given by Random Forest, with 90% certainty. As such, and taking into account the fact that J48 allows for visualization and Random Forest does not, J48 was chosen as the final algorithm to implement within the system. Figure 5.4b shows the model generated by J48, as shown by WEKA.

The tree is to be read as follows (this example is taken from the first line):

IF offers = yes THEN phishing = yes (133.89 instances reach this node, 13.67 of these are misclassified).

The reason for the decimal values is that when an attribute value is missing, the J48 algorithm ‘splits’ the instance up into parts proportional to the number of non-missing values for that attribute, so less of a whole instance is counted as reaching that node.

```

offers = yes: yes (133.89/13.67)
offers = no
| redirection = true
| | spellingErrors <= 5: yes (26.78/5.0)
| | spellingErrors > 5
| | | spellingErrors <= 15: no (7.0)
| | | spellingErrors > 15: yes (3.0/1.0)
| redirection = false
| | spellingErrors <= 4
| | | lookandfeel = identical
| | | | urlsimilarity <= 7
| | | | | urlsimilarity <= 3
| | | | | | urlsimilarity <= 0: no (5.0)
| | | | | | urlsimilarity > 0: yes (2.0)
| | | | | urlsimilarity > 3: no (11.25)
| | | | urlsimilarity > 7: yes (16.2/3.2)
| | | lookandfeel = veryclose
| | | | urlsimilarity <= 6: yes (4.0)
| | | | urlsimilarity > 6: no (9.17/2.0)
| | | lookandfeel = similar: no (32.42/4.0)
| | | lookandfeel = different: no (24.32/4.0)
| | | lookandfeel = unique: no (32.42/1.0)
| | spellingErrors > 4: no (291.55/19.0)

```

*Figure 5.4b. The tree generated by J48*

From this model, it can be seen that ‘offers’ is considered a very important attribute, with a prediction potentially being made based purely on the value of this one attribute. This is consistent with what was found during the previous set of experiments, particularly with OneR.

## **Chapter 6. User interface implementation and testing**

Now that the algorithm choice has been finalized, it can be used within the user interface. Therefore, it is now time to build this interface, integrate the crawler and data mining model into it, and test it to ensure that predictions can be generated, and that the system fails gracefully where an incorrect state is achieved.

### **Section 6.1. User interface implementation**

This section will discuss how the user interface was implemented, based on the requirements analysis and design done in Chapter 3. An overview of the use of JavaFX is presented first, then details of the use of concurrency within JavaFX, why it was used and how.

The point of the user interface is to allow users to specify their own websites, and receive a prediction on whether their nominated website is phishing or benevolent. As such, the interface should be easy to use, and accessible to as wide a range of people as possible.

Figure 3.3.1 on page 18, the wireframe diagram for the user interface, shows that there will be a menu containing program options and a help screen at the very top, then a brief description of what the system does and how it is used. In order to enter values into the system, there is a labelled textbox for entering the URL of the website to obtain a prediction for. Below that, there is two drop down lists for entering the two human attributes. To the right of these dropdown lists, there is space for the prediction to be displayed. At the bottom, there is the button that generates a prediction based on the inputs, and a progress bar so the user can see what's being done.

Fortunately, JavaFX offers classes for each of these objects – MenuBar, Menu, Label, TextBox, ChoiceBox, ProgressBar and Button are all pre-implemented, and ready to use. Therefore all that needed to be done to build a user interface was to specify where everything should go, and how certain elements should react when the window size changes.

JavaFX does not build in responsive design as standard. Everything is static, and has a fixed position and size. Therefore, responses to window size changes need to be programmed specifically. What was done for this was to identify specific UI elements, and attach a change listener to the main window's width property, so that when the screen width changes, the URL bar, progress bar and prediction label will all change with it. Only these specific elements were changed, because if the screen size gets to the point where the choice boxes need to respond then the system is already as good as unusable, since the label showing the results of the prediction will not be on screen.

In order to deal with height changes, the entire user interface was wrapped in a ScrollPane, which automatically creates a scrollbar if any UI element falls off the edge of the screen.

Lastly, the menu needs to be discussed. The menu as shown in Figure 6.1 shows two menus, labelled 'Options' and 'Help'. The Options menu contains the accessibility features, whereas the Help menu contains the help screen. The help screen is simply a document written in HTML, and loaded into a JavaFX WebView to display it on screen. The options menu is more involved.

In Subsection 3.3.2, the issue of accessibility was discussed, and two means of making the system more accessible were derived – changing the font size, for the hard of sight, and changing the system theme, for the colourblind. For the font size, four different font sizes were offered – 11, 13

(the default), 16, and 18 point. These font sizes were carefully chosen to allow for a variety of experiences, while still allowing the labels to fit in their allocated areas. For the theme changes, three choices were made available. Taking inspiration from Visual Studio and its three themes – light, blue and dark – those same three were offered in this system too.

Now that the user interface has been developed, the next stage is to make it do something. This involves integrating the data mining model and the data gathering elements, so as to compose a complete system.

## **Section 6.2. System integration**

This section will detail how the data mining model and data gatherer subsystems combine with the user interface to deliver a working product that meets both parts of the project aims. Firstly, an overview of what happens when a user clicks the “classify website” button is presented, showing the roles of each subsystem in generating the final prediction. Then, the use of concurrency is covered, to describe how progress through the progress bar is shown while a prediction is being generated.

When the system is first started, the data mining model is loaded from a file using WEKA’s `SerializationHelper` class. This allows the user to visualize the model at any time, even before making a prediction. Upon the ‘classify website’ button being clicked, the data gatherer uses the provided URL to collect the three computed attributes, a new data Instance is created from the collected computing attributes and the provided human attributes – specifying ‘?’ if a particular attribute has not been set - the training dataset is loaded from a file, the new Instance is associated with this dataset to tell the model it should treat the new Instance as part of that dataset, the model then classifies the new Instance, and finally the result is displayed for the user.

This, then, is quite a lengthy, in depth process, and takes some time to complete. Progress updates are sent to the progress bar upon initializing the classification, obtaining a value for each of the computed attributes, loading the training dataset from a file, setting up the new instance using collected and provided data and classifying the instance. However, the user interface runs on a separate thread to the event handler that contains the button action code, so a special concurrency mechanism must be used to send messages back and forth.

This mechanism is implemented within the JavaFX Task class, which provides multithreaded capability. Code executed within the Task’s `call()` method will execute in a different thread to the default `EventHandler` thread, and Task contains functionality to bind properties of UI elements to properties of the Task, so to update the progress bar, Task’s `updateProgress()` method can be called, and the progress properties of both the Task and the progress bar will update. Similar techniques were used to update the progress label that tells the user what the system is actually doing at that moment, and to pass the final result back to the user interface to be displayed on screen.

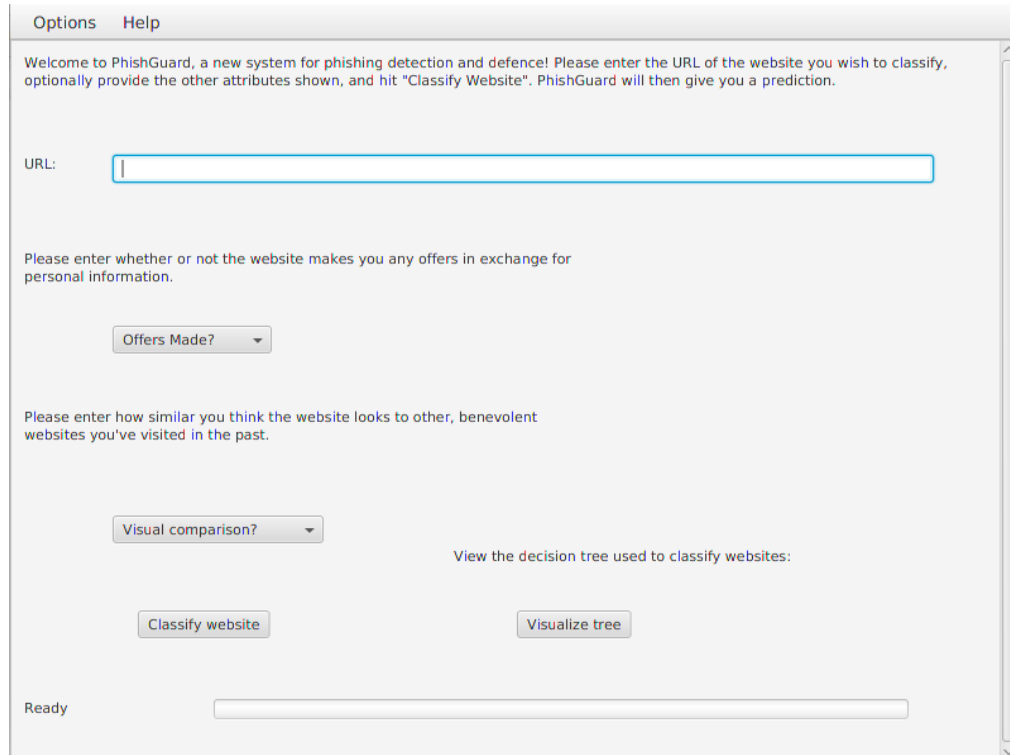
Exceptions were dealt with in a similar way. Whenever an exception arises in the `call()` method, the Task has an exception property that updates, halting execution of the classification task and displaying an error message on screen.

## **Section 6.3. User interface testing**

In order for the system to be fully ready for deployment to users, it must be tested. The full system can be tested in ways that the web crawler and data mining model cannot as separate pieces, and

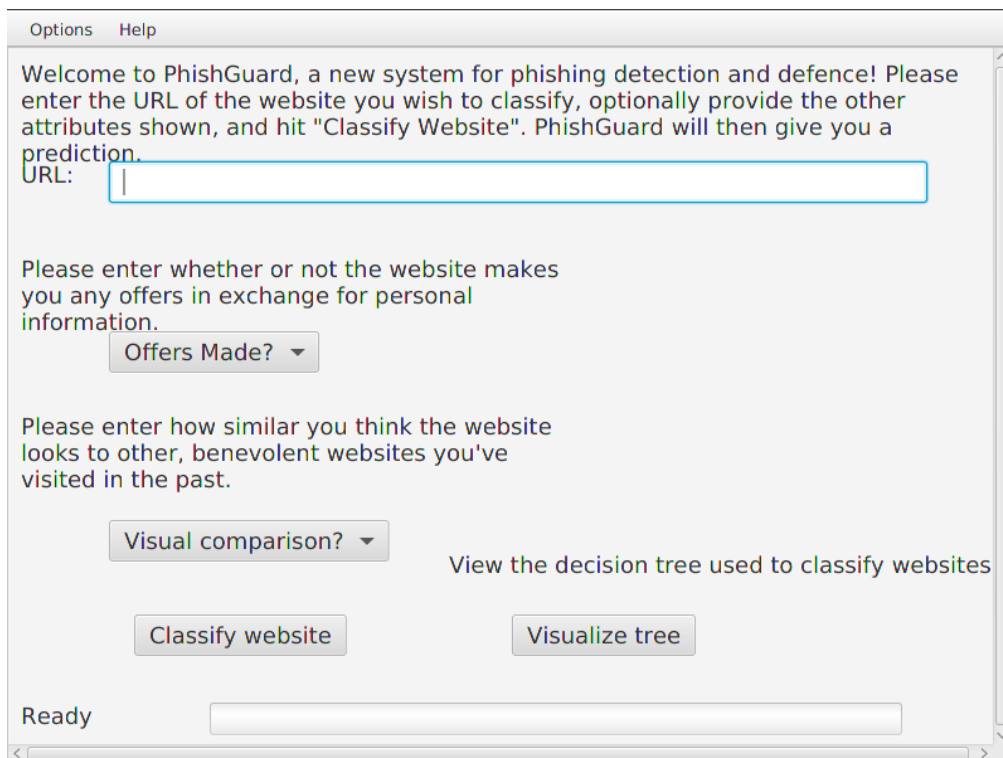
therefore this section deals with not only testing the user interface, including the accessibility measures, but also the system as a whole.

The user interface testing is not something that can be automated, so the following tests were performed manually. Firstly, the accessibility features were tested. Figures 6.3a through 6.3d show the results of these tests, as these features cannot be proven to be working and correct without visual confirmation.

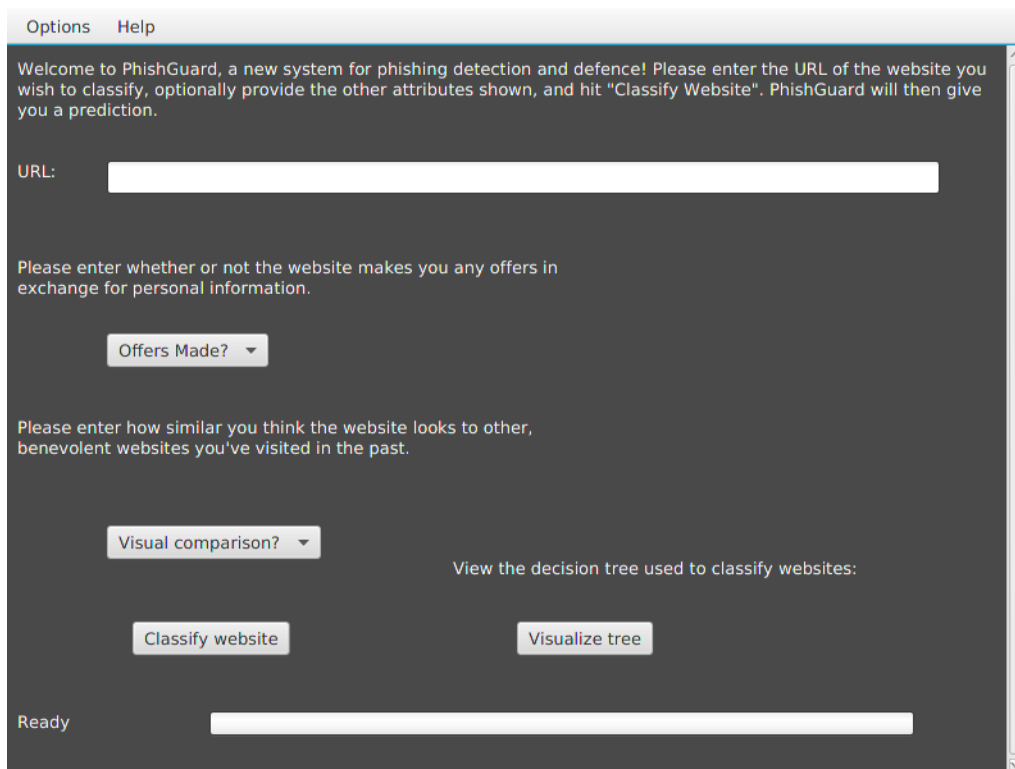


The screenshot shows a web application window titled "Options Help". The main content area contains a welcome message: "Welcome to PhishGuard, a new system for phishing detection and defence! Please enter the URL of the website you wish to classify, optionally provide the other attributes shown, and hit 'Classify Website'. PhishGuard will then give you a prediction." Below this is a text input field labeled "URL:". Further down, there is a label "Please enter whether or not the website makes you any offers in exchange for personal information." followed by a dropdown menu labeled "Offers Made?". Below that is another label "Please enter how similar you think the website looks to other, benevolent websites you've visited in the past." followed by a dropdown menu labeled "Visual comparison?". To the right of this dropdown is a link "View the decision tree used to classify websites:". At the bottom left are two buttons: "Classify website" and "Visualize tree". At the bottom right is a progress bar labeled "Ready".

*Figure 6.3a. The smallest font, at 11pt.*



*Figure 6.3b. The largest font, at 18pt.*



*Figure 6.3c. The dark theme, with the default font size of 13pt.*

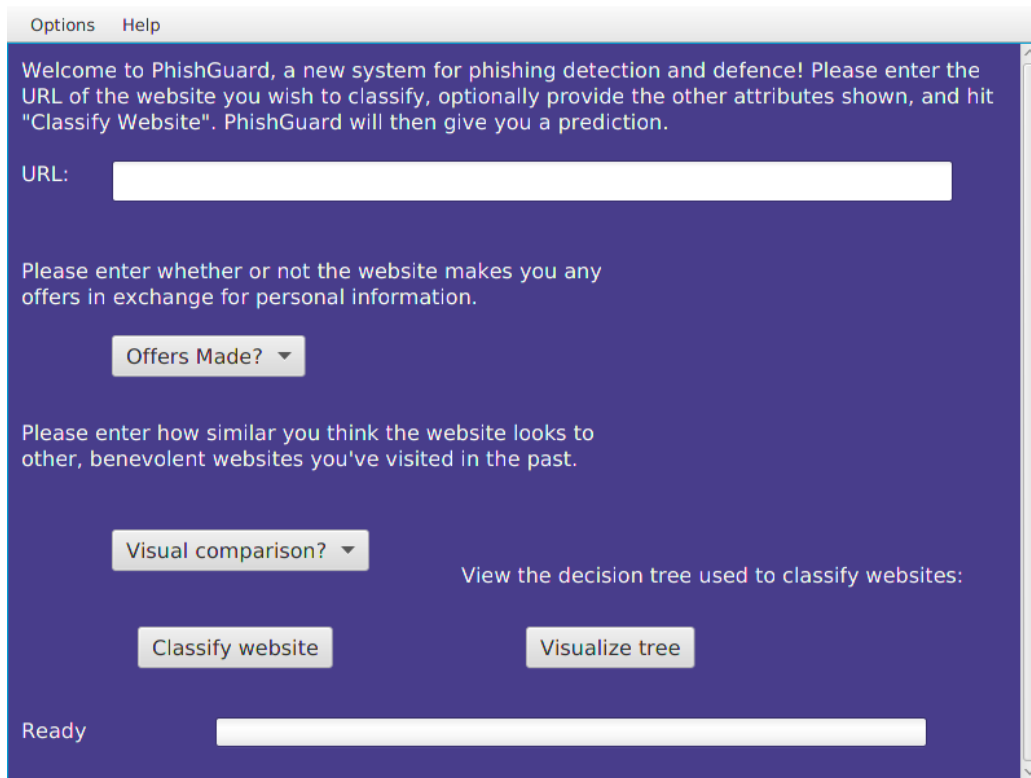


Figure 6.3d. The blue theme, with 16pt font.

Next, the help screen is tested. This appears in a new window.

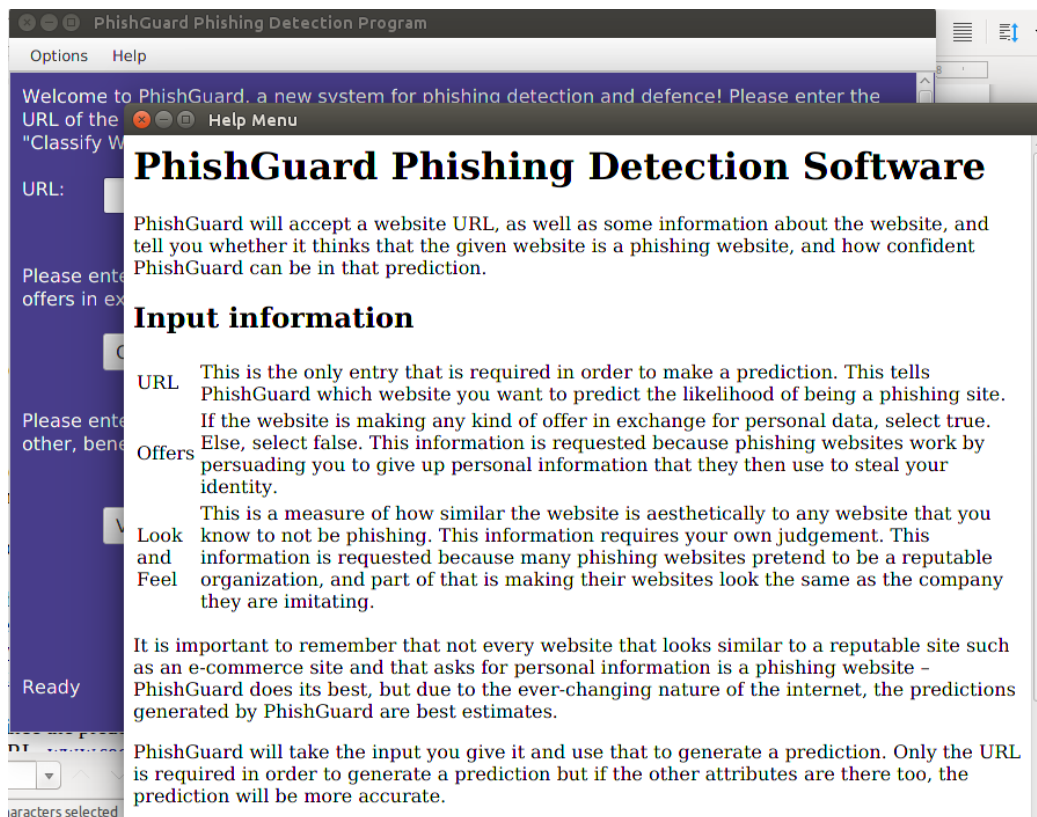


Figure 6.3e. The help screen, detailing how to use the system

Now that the user interface has been tested, the next step is to test the full system. As with the web crawling subsystem, this will involve positive and negative tests. The positive test is covered first; this is to ensure that all the variables necessary to obtain a prediction are in place, and to provide a baseline for modifying to derive negative tests from.

The positive test involves a valid URL being entered into the URL bar, and optionally values being set for the human attributes from the choice boxes. This generates both a benevolent prediction for [www.firstyearmatters.com](http://www.firstyearmatters.com), and a phishing prediction for <http://1granddailyprofit.com/5daymillionaire/> which was also in the training set.

Since the prediction only really depends on the URL, negative testing was carried out on an invalid URL, [www.secondyearmatters.com](http://www.secondyearmatters.com), a random string that is not in the URL format, and a blank input. All of these failed, with appropriate error messages. If the URL is invalid or does not match the URL format, the error message is “This URL is not valid” whereas if the input string is blank then the error message is “A URL is required”.



## **Chapter 7. Critical evaluation and conclusion**

Throughout this dissertation, a complete discussion of the Final Year Project has been presented. This discussion has gone through the entire development process, from requirements gathering, software design, data model training and testing, software implementation and testing. This puts me in a position to be able to look back on the project and, with hindsight, critically evaluate how it has met its goals, before concluding with a review of how the project fits into the state of the art. Furthermore, the lessons learnt can be evaluated, and the strengths and weaknesses of the system itself.

### **Section 7.1. Critical evaluation of the project**

This section will cover the evaluation of the project itself. Starting with a comparison of the project's achievements against the objectives, and then reviewing the project plan and explaining any deviances from it. Then, a system evaluation is performed, considering its strengths and weaknesses. Finally, an overview of the lessons that were learnt during the course of the project is discussed.

#### **Subsection 7.1.1. Review of the project's achievements against its objectives**

The project objectives were laid out in Section 1.3, but for reference, they are repeated here.

- To write a web crawler capable of recording the URL similarity, number of spelling errors and whether a redirection was attempted.
- To write unit tests to measure the performance of the crawler on test websites
- To write a user interface consisting of a form that allows a user to submit a URL, similarity to an existing website, and whether or not the website in question offers something in exchange for personal information.
- To link the user interface to a back end, consisting of the crawler and an algorithm implemented within WEKA, such that the crawler will gather the remaining attributes and the algorithm will run on the resulting data.
- To configure the crawler to gather an approximate 1:5 ratio of phishing and non-phishing sites.
- To provide the human attributes in at least 90% of the training data.
- To evaluate algorithms selected for this type of dataset, and select one to generate a model.
- To estimate the predictive performance of this model.

All of these objectives have been achieved, creating a system capable of predicting whether a given website is a phishing website, thus meeting the first part of the project aim. Secondly, the system records a predictive accuracy of almost 90%, thus meeting the second part of the project aim.

All of these objectives correspond to at least one of the three subsystems. By designing, implementing and testing these subsystems and verifying they are fit for purpose, the corresponding objectives are achieved.

### Subsection 7.1.2. Review of project plan

The plan, as initially submitted, was largely the same as it is now, at the end of the project. There is one difference, however. Initially, the third computed attribute was not the number of spelling errors, but rather the number of popups. This was changed because the internet operates on a client-server model, with processing being shared between the server that hosts the web page and your browser, that loads it. Web crawlers such as crawler4j operate on the server side, whereas popups are loaded in client side javascript. This makes something like a popup impossible to detect on the server side, and building a crawler that integrates client-side javascript into it is beyond the scope of this project.

Next, the time plan needs to be evaluated. Both the initial time plan as submitted in the project plan, and the updated time plan as of 10/3/17 are shown in Figures 7.1.2a and 7.1.2b.

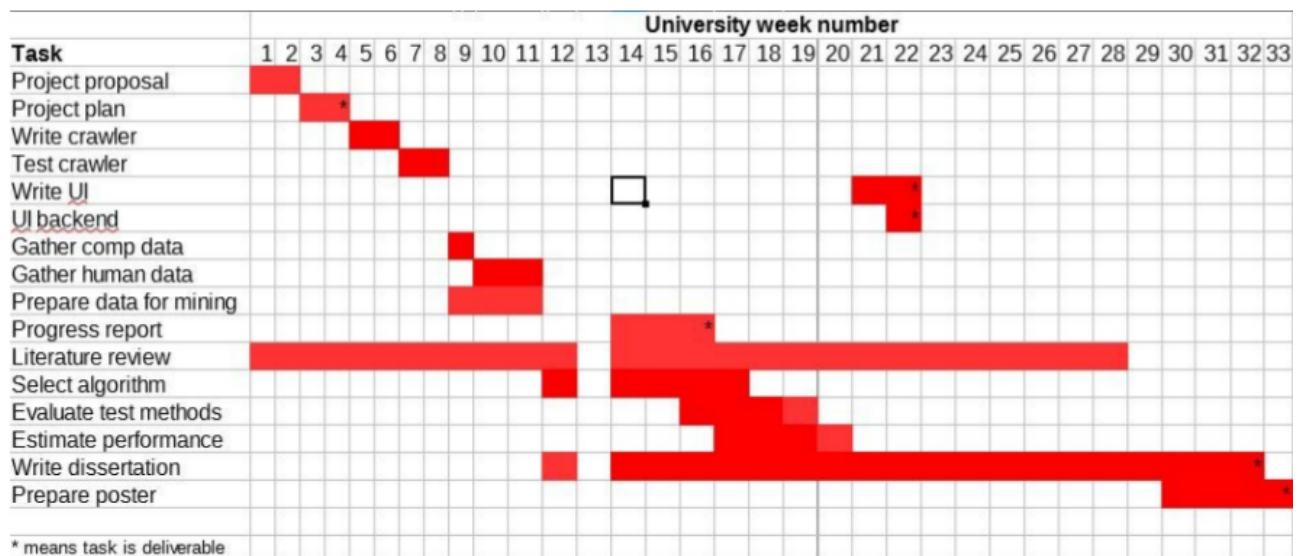


Figure 7.1.2a. The original time plan, submitted in the project plan

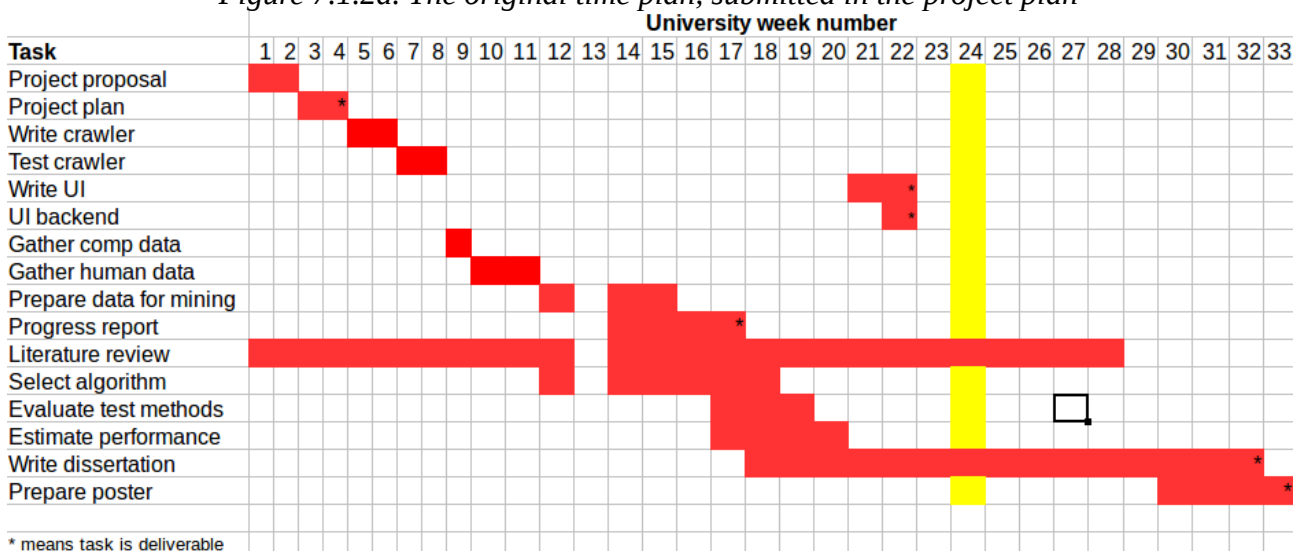


Figure 7.1.2b. The updated time plan as of University week 24

The time plan has largely stayed the same across the project. Most of the changes are simply moving a particular task back or forward a small amount, which is well within the scope for good time management. The largest change has been to shrink the dissertation writing task, because this had not been started by week 16, but the shrunken timespan for this task has still been sufficient to spend enough time on this dissertation to get it to the best standard possible.

Another major difference has been that data preparation has been moved after data gathering, rather than concurrent with it. This is because during data gathering, I realized that both the human and computed attributes need to be in place before preparation can be done.

### Subsection 7.1.3. Evaluation of system strengths and weaknesses

Nothing that exists is perfect, and this system is much the same. It has things it does well, and things that it does not do as well. This section suggests improvements to the technical side of the project, based on the current system's shortcomings.

One of the strengths of the system is its high predictive accuracy. This means that when a user nominates a website and receives a prediction, there is a very high chance of the prediction being correct. Therefore, the user can trust the prediction, which makes their experience using the software simpler, and therefore would contribute to its popularity were it to be sold commercially.

Another of its strengths is the simple user interface. As things stand, the user only has to enter a URL in order to achieve a prediction. The prediction can be more accurate if the human attributes are input as well, but this is not required in order to obtain a prediction. This means that non-technical users can use the system and receive predictions, which would, as above, increase the system's target market in a hypothetical commercial context.

However, the system is far from perfect, and one of its major glaring flaws is the small number of attributes. The list of attributes was compiled very early on in the project, before a substantial literature review was undertaken, and so could have been much improved by integrating some of the attributes and methods used within the literature.

Another flaw, albeit a smaller one, is scalability. The data gatherer's methods are not atomic – the values are not collected and used in a single, thread-safe operation. If this system is used by multiple users as-is, say on a client-server configuration, values collected by one user can be overwritten by values collected by another user. This is fine for the current configuration, whereby each instance of the software is separate and independent, but if the system is ever offered as a service, this will need to be addressed.

### Subsection 7.1.4. Lessons learnt during the course of the project

Leading on from the previous section, particularly the part about the project's shortcomings, the lessons that I learnt during the course of the project will now be discussed.

Firstly, I needed to learn about concurrency. This was required in order to get the user interface's progress bar to work correctly, and a lack of concurrency in the data gatherer/web crawler subsystem was the cause of the lack of scalability mentioned above. Fortunately, JavaFX uses a wrapper around the standard Java concurrency framework, which makes concurrency within JavaFX much easier to use than concurrency within other kinds of Java applications.

Secondly, I used technologies introduced to me throughout my degree course in new ways, namely WEKA. During the AIDM module, we used WEKA's explorer interface to mine small datasets. During this project I took this knowledge several steps further through visualizing data models, using the experimenter interface to conduct paired t-tests and hence prove statistical significance or insignificance, and most importantly using WEKA's Java API to use its functionality within my own application, loading a pre-trained data mining model from a file and classifying a new instance using it.

Lastly, I used all-new technologies, ones that exist outside the university framework. I learned about Maven during my industrial placement, and it proved invaluable for managing dependencies and easily adding third party libraries to my codebase. Through the use of Maven, I was able to use other APIs and programming libraries, such as crawler4j, which formed the foundation of the web crawler, and LanguageTool, which was used to gather the 'number of spelling errors' computed attribute.

## **Section 7.2. Project conclusion**

During the previous section, the discussion and evaluation focused on the project itself. In this section, the wider context will be considered, including how the project compares to the current research and what improvements can be made in order to make this system into publishable research.

### **Subsection 7.2.1. Comparison of results with existing literature**

This subsection will by necessity be shorter than the literature review presented in Section 2.1, but there are still things to say.

Xiang (2011) developed a multiple-stage framework to detect phishing websites. They considered their results in terms of true/false positives and negatives, as opposed to predictive accuracy, implying the cost of a false prediction was much higher than it is for this project. Nonetheless, this project still achieved a higher rate of correctly classified instances – 89.4% compared to 89% for their CANTINA+ system.

Furthermore, Abu-Nimeh et al (2007) used a wide variety of classifiers in an attempt to find ways to detect phishing websites. One of the methods they used was a classification tree, therefore comparable to the final choice of J48 for this project. Again, they considered false positives and negatives instead of predictive accuracy, and obtained 17% false negatives for classification trees. Therefore, this project surpasses this research in terms of accuracy, despite using significantly fewer attributes – 5 to Abu-Nimeh et al's 43.

### **Subsection 7.2.2. Potential improvements to the project**

There is always more to be done, and therefore this section deals with potential improvements to the project in light of existing literature.

Akanbi et al (2014) served as a guide for this project, and most of the algorithms used in the later stage of the data mining experiments came in part from this work. As such, it came as rather a disappointment when the results of this work failed to match it, but not a surprise – that study resulted in a predictive accuracy of over 99%!

Furthermore, the lack of attributes is likely to render this project unpublishable as too simple. As such, if I were to redo the project, I would have done a literature review before starting to compile the attribute list, and in this way follow in the footsteps of other researchers.

If, however, the system was to enter a commercial environment, then it is the user interface that will need to be prioritized for improvements. More features would be necessary, including the ability to not only visualize the model but also have it highlight the path down the tree taken to achieve a prediction, and the addition of concurrency to make it usable in a multi-user environment.

### Subsection 7.2.3. Review of other issues

The vast majority of software projects involve some form of ethical, legal, social or professional issue that needs to be addressed.

In terms of this project, there are potentially professional issues caused by having a single human expert manually classify all of the training websites, as well as providing the human attributes. The human attributes are subjective in nature, and therefore it is likely to be seen as poor practice for a single person's opinion to have such an impact on the predictions the system makes. For a system on the scale of a final year BSc project, this is okay, since only one person should be building the software anyway. But for a commercial environment, having so much determined by one person could cause issues, especially if other users disagree with my opinions on the values for the human attributes.

Any other issues are largely mitigated by the small scope of the project, but as before would become apparent once a larger scale is applied. Such issues involve false positives giving a benevolent website an undeserved reputation as phishing, or false negatives leading a user to believe a phishing website was benevolent.

## **References**

Abu-Nimeh, S., Nappa, D., Wang, X. and Nair, S. (2007). A Comparison of Machine Learning Techniques for Phishing Detection. In: *APWG eCrime Researchers Summit*. Dallas: Southern Methodist University.

Aha, D., Kibler, D. and Albert, M. (1991). Instance-based learning algorithms. *Machine Learning*, 6(1), pp.37-66.

Akanbi, O., Amiri, I. and Fazeldehkordi, E. (2014). A machine-learning approach to phishing detection and defense. Amsterdam: Elsevier.

Basnet, Ram B. and Tenzin Doleck. "Towards Developing A Tool To Detect Phishing Urls: A Machine Learning Approach". 2015 IEEE International Conference on Computational Intelligence & Communication Technology (2015): n. pag. Web. 11 Jan. 2017.

Bayes, T. 1763. An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions of the Royal Society*, 53:, 370-418.

Bergholz, A., De Beer, J., Glahn, S., Moens, M., Paaß, G. and Strobel, S. (2010). New filtering approaches for phishing email. *Journal of Computer Security*, 18(1), pp.7-35.

- Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), pp.5-32.
- Cs.waikato.ac.nz. (2008). Attribute-Relation File Format (ARFF). [online] Available at: <http://www.cs.waikato.ac.nz/ml/weka/arff.html> [Accessed 14 Jan. 2017].
- Damerau, F. (1964). A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3), pp.171-176.
- FBI, (2016). *FBI Warns of Dramatic Increase in Business E-Mail Scams*. [online] Available at: <https://www.fbi.gov/contact-us/field-offices/phoenix/news/press-releases/fbi-warns-of-dramatic-increase-in-business-e-mail-scams> [Accessed 6 Feb. 2017].
- GitHub. (2015). yasserg/crawler4j. [online] Available at: <https://github.com/yasserg/crawler4j> [Accessed 15 Jan. 2017].
- Gwtproject.org. (n.d.). GWT Project. [online] Available at: <http://www.gwtproject.org/> [Accessed 6 Jan. 2017].
- Hadi, W., Aburub, F. and Alhawari, S. (2016). A new fast associative classification algorithm for detecting phishing websites. *Applied Soft Computing*, 48, pp.729-734.
- Hajgude, J. and Ragha, L. (2013). Performance Evaluation of Phish Mail Guard: Phishing Mail Detection Technique by using Textual and URL analysis. *International Journal on Recent Trends in Engineering and Technology*, 8(1).
- He, M., Horng, S., Fan, P., Khan, M., Run, R., Lai, J., Chen, R. and Sutanto, A. (2011). An efficient phishing webpage detector. *Expert Systems with Applications*, 38(10), pp.12018-12027.
- Hedley, J. (2017). *jsoup Java HTML Parser, with best of DOM, CSS, and jquery*. [online] Jsoup.org. Available at: <https://jsoup.org/> [Accessed 29 Mar. 2017].
- Holte, R. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11, pp.63-91.
- Kim, H. and Huh, J. (2011). Detecting DNS-poisoning-based phishing attacks from their network performance characteristics. *Electronics Letters*, 47(11), p.656.
- Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In: *Proceedings of 14th international joint conference on artificial intelligence*. San Francisco: Morgan Kaufmann, pp.1137-1143.
- Lakshmi, V. and Vijaya, M. (2012). Efficient prediction of phishing websites using supervised learning algorithms. *Procedia Engineering*, 30, pp.798-805.
- Levenshtein, V. (1966). "Binary codes capable of correcting deletions, insertions, and reversals". *Soviet Physics Doklady*.
- Porter, B., Zyl, J. and Lamy, O. (2017). *Maven – Welcome to Apache Maven*. [online] Maven.apache.org. Available at: <https://maven.apache.org/> [Accessed 10 Mar. 2017].

Ma, L., Ofoghi, B., Watters, P. and Brown, S. (2009). Detecting Phishing Emails Using Hybrid Features. *2009 Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing*.

MacNeill, C. and Bodewig, S. (2017). *Apache Ant - Welcome*. [online] Ant.apache.org. Available at: <http://ant.apache.org/> [Accessed 10 Mar. 2017].

Microsoft Research, (1998). *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines*. Technical Report MSR-TR-98-14. [online] Microsoft Research. Available at: <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-98-14.pdf> [Accessed 8 Mar. 2017].

Murray, J. and Burchfield, R. (2000). *The Oxford English dictionary*. 3rd ed. Oxford: Clarendon Press.

Nevonprojects.com. (2015a). Detecting Phishing Websites Using Machine Learning | NevonProjects. [online] Available at: <http://nevonprojects.com/detecting-phishing-websites-using-machine-learning/> [Accessed 5 Jan. 2017].

Nevonprojects.com. (2015b). Heart Disease Prediction Project | NevonProjects. [online] Available at: <http://nevonprojects.com/heart-disease-prediction-project/> [Accessed 5 Jan. 2017].

Nutch.apache.org. (2017). *Apache Nutch™* -. [online] Available at: <http://nutch.apache.org/> [Accessed 10 Mar. 2017].

Odell, M. (1956). The profit in records management. *Systems*, (20).

Phishing Activity Trends Report (2016). Phishing Activity Trends Report. Anti Phishing Working Group. [Online] Available at: <http://www.antiphishing.org> (Accessed 6<sup>th</sup> Feb 2017).

Php.net. (2017). *PHP: Hypertext Preprocessor*. [online] Available at: <http://php.net/> [Accessed 10 Mar. 2017].

Quinlan, J. (1986). Induction of decision trees. *Machine Learning*, 1(1), pp.81-106.

Quinlan, J. (1992). *C4.5 - programs for machine learning*. 1st ed. San Mateo, Calif: Kaufmann.

R-project.org. (2016). R: What is R?. [online] Available at: <https://www.r-project.org/about.html> [Accessed 7 Jan. 2017].

scrumalliance.org. (2016). Learn About Scrum. [online] Available at: <https://www.scrumalliance.org/why-scrum> [Accessed 16 Jan. 2017].

Shiny.rstudio.com. (2017). *Shiny*. [online] Available at: <https://shiny.rstudio.com/> [Accessed 10 Mar. 2017].

Smart Vision - Europe. (2016). What is the CRISP-DM methodology?. [online] Available at: <http://www.sv-europe.com/crisp-dm-methodology/> [Accessed 19 Oct. 2016].

Virtualbox.org. (2017). *Oracle VM VirtualBox*. [online] Available at: <https://www.virtualbox.org/> [Accessed 16 Mar. 2017].

Vmware.com. (1999). VMware Virtualization for Desktop & Server, Application, Public & Hybrid Clouds. [online] Available at: <http://www.vmware.com/> [Accessed 15 Jan. 2017].

Webarchive.jira.com. (2017). *Heritrix - Heritrix - IA Webteam Confluence*. [online] Available at: <https://webarchive.jira.com/wiki/display/Heritrix> [Accessed 10 Mar. 2017].

Wiki.languagetool.org. (2017). *Java API - LanguageTool Wiki*. [online] Available at: <http://wiki.languagetool.org/java-api> [Accessed 29 Mar. 2017].

wikipedia.org. (2016). *Phishing*. [online] Available at: <https://en.wikipedia.org/wiki/Phishing> [Accessed 19 Oct. 2016].

Xiang, G., Hong, J., Rose, C. and Cranor, L. (2011). CANTINA+. ACM Transactions on Information and System Security, 14(2), pp.1-28.

Wenyin, L., Huang, G., Xiaoyue, L., Min, Z. and Deng, X. (2005). Detection of Phishing Webpages based on Visual Similarity. In: *14th international conference on World Wide Web*. Hong Kong: ACM.

Witten, I. and Frank, E. (2005). *Data mining: Practical Machine Learning Tools and Techniques*. Amsterdam: Morgan Kaufman.

Zhang, H., Liu, G., Chow, T. and Liu, W. (2011). Textual and Visual Content-Based Anti-Phishing: A Bayesian Approach. *IEEE Transactions on Neural Networks*, 22(10), pp.1532-1546.



## **Appendices**

### **Appendix A. The project log book**

## **Final Year Project Log Book**

**Sophie Brown**

**Wednesday 21<sup>st</sup> September 2016:**

I knew I wanted to do a data mining project of some sort, and so I approached Dr Bryant to be my supervisor with some thoughts on what I wanted the project to look like. He agreed to be my supervisor, and tasked me with writing a project proposal, reminding me to think more deeply and write more clearly.

I will definitely keep that in mind for most of the project, however my intentions with this log book are to record any thoughts I have before I refine them, as well as to lay out my reasons for making the choices I have.

I knew that thinking of specifics for this project would be difficult from the start – the Data Mining module was fairly theoretical in nature, going in depth on how various algorithms worked. There was a short case study, but how well that translates to being able to propose and scope a year-long project is unknown to me at this time.

The reasons for this are that the actual data mining during the case study seemed very simple from the point of view of me, the data miner. The whole process of running several algorithms on a dataset to generate a pattern, which is the first half of data mining, and analysing the predictive accuracy, which appeared to me as a simplified version of the second half of data mining, took less than 2 hours.

Clearly then, additional scope is required in order to propose a challenging project. I considered adding a user interface, so that users can generate their own data. This new row of data is then aggregated with the others, a pattern induced from the dataset, and that pattern tested. The up side of using this method is that it would be simple to implement – I already know how to write user interfaces in Java, WEKA (the data mining tool we used in AIDM) is written in java, and offers an API so I can effectively make WEKA the back end of my application. The down side of using this method is that it effectively constrains the size of the datasets I can analyze – no user is going to want to enter 50 different pieces of information about a topic.

In addition, I need a concrete subject for my project – a dataset I want to mine, questions I want to answer. Dr Bryant recommended a project that does not involve other people at any stage, as such a project would be more difficult to gain ethical approval for and to work with in general.

My first idea was for a medical diagnosis system – a user would enter their symptoms, and the system would give them what it has diagnosed them with, as well as the likelihood that this symptom is correct according to its own knowledge (in other words, the predictive accuracy). This option was very quickly discarded because of the above ethical implications.

Next, I considered a system that would analyse apache log files for a certain website, and produce some potential optimizations that can be made in terms of usability and efficiency. As an example, if a user starts at the home page, spends 30 seconds there, clicks to a second page and spends only one second there before navigating to a third page and spending 3 minutes there, then it is likely that the user merely navigated to the second page as a stepping stone to the third, and this can therefore be suggested as something to improve.

The problem with this setup is that of getting permission from a webmaster to mine their log files. I thought of using firstyearmatters as the website to use, as Dr Young is likely to agree to providing log file data for a project like this – and in fact already has provided historical data as part of the first year CSIL module – but the problem with that is that FYM is already well designed in terms of usability, and the results of this project are likely to look like “remove any and all hierarchies and link to everything on the home page” which is very bad usability. This problem is likely to persist for most websites, even if I could get permission to rummage through old log files. Therefore, this idea was canned.

My third idea was analysing house prices. Users would provide features of their house, such as number of bedrooms, proximity to commerce centres, classification of building, size of the lot, county etc and the system would use this data to provide a prospective house price. Inspiration for this project was taken from <https://www.kaggle.com/competitions>, so using this as a basis for my project is likely to be a plagiarism issue. In addition, the kaggle project uses 79 different attributes, many of which an aspiring homeowner will not know. Cutting down the number of attributes was considered, but would not counter the potential for plagiarism.

My fourth idea was for a system that would accept website information such as the URL, domain name, whether any popups appeared, and the density of ads, and build up a picture of whether the website is likely to be a scam site or not. Out of all my ideas so far, this one is most likely to be the one that gets taken forward. The scope problem is nullified somewhat by the use of the previously-mentioned user interface to WEKA, and I can build a tiny web crawler to gather URLs, then visit a sample of the sites at random to check out the popup and ad frequency parts. This will mean I will have missing data, which will be something else to consider.

As far as project ideas go, it's not overly inspired, but it does not have any of the flaws that struck out the rest, so it is the most worthy of presenting to Dr Bryant for further feedback and refinement.

## **Monday 3<sup>rd</sup> October**

Dr Bryant has given me feedback on my project proposal, and I now need to revise my draft in accordance with this feedback.

First things first, I needed to clearly define what was meant by “scam site”. Initially I had restricted my proposal to just dealing with phishing sites – sites that masquerade as legitimate, trusted sites to fool you into providing information or money – but changed it to the more general scam sites after some grappling with my understanding of the scope of a DM project. Since my only previous experience with data mining in practice was a short case study, I kept things broad and vague in the initial draft, so that the scope could be refined in the next iteration of the proposal.

Hence, I changed my proposal back, to refer to phishing sites instead of scam sites. Because I was now dealing with something more specific, I was able to add another attribute to my list; whether or not the website providing the link to the website I am gathering data from was truthful about where

that link leads. For example, if [www.website.com](http://www.website.com) claims to contain a link to PayPal, but actually links to [www.thisIsAScam.li](http://www.thisIsAScam.li). This could be a simple boolean to record whether the link is truthful or not, or could be a numerical measure of how different the promised link is to the real thing. (Note: due to the prevalence of URL shorteners such as bit.ly and ad.fly, this is unlikely to be able to be automated, and therefore missing data is likely here too) Measures can be taken through something like the Levenshtein distance, or similar.

Furthermore, I needed to add more attributes. I went with popups – since the browser can stop popups, it stands to reason that it can record how many a given site attempted to bring up, and that more popups increase the likelihood that a given site is fraudulent.

Furthermore, certain popups are persistent, and will repeatedly appear. This too is an indicator of a phishing site, albeit not as common as mere prevalence of them. In addition, having two attributes that depend on one element of a website – in this case, popups – increases the influence this element has over the final classification. Hence, I added this is a potential attribute, awaiting Dr Bryant's feedback.

## **Sunday 9<sup>th</sup> October**

After a meeting with Dr Bryant, I started work on my third, and hopefully final, draft of my proposal. Since the deadline for the project plan is rapidly approaching, I need to start thinking about that, more than this proposal.

One of the things I was asked to do during my meeting was to hand-sketch a diagram of my project structure – where the training data is acquired from in terms of automated data gathering or human data entry, the algorithm used to turn this data into a model, the test set, and then, the differences between the real cases and training set, again in terms of what was gathered automatically or through a human.

Furthermore, my second iteration of the proposal was much more oriented around simply tossing ideas around, and these now need refinement, to start the process of turning what I have now into SMART objectives for the project plan.

This means compressing my proposal into something a bit more coherent, so I can more easily draw a diagram that accounts for every attribute and how it is collected.

So I removed the “popup potency” attribute, reasoning that repeated popups, in my own experience anyway, are more affiliated with virus sites, not phishing sites.

Also, the attribute dealing with the linking site lying about where its link led to was replaced with a more simple “if the website attempts a redirection, its likely to be a phishing site” attribute. The reason for this was that after two iterations of the proposal, I still couldn't find a way to phrase what I was looking for, which I knew would be trouble come the dissertation. While trying to explain this attribute to Dr Bryant, he misinterpreted it as the redirection, and the more I thought about it, the more that made sense. It'd be easier to implement, gather data for, and describe when it comes to write my dissertation.

## Thursday 13<sup>th</sup> October

My seminar with Dr Bryant today reminded me that I need to focus on breaking my project down into SMART objectives, and break these objectives down further into tasks for a time plan.

One of the things I noticed while drawing my diagram is the existence of many ‘mini-programs’ in different languages in order to do certain things. For example, a web crawler to gather training data, a BASH script to process the resulting data into an ARFF file, a user interface as an optional objective, which will need to call the crawler, and tell it to gather data on a specific URL. All of this will be underpinned by WEKA. Since WEKA is written in Java, I need to write the user interface in Java to call WEKA’s API easily. I initially thought to write the crawler in PHP, since PHP is a server-side web based language, but there are web crawlers pre-written in Java, so I can use those.

Also, I may type the ARFF headers into a file manually, and simply have the crawler output to it, which would neatly remove the need for the BASH script.

Therefore, a SMART objective could be “to write the software necessary to gather and mine the data”, which could in turn be broken down into writing the crawler, the user interface and a back end interface to WEKA. This is specific because it deals with a particular part of my project, measurable in that the output of the programs can be measured, realistic because I need to do this in order to get my project done, and time driven in that parts of it need to be done in order to actually mine the data.

Another SMART objective could be to actually gather data and prepare it for mining. Preparation will be ensuring there’s instances of both classes in the dataset, making sure the dataset is of an appropriate size, making sure there are no, or very few, missing values. The measurable output of this objective will be a dataset of several hundred rows, with approx. 10% missing values (subject to change), with a certain number of

Missing values have been struck down from my project because the real cases that will be used during the deployment phase will likely not have missing values – the attributes that are automatable will not have any, as they will be automatically generated, and the attributes that are not will be optional in the user interface, but including them is likely to generate a more accurate prediction during the data mining.

Speaking of the actual data mining, another SMART objective could be to discover a DM algorithm that will provide a model with a high predictive accuracy on both the training and test sets. This will require a literature review to find the best algorithm for this specific use-case, and likely a great deal of trial and error – WEKA has hundreds of algorithms, and trying out all of them is likely to take a prohibitively long time – however, if I limit my search to algorithms that provide, say, 80%+ predictive accuracy, then I can stop when I find one that does that.

Summary of SMART objectives and subtasks:

1. To write the software necessary to support the data mining aspect of the project. Subtasks:
  - Write the web crawler
  - Test the web crawler by gathering some data with it
  - Write the User Interface
  - Provide a back end to the user interface using WEKA
2. To prepare a dataset for training

- To configure the crawler to gather a mix of phishing and non-phishing sites
- To provide the human attributes in at least 85% of the training data
- 3. To create a model that provides high predictive accuracy
  - To do a literature search for algorithms that work well on this type of dataset
  - To do a literature search for testing methods that work well on this type of dataset
  - To ensure a predictive accuracy of 80%+

My dissertation will not be in the SMART objectives list, but it WILL be in the time plan – Likely starting at the same time as my first lit review, because before then, all I will be able to write is the generic, boilerplate parts, such as the title page and abstract.

Speaking of the time plan, I need to consider which of these sub tasks will be in which order, and approximately how much time they'll take.

I can immediately start writing a web crawler, now that I've found a java library for it, so that's my first task. I anticipate it taking me some time to get to grips with the library, and to write the logic to parse the raw website data into attributes for my dataset, so I gave this task 2 weeks.

I cannot test the crawler until after I've written it, so that must be after the "write crawler" task. Testing exists to uncover faults, so I gave this task 2 weeks as well, to uncover and fix any bugs with my crawler.

Once my crawler is written up and fully tested, I can put it to work gathering data, and parsing it into a dataset. The data gathered by the crawler will be incomplete, as my system is a collaborative learner, so I will need to fill in the missing pieces by hand.

While these two tasks are ongoing, I will be preparing the dataset for mining, so as to ensure that both class values are represented as accurately as possible within the dataset. Combined, I estimate these three tasks to take about 3 weeks. Most of this is likely to be preparing the dataset, as neither of the other tasks will be particularly difficult, although populating the dataset with human-only attributes is likely to be time consuming.

After I have my data, I need to start my literature review. First in search of an algorithm to run on my data, then on an adequate testing method for the subsequent model. These can be largely concurrent, with about a week reserved for each specific task each way.

Early on in my literature review, I will start my dissertation in earnest – once I have something solid to start writing about. I do not want to start any earlier than this, because I run the risk of trying to do too many things early in the project and burning out.

It is highly likely I will have a working algorithm and test method early into these tasks, and the remainder of the time will be testing them, to ensure that the model generates at least 80% predictive accuracy on test data. But, once I have an optimal algorithm and test method, I will still need to spend time ensuring a high degree of accuracy – which may even require partial data replacement.

Once predictive accuracy is assured, I can start on the deployment phase, starting with a basic Java UI that will take a URL and the "human" attributes of the dataset, use the URL to find the "computerized" attributes, then aggregate them into a row of data that will not possess a class value.

Since the actual processing is done by the crawler, which will already exist, this will only take one week to do.

Once the UI is created and able to generate a real case, I can connect it to WEKA, use my model on the case, present the result to the user, then append the new row to the training data, so that the algorithm can be re-run and a better model generated and tested for use on the next real case.

As the deadline for the project fair draws nearer, I will also need to start my poster for the fair. I will start this 4 weeks before the deadline so as to give me plenty of time to create a good poster.

### **Saturday 15<sup>th</sup> October**

Now that I have a time plan and some SMART objectives, I can put my plan together. While doing so, I noticed that my objectives, while implicitly measurable, do not contain an explicit quality or quantity measure, as stated in the guide.

However, an objective of “write software to do a task” should be sufficiently measurable, given that my programming assignments have been marked throughout the two years.

The objective to “prepare a dataset” is much harder. For now, I’ve stated the characteristics my data must have before I consider it “prepared”.

The data mining objective is measurable through the predictive accuracy of the model.

The methodology section of the plan, however, I am utterly stumped by. I do not know any formal methodology processes for data mining – perhaps we were taught one in the Data Mining module in second year, but informally, and behind the scenes.

What exactly constitutes a methodology? Would I need a mixture of software engineering processes and data mining processes?

Time is running out for me to send my draft to Dr Bryant – I want to get it done today, receive feedback on or by Tuesday, incorporate the feedback into a second draft for Thursday, and incorporate what feedback I can from that meeting into the final submission on Friday.

Therefore, I may need to seek clarification on this issue, as well as how I can improve my plan, to be a project that will get me the grade I want.

### **Tuesday 18<sup>th</sup> October**

I had a meeting with Dr Bryant about how I could improve my project plan, and what methodology I should use for my project. As such, before the meeting I looked up data mining methodologies, and found CRISP-DM, the Cross Industry Standard Practice for Data Mining. It seemed to line up with what I had been taught, so I noted it down as a potential methodology to bring up during my discussion.

In addition, there is likely to be a separate methodology needed for the software engineering parts of my project. Since I have some idea of what to do, but not enough that a waterfall model would

work well, then an agile methodology would seem appropriate. It would, however, need to be scaled down, since I'm building a small software product in a small amount of time.

During the meeting, I received extensive feedback on all aspects of my project.

The title needed to be altered – my project is not simply a data mining project, so stating that it is at the beginning is likely to turn people away, who aren't interested in data mining. In addition to this, "categorize" was the wrong verb, since there are only two classes, and "distinguish" would be better in this case.

For the aims, there are effectively two separate aims mentioned. This is not necessarily a bad thing, since if one fails I can claim that the true aim was the second and vice versa. I also need to change the wording again.

The objectives are among the most important part of the plan, and so these got the most treatment.

First and foremost, all of my objectives need to be measurable. My first objective was only implicitly measurable, and needs to be quantified. Since this objective deals with writing the software for my project, I need to think in terms of software quality metrics, which means talking to Dr Bass, or doing my own research.

Objective two, the objective about preparing the dataset, is much more easily made measurable, in that I simply need to state what I want to measure. I'm mostly there now, I just need to make explicit what I want to do.

The third objective, the one about creating a model that provides a high predictive accuracy, needs a bit more tweaking. Every student does a literature review, so that's redundant. But the second half of those sentences, the parts that deal with finding an algorithm and test method, are necessary. Those lines should be re-written to reflect this.

Throughout all of this, I need to consider that the objectives are there to measure my progress towards my aims. Therefore, they all need to exist to further the process of achieving my aims.

Moving on to the resources section, not many people will know what WEKA actually is, nor that it's written in Java. I need to make that more explicit, as well as the fact that I'm using algorithms coded in WEKA, and calling those from my application.

Finally, the time plan. My biggest mistake with that was assuming that the time plan starts after the submission of the plan, in week 4. It really needed to start in week 1, and include all deliverables – the plan, the progress report, everything.

### **Wednesday 19<sup>th</sup> October**

Now that I've figured out what I need to do, and broken it down into small, logical tasks, I then needed to actually do the work of amending my plan.

Most of this work was fairly simple, however, I ran into problems when the time came to rewrite the first set of objectives in my project plan. I did not know of any software quality metrics, and would not have had time to research one that would be appropriate for the small-scale programs I

will be making. Therefore, I re-wrote these objectives in terms of the output of the programs, trusting that the outputs are measurable and therefore makes a suitable objective.

For the methodology segment, I was advised to think primarily in terms of a software engineering approach, and integrate parts of a data mining approach into it. The only data mining methodology approach I know is the CRISP-DM approach, so I integrated the “Data gathering”, “Modelling” and “evaluation” phases of that approach into a scaled-down version of scrum – scaled-down because I’m building small systems in small timeframes, sometimes as little as a week.

### **Thursday 27<sup>th</sup> October**

Had seminar with my supervisor on the literature survey. When I come to write my dissertation, I need to provide justifications for using the technologies I have, that make it clear that I’ve evaluated multiple technologies for this purpose and say why I chose the technology I did.

I need to not simply have websites in my literature review – I need to read papers and journal articles to gain maximum marks.

In addition to this, there are potentially millions of relevant articles, and depending on the size and language used in these sources, I will only have time to study in-depth about 5-15 sources. I can often find relevant journals from textbooks, for example the Witten et. al. Book repeatedly references the “Machine Learning” journal.

Furthermore, reference journals and electronic copies of books and papers as if I were referencing the paper copy. This links back to “not having only websites in the lit review” point.

Later on, I tried crawler4j as a web crawler – seems to work. Crawlers in general tend to crawl web PAGES as opposed to web SITES, hence my crawler will only want to crawl a domain once. To that end, I’ll maintain a list of domains in-memory, and check each new link to see if it matches up with the list. If it’s a new domain, add it to the list and crawl it. Otherwise, skip it.

Phishing websites turned out to be not as easy to find as I’d hoped, quick scan of a few papers revealed most phishing initially done by email containing links to phishing site. Will set up a deliberately insecure windows box inside a virtual machine and hope to attract some spam emails to set as seeds for my crawler.

In addition, I did some research into the issue of tracking popups, and its less than trivial. Because of the dynamic nature of javascript – a call to window.open() could actually look like anything – what I would need to do is run every site in a sandbox and track opening windows that way. Or do it manually. Or find another attribute. I’m leaning towards finding another attribute, one that can be automated.

### **Saturday 29<sup>th</sup> October**

I built a prototype web crawler, using crawler4j as the crawling API and Jsoup as a page parser. In addition, I started building up a list of trustworthy sites. For now, since my data mining logic isn’t implemented, I will simply assume that if there is less than 5 characters difference between the crawled page and a page on the list, then it’s a phishing site. Since I’m starting from academic sites like firstyearmatters, there are unlikely to be phishing sites near this part of the network. I then mark the levenshtein distance as an attribute, by having a class that will encapsulate these attributes.



I then need to know if the site redirects to somewhere else. Luckily, there's a set of return codes for this – 30X – and a page parsing library that allows me to get this return code; Jsoup. Using this combination, I can set a boolean value in my attribute data object to tell if this website has redirected me or has attempted to redirect me.

## **Monday 31<sup>st</sup> October**

Started thinking about replacements for the popups attribute. I considered using spelling and grammar checks as a replacement, as while there are likely to be false positives in the benevolent sites – company and product names that are not in a dictionary, for instance – there are likely to be far more in phishing websites that have been mass produced by people who may not speak english very well. Also, if repeated errors can be counted only once, this will cut down on the number of false positives without impacting the true positives very much.

I tried LanguageTool as a spell checker, but only conducted brief tests before needing to stop for the night. I will ask Dr Bryant about this problem at my meeting tomorrow, and take his feedback into account.

## **Tuesday 1<sup>st</sup> November**

I had a weekly supervisor meeting with Dr Bryant today, in which I informed him of my progress and the stumbling block I had run into with regards to the “counting popups” problem. He mentioned that his browser, Mozilla Firefox, contains a popup blocker, and that most in-browser popup blockers keep a count of how many popups they've managed to block, as a means of showing to the user how useful the blocker is. He recommended that I find an open-source blocker and attempt to interface my code with that of the blocker. The problem with that is that my web crawler is written in Java, whereas popup blockers are likely to be in Javascript. In addition, - and this is a point I only considered after the meeting – I am not using a browser to visit these sites. I'm using a web crawler. In effect, the crawler is replacing the browser's job of connecting to a web site and downloading data onto my local system.

I received the go-ahead to add the “spelling and grammar check” attribute, with the reasoning that if it would be easy to implement, then I may as well do it. That way, if I do not manage to get the popup count working, I have something to fall back on, and if I do, then I have two more attributes, and hopefully this will lead to a more accurate model when the time comes to generate that.

## **Monday 7<sup>th</sup> November**

Today I started writing unit tests for my crawler. I'm using the crawler4j library to provide web crawling functionality, which has a central visit() method to visit web pages, so I thought that would be a good first place to test.

Unfortunately, the structure of this method is not conducive to testing. It returns nothing that can be tested against, and at present, has no side effects. When the system is used to gather data, the side effects of the method will be that a line of text is appended to a text file. This does not tell me much about all the things the method does to formulate that line of text, so I started breaking the test class down into the various forms of processing used to get the final result, even if at present, nothing is being done with it.

There are three sources of data that my crawler is gathering – the URL similarity to a list of known good websites, the redirection status, and the number of spelling mistakes. I started with a test to get a set of spelling mistakes, then asserted that there is at least one element in this set, trusting that any website would have at least one word that isn't in the dictionary. Unfortunately, my test site – the firstyearmatters home page – did indeed contain zero spelling errors. This is possible, if unlikely, but I could hardly NOT test the resultant set, so I changed the test website to be the ebay home page. This did come up with some errors, so I am confident that my method is working as expected.

## **Tuesday 8<sup>th</sup> November**

Next, I tested the redirection testing code. For this, I needed a positive and a negative test – a website which redirected and one which did not. My existing ebay URL sufficed for the latter, and I used a URL shortening service to provide the latter – shortening firstyearmatters.info. However, the response code generated by the URL shortener came out to be 200 – an OK status code, not a redirection one!

I stepped through the code in the debugger and found out that Jsoup, the parsing library I'm using, follows redirect requests by default. Therefore, I put in the shortened URL, got a 301 redirection response code, that redirection was then followed, this time using the true URL, a 200 code was returned, and this code was the one in the response given to my test. The fix was to simply disallow following redirects – a policy that gave me the correct test results, but could lead to problems later on. I will keep an eye on this section of code, and take another look if needs be.

Next, I tested the levenshtein distance code. Just like the redirection test, this is a binary test. Something is either in the list of approved websites, in which case the levenshtein distance tested against every member of that list will come out to be 0, or it will not be on the list, in which case the levenshtein distance will be something greater than 0. We don't know what the distance will be, but either way, having both kinds of tests in my test suite is a guard against my code failing to perform when I need it to.

Finally, with all the intermediate processing tested for, I wrote a test for the main “visit()” method, implementing data writing functionality to a test ARFF file and testing that the file does indeed increase in size when a web page is visited. With the increased confidence that all the pieces worked, I was not surprised when this method passed, first time.

## **Saturday 12<sup>th</sup> November**

I didn't do any practical work today, but I did think about the possibility of implementing the “popups” attribute into my crawler, in addition to the “spelling mistakes” attribute. Arguments FOR working on it include the possibility that prevalence of popups is likely to be a more accurate indicator of whether a given site is a phishing site, as using spelling errors as a significant factor may lead to inherent bias in the results – if “high numbers of spelling errors” correlates strongly with “phishing site”, then my project is more likely to categorize websites that are not written in English as phishing.

Arguments AGAINST include something I realized after my meeting with Dr Bryant on the 1<sup>st</sup> November – that I will not be able to leverage any kind of browser plug-in such as a popup blocker to access information on what kind of popups are on the screen, because I am not using a browser to download and view the web page – in fact, at this stage I am not even viewing the web page at all! In effect, the crawler replaces the browser's function, so I would effectively need to write my own

popup blocker. That is likely to take lots of time – considering that there are entire companies who write popup blockers as their main, or a considerable portion of their business model, writing something that could serve my needs in this respect is likely to be a Final Year Project of its own.

For that reason, I will not be implementing a popup scanner into my web crawler. The Upside of this is that it gives me something to talk about, discuss and justify, when I come to write my progress report and dissertation.

## **Tuesday 15<sup>th</sup> November**

I had a supervisor meeting with Dr Bryant today, in which I received verbal feedback on the copy of my log book that I sent to him on Saturday 12<sup>th</sup> November. Feedback was largely positive, with a note that I was perhaps a little too hasty in dismissing the idea of incorporating a popup scanner into my crawler outright, and that I should at least look through the source code of open source blockers such as the one built into FireFox.

In addition, he suggested that I should try a more iterative approach to the data mining aspect of my project, in case there is something fundamentally wrong with the attributes I've selected. Which means I will need some examples of phishing websites, because mining data that contains just one class will not be a very good test of anything at all, really.

I considered the problem of where to get phishing sites with Dr Bryant, and he recommended an information security analogue to the UCI Machine Learning repository. In addition, my project is highly unlikely to be the first time anyone has ever needed to gather some phishing websites together for research purposes, and those datasets may be available in the references of the papers published.

## **Thursday 17<sup>th</sup> November**

Today I started to gather data to begin preliminary mining, in an attempt to understand whether the attributes I have chosen are viable or not for the task of differentiating phishing sites from benevolent ones. As such, I needed as broad a representation of attribute values as I could find.

While doing this, I noticed that URLs that should provide a value of true for the redirection attribute were instead providing a value of false. The reason for this was that my crawler was actually following the redirections given, and using the redirected URL as the value to gather data from, just like how Jsoup follows redirections when I was writing my unit tests. Unfortunately, my program works slightly different when crawling than when running unit tests. There is a visit() method that I have overridden to gather all the data, then every hyperlink on the page is scanned and tested according to a shouldVisit() method, then the links that pass this test are visited and the cycle repeats. Therefore, I needed to set a flag in the shouldVisit() method to test for redirection, and then pass that flag on to the visit() method, without it being overwritten by any of the subsequent calls to visit().

What I ended up doing was creating a HashMap of string and boolean, to hold a URL and its redirection status. I would then retrieve the status from this map, then remove the entry once I was done with it to avoid memory leaks.

This means that the unit tests I wrote to test that my redirection-detection code worked needed reworking, as the methods now take different arguments and do different things.

## **Thursday 24<sup>th</sup> November**

During my last meeting with Dr Bryant, in which I updated him on my progress in finding a corpus of spam emails, he recommended I write a BASH script to automatically extract the URLs from the text files, as a chance to show off my knowledge around the project area and give me something to discuss in the progress report and dissertation. Since Dr Young is my assessor, using BASH in particular would likely impress him.

So, after brushing up on my BASH scripting, I had a script that iterates over every file in the directory where the corpus is stored, applying the regular expression 'https?://.\*?(\"| )' to it which matches a list of URLs, then echo the list to a different text file.

This method is not perfect, as there is some garbage in my data which will need to be dealt with. I will be using this list of URLs as input for my preliminary data mining, so as to investigate whether there are any obvious emergencies or flaws in the attributes I have selected for computerised processing.

## **Sunday 27<sup>th</sup> November**

Today I started thinking about my literature review, downloading papers from SOLAR for an overview of the abstract, introductions, conclusions and reference list before deciding if the article is worth including. In addition, I began considering arguments for and against the various technologies and programming languages I use, with a view to constructing convincing justifications for my choices.

Also, I wrote some sed commands to strip out all of the junk data in my results file. Fortunately, all I needed to do was to separate each URL onto its own line and remove fragments of <br> and </a> tags that got left in.

## **Saturday 3<sup>rd</sup> December**

I took some time off from my project to work on assignments for modules with a more imminent deadline. Therefore, I fell a bit behind on my project.

To catch up, I need to start doing some proper data mining, which means finding instances of both phishing and non-phishing websites. The non-phishing was simple – I could point my crawler at firstyearmatters and let it go, since academic websites such as firstyearmatters and the websites it links to are unlikely to be linking to unsavoury websites. The phishing sites I already have, so I need to configure my crawler to be able to visit each URL in a file. But first, I wanted to fix the unit tests for the redirection detection.

Once I'd done that, I set about writing a method that would allow my crawler to crawl sites from a file. However, the redirection detection method I was using depended on there being a page linking to the one to crawl. In this case, there is not – everything has to be done within the visit() method.. What I really need is a built-in way, within crawler4j, to both acknowledge that a redirection has occurred, but follow it anyway.

After some consideration, I realized that wasn't necessary after all. What I needed to do was to treat every URL in the file as a "seed URL" - the start of a crawl. I could configure the crawler to only crawl seed sites, and suddenly I had a crawler configuration that would crawl only the sites in a file – no more, no less.

Hence, I crawled for a set of phishing websites and a set of non-phishing websites. In total, I had 115 phishing and 519 non-phishing. Then I ran a collection of algorithms on this data. 1R, 0R, K-NN, Naive Bayes, J48. The algorithms with the highest predictive accuracy were K-nearest neighbour and J48, with the exact same predictive accuracy. The following table shows each algorithm run and the predictive accuracy generated.

Algorithm	Predictive accuracy	Notes
1R	91.0095%	Selected SpellingErrors as attribute to test
Naive Bayes	84.858%	Predicted phishing 18% of the time
K-NN	95.2681%	K value of 1
J48	95.2681%	Number of leaves: 9 Size of tree: 17

Certainly, these are promising results. Can they be improved? Logically, if the data has its missing values filled in then the results are likely to be different. Whether different means better however, is yet to be seen. I will re-do the crawling, making sure I have less than 200 examples in total, then compare the results generated by the loss of half the data. Then, I will fill in the human-generated values manually, and compare the results generated by a smaller dataset with more filled in attributes to a larger one with only three.

### Sunday 4<sup>th</sup> December

Today I went back through my data mining results from yesterday, and noticed that I had only ran K-nearest neighbour when  $K = 1$ . So I decided to run the algorithm again, with different K values this time.

1. 95.2681%
2. 94.0063%
3. 93.8486%
4. 93.0599%
5. 92.9022%
6. 92.2713%
10. 90.2208%
20. 85.489%

Hence, we can see that the predictive accuracy steadily decreases as K increases. What does this mean, to have  $k = 1$  give the highest predictive accuracy? Is my dataset too simple? The decrease in predictive accuracy is not linear, but rather logarithmic – it is likely to tend towards some minimum.  $K = 30$  gave the same predictive accuracy as  $K = 20$ , so I can assume that's the lower bound.

However, the dataset I plan on filling in with manually-gathered data must, by necessity, be smaller than this one. Hence, I will crawl for a dataset containing at most 200 examples, of which 35 will be phishing and 165 non-phishing, so as to maintain a similar 6:1 ratio of benevolent:phishing.

## Wednesday 14<sup>th</sup> December

After some time off from my project to work on other modules, I crawled for a dataset matching the size and maintaining the ratio described above. After crawling for this data, I ran the same algorithms described above, and got virtually the same results, maybe differing by a percentage point at maximum. This was interesting, so I asked Dr Bryant about it in the next meeting I had with him, and he said that it made sense since the ratio between the class values is the same, and the number of attributes is also the same.

In addition to this, I also steadily worked through the dataset, filling in the human-generated parts – how similar the site looks to a known benevolent website, and whether it makes the users any offers as part of a request for information. Once I had this compiled dataset, I ran it through weka again. The results are shown.

Algorithm	Predictive accuracy	Notes
0R	82.9146%	
1R	92.9648%	Selected redirection as attribute to test
Naive Bayes	93.4673%	
K-NN	90.4523%	K value of 1
J48	92.4623%	Number of leaves: 2 Size of tree: 3

Hence, the maximum possible predictive accuracy decreases, but is consistently being achieved or almost achieved. I will bring these results to Dr Bryant tomorrow, and discuss their implications then.

## Saturday 17<sup>th</sup> December

During my meeting with Dr Bryant, I was advised to re-run the K-NN test for multiple values of K, to detect noise. The results of this test, for  $K = \{2 \dots 10\}$  are shown below.

2. 88.4422%
3. 91.4573%
4. 91.4573%
5. 92.4623%
6. 92.4623%
7. 93.4673%
8. 93.4673%
9. 93.4673%
10. 93.4673%

We can see the predictive accuracy has dipped somewhat, but then increases. Hence, there is likely some noise in the data that went undetected when  $K = 1$ , but was perhaps overly compensated for when  $K = 2$ . And as  $K \geq 3$ , the metaphorical bumps in the road evened out, and the predictive accuracy increased beyond the initial value, when  $K = 1$ .

Furthermore, most of the examples within this dataset are non-phishing, with a ratio of 6:1 benevolent:phishing. Hence, I should try again, with different ratios. As a starting point, I will try 1:1 and 3:1 ratios.

Furthermore, the selection of the sites could be a factor in the predictive accuracy being what it is. Most of the benevolent sites taken from this sample were webcomic websites such as xkcd or poorly drawn lines. Perhaps a sample taken from a different part of the internet, such as firstyearmatters, would provide different data and hence different results.

As such, I will compile two further datasets. One with a phishing:benevolent ratio of 1:1 with the benevolent examples starting from firstyearmatters, and one with a 3:1 ratio, with benevolent sites starting from amazon.co.uk.

### **Sunday 18<sup>th</sup> December**

Today, I started compiling data for my next two experiments. I decided that today would best be spent gathering the computerized elements for both datasets, then work on the human elements one at a time. Gathering the computerized attributes will take some time, hence I decided to start reading some research papers in the area, to build the foundations for my literature review.

Akanbi et al (2015) was the first one I looked at, and proved useful for finding algorithms to look at. The main classifiers referenced are C4.5, SVM, KNN and LR. Out of these, I have only used KNN, and C4.5 is not implemented within weka. Hence, Linear Regression and Support Vector Machine are probably worth investigating, to compare the results generated by these classifiers with the ones created by 1R, naive bayes, KNN and j48. Furthermore, “K-NN performs better with a decreasing size of dataset while classifiers like SVM and C4.5 performs better with increasing size of dataset.”. This is very interesting, and could bear some further investigation in my specific case, to see where the line is likely to be. However, I will primarily be doing experiments with 200 or fewer examples, out of sheer necessity. As the number of examples increases, the number of pages I will need to manually verify also increases, which is tedious work.

### **Monday 26<sup>th</sup> December**

In between this log and the previous one, I decided to repeat the experiments with the 1:1 and 3:1 ratios described above. The results are below.

#### 1:1 benevolent:phishing

Algorithm	Predictive accuracy	Notes
0R	50.7463%	Baseline accuracy – other algorithms should exceed this
1R	79.1045%	Offers* selected as attribute to test
Naive Bayes	82.5871%	
J48	83.0846%	Number of leaves: 10 Size of tree: 16

And the K-NN accuracies, for K = 1-10 are listed below.

1. 82.0896%
2. 80.0995%
3. 84.5771%
4. 81.5920%
5. 82.0896%
6. 83.0846%

7. 83.5821%
8. 82.5871%
9. 81.0945%
10. 80.5970%

### 3:1 benevolent:phishing

Algorithm	Predictive accuracy	Notes
0R	69.8492%	Baseline accuracy – other algorithms should exceed this
1R	86.9347%	Offers selected as attribute to test
Naive Bayes	86.4322%	
J48	84.9246%	Number of leaves: 8 Size of tree: 12

And the K-NN values, again for K = 1-10:

1. 83.9196%
2. 81.4070%
3. 86.4322%
4. 85.4271%
5. 85.9296%
6. 87.9397%
7. 85.9296%
8. 84.9246%
9. 87.4372%
10. 87.4372%

I noted earlier that the high predictive accuracies from the 6:1 benevolent:phishing dataset could be the result of a fluke, or data that is not representative of the overall universe of examples. Having repeated the experiments and gathered largely the same conclusions, namely a predictive accuracy consistently higher than the baseline provided by zeroR and it's "always predict the majority class" function, I feel more confident in saying that the results gleaned initially were not a fluke, and are in fact indicative of some wider trend. Ideally, this trend would be that the attributes I have chosen are good attributes, and the next stage of the project would be to find an algorithm that provides even better accuracies across this and other datasets. However, all that is currently shown is that the selection of attributes I have made gives high results across approximately 600 examples of data. This is likely to be generalizable across the entire universe of examples, but perhaps there are underlying structural problems with this setup. Would different attributes, with different possible ranges of values, provide higher accuracies?

For example, something I noticed when exploring the datasets during preprocessing is that the most common value for the "spelling errors" attribute was 0. For instance, in the 3:1 dataset of 199 examples, 153 of the examples had 0 spelling errors. This indicates that spelling errors is not a very reliable attribute, and could be replaced by something more reliable.

However, I will now take a break from my project in order to write my progress report. I believe I have made a solid start to the project, and attained promising results consistently across multiple test sets. Furthermore, my plan has changed marginally – the removal of the popups attribute



removed the need for virtualbox, I gathered URLs from a spam email archive (<http://untroubled.org/spam/>) instead of trying to attract said emails myself through an unsecured windows box, my time plan changed slightly and I changed the structure of the data to gather.

Furthermore, I have a challenge to complete – get a higher predictive accuracy – and avenues for solving this problem; replacing the spelling errors attribute with something new and using a different algorithm. This combination, plus the justifications for the various technologies I used inherent in the literature review, should provide me with a coherent structure for my progress report and a solid start on the next phase of the project.

## **Tuesday 24<sup>th</sup> January**

I had previously found, through Akanbi et al, that K-NN is a good algorithm to use for smaller datasets like the ones I'm using. My preliminary results seem to bear that out. My task now is to look through the research papers I've accumulated for other algorithms to use. Once I've looked through the existing stock of papers on my laptop, I'll search for more, using the Witten book as a starting point.

First, I will read Akanbi et al more thoroughly in search for hooks.

(Alnajim and Munro, 2009): 4 different types of phishing.

1. Email-to-email: when someone receives an email requesting sensitive information to be sent to the sender.
2. Email-to-website: when someone receives an email embedded with phishing web address.
3. Website-to-website: when someone clicks on phishing website through a search engine or an online advert.
4. Browser-to-website: when someone misspelled a legitimate web address on a browser and then referred to a phishing website that has a semantic similarity to the legitimate web address.

Worth a look at this paper to see if it details methods for detection, but not too thoroughly.

Garera et al. (2007) proposed SpoofGuard which looks similar to my project but a different set of attributes. Definitely add this to the list!

Kim and Huh (2011) was the one that showed K-NN was a good algorithm. Worth looking at, since the algorithms that lost out to KNN in this study may beat it here.

Basnet et al. (2008) used Artificial Neural Networks to obtain high detection rates. Look into this – is ANN an algorithm or a completely different method entirely?

Toolan and Carthy (2009) used classifier ensemble methods to improve hit rates – is that just using multiple algorithms and using the highest accuracy ratings?

So that's quite a list to check out. However, I will build this list up further by looking at other papers before tackling it. Next up, Islam and Abawajy (2013):

This paper proposes a “multi-tier” phishing detection approach, and is not nearly as long as Akanbi et al, so I do not anticipate finding nearly as much here.

This paper seems to define “multi tier” as using multiple algorithms, so this could be more useful than I thought. So what’s an ensemble?

Paper cites (Zhang et al., 2003; Sahami et al., 1998; Spamassassin public corpus, 2006; Drucker and Shahrari, 2003; Phishing activity trends report, 2005; Cohen and Singer, 1999; Cristianini and Shawe-Taylor, 2000; Kaitarai, 1999; Huan and Lei, 2005; Islam and Zhou, 2007, 2008; Islam et al., 2009; Phishingcorpus homepage, 2006; Anti Phishing Work Group, 2007; Abu-Nimeh et al., 2006; Bergholz et al., 2008; Fette et al., 2007; Kirda and Kruegel, 2006; Kruegel et al., 2005; Ludl et al., 2007; Ross et al., 2005; Wenyin et al., 2005; Quinlan, 1986, 1996; Witten and Frank, 2005; Mitchell, 1997; DeLong et al., 1988; André et al., 2010; Yu et al., 2009; Beck and Zhan, 2010; Liping et al., 325 2009; Hamid and Abawajy, 2011; Crain et al., 2010; Neville and Horie, 2011; Toolan and Carthy, 2009; Shahriar and Zulkernine, 2011; Huang et al., 2009; Haijun et al., 2011; Kim and Huh, 2011; Fuet al., 2006; Toolan and Carthy, 2010; L’Huillier and Weber, 2009; Abu-Nimeh et al., 2007; Miyamoto et al., 2009; Fette et al., 2007; He et al., 2011; Vishwanath et al., 2011; Al-Momani et al., 2011; Microsoft, 2005; Mozilla. Phishing protection, 2007; Netcraft, 2007; Moore and Clayton, 2007).

Obviously I can’t read ALL of those with anywhere near the degree of thoroughness required to actually get anything from them. However, I know what I’m looking for – alternative algorithms that give a high predictive accuracy on small and large datasets. Eventually, the dataset that will train the algorithm will grow in size to a point where KNN is outclassed.

The paper does mention Support Vector Machines (SVM) as an algorithm to use. Akanbi et al also mentions this, so perhaps it is worth looking at.

The next paper is He et al (2011). Unfortunately, this paper proposes a method for detecting phishing websites based on the identity they claim, and the DOM model, and therefore does not contain any new insights that can be used in my project.

### **Wednesday 25<sup>th</sup> January**

Today I’m continuing my search for an algorithm better than K-nearest neighbour. The next paper on the list is Hadi et al (2016), which proposes a fast associative classification algorithm for detecting phishing websites. Unfortunately, as associative classification rule mining is a previously unknown topic, there are no associative classifiers implemented within WEKA.

Furthermore, the paper discusses the performance of the new algorithm only in contrast to other associative classifiers, so there is no information here that can be of use.

Next up is Xiang et al (2011), which proposes a new machine learning framework for detecting phishing sites, called CANTINA+. This framework filters pages based on whether they have a login form or not – since a website without one cannot request any information and therefore cannot phish – and then uses machine learning algorithms to categorize the site as phishing or benign based on features of the site. This paper focuses on minimizing the false positive rate, but there could still be some discoveries to find within it.

Of particular note is the use of hash-based detection of near-duplicate pages. This is based off of the knowledge that phishers can create hundreds, or even thousands of websites in a very short space of time, all with identical HTML. Once noise such as variable names, spaces and newlines are removed from the HTML, the SHA-1 hash is then computed and the result compared to an existing bank of phishing site hashes. This approach, while interesting and useful as a detection technique in its own right, does not help me.

The machine learning algorithms used in this paper are support vector machine, random forest, J48 and Bayesian networks.

These names keep coming up in the research, again and again. Therefore, I need to look for implementations of them, ideally within WEKA. Even if they are not part of the standard WEKA package, since it is open source, it is possible that an external package has been written as an extension to WEKA that does implement this algorithm.

SVM is implemented within WEKA, so this is good.

Random Forest – also there.

J48 – I’ve already used this algorithm for the preliminary data mining, so this is there.

Bayesian Network – check.

Therefore, I will need to re-do the data mining from earlier, using the same datasets, but the following algorithms:

0R

Bayes Network

SVM

Random Forest

J48

K-NN for K = 1-10

## Thursday 26<sup>th</sup> November

Today, I will be re-doing the data mining on the 6:1, 3:1 and 1:1 datasets, using KNN for K = 1-10, SVM, RF, J48 and Bayesian Net. I will also include ZeroR as a baseline.

### 1:1

Algorithm	Accuracy	Notes
0R	50.7463%	Baseline accuracy
J48	83.0846%	Number of leaves: 10. Size of tree: 16
RF	84.5771%	100 trees, considering 3 random features. Out of bag error: 0.1244
SVN	83.0846%	Number of kernel evaluations: 5672 (71.202% cached)
Bayes Net	82.5871%	

And the K-NN accuracies, for K = 1-10 are listed below.

1. 82.0896%
2. 80.0995%
3. 84.5771%
4. 81.5920%
5. 82.0896%
6. 83.0846%
7. 83.5821%
8. 82.5871%
9. 81.0945%
10. 80.5970%

### 3:1

Algorithm	Accuracy	Notes
OR	69.8492%	Baseline accuracy
J48	84.9246%	Number of leaves: 8. Size of tree: 12
RF	82.4121%	100 trees, considering 3 random features. Out of bag error: 0.1508
SVN	84.9246%	Number of kernel evaluations: 10499 (90.849% cached)
Bayes Net	87.9397%	

And the K-NN values, again for K = 1-10:

1. 83.9196%
2. 81.4070%
3. 86.4322%
4. 85.4271%
5. 85.9296%
6. 87.9397%
7. 85.9296%
8. 84.9246%
9. 87.4372%
10. 87.4372%

### 6:1

Algorithm	Accuracy	Notes
OR	82.9146%	Baseline accuracy
J48	92.4623%	Number of leaves: 2. Size of tree: 3
RF	94.4724%	100 trees, considering 3 random features. Out of bag error: 0.0603
SVN	92.9648%	Number of kernel evaluations: 8192 (89.099% cached)

Bayes Net	93.9698%	
-----------	----------	--

And the K-NN values, again for K = 1-10

1. 90.4523%
2. 88.4422%
3. 91.4573%
4. 91.4573%
5. 92.4623%
6. 92.4623%
7. 93.4673%
8. 93.4673%
9. 93.4673%
10. 93.4673%

The maximum accuracy values for these results differs with each dataset. For the 1:1 ratio set the highest predictive accuracy, 84.5771 is given by Random Forest, tied with K-nearest neighbour with a K value of 3. For the 3:1 dataset, the maximum accuracy is 87.9397, given by Bayesian Network and 6-NN. And for the 6:1 dataset, the maximum is 94.4724, also given by Random Forest, but unmatched by K-NN.

These differences mean that there is no one algorithm that is supreme in terms of accuracy across all the possible examples, and that the deployment phase will need to use many different algorithms, and decide on the most favourable outcome.

Since users will be inputting their own data, which will be classified and then used to update the classifier, the algorithm used needs to be updateable. Of the algorithms presented, only K-NN implements WEKA's UpdateableClassifiers interface, which marks the algorithm as capable of updating itself one instance at a time. This means that if the other algorithms are ran, they will have to be re-trained, re-tested every time a user wishes to classify a site. At present, training one algorithm and performing ten-fold cross-validation takes less than a percent of a second, but as the size of the dataset grows, performance is likely to drop.

### **Wednesday 1<sup>st</sup> February**

Today, I started work on my dissertation. I wrote a 225-word abstract, and will consider how to divide the remainder of the work into chapters. As my project is a blend of data mining and software development, I can make limited use of the software development model for the dissertation presented in the project guide. However, I will also need to present my data mining experiments and analyze their results, and come to conclusions. I may need to do this multiple times throughout the dissertation – I will ask Dr Bryant at the next meeting for advice on this. The hope is that, while I will need to include sections for software development design, implementation and testing, these will be smaller than on a purely design and develop type, which will free up space for talking about my data mining experiments.

Furthermore, I received feedback from Dr Bryant about my progress interview, and one of the main points was that I should have known more about how K-NN works, and why setting an even value of K shows that I'm not really understanding the algorithm. (The answer is that K-NN compares the K nearest neighbours to the unseen example to perform classification, and an even value of K raises the potential for ties)

## Thursday 2<sup>nd</sup> February

Today I am thinking more about how to divide my dissertation up into chapters, with specific focus on the literature review, as I have already done one for my progress report and can take significant inspiration from that. Furthermore, I can also start writing up my experiments, as I have achieved results from algorithms and discussed their implications both in my log book and my meetings with Dr Bryant. Unfortunately, I neglected to bring my tablet to my last meeting in which the methodology for writing up an empirical experiment such as this one, so I will need to look it up. And also make sure I bring my tablet, or at least a pen and paper.

When writing up an experiment, the first step is to describe the purpose of the experiment – the reason for doing it, or what I hope to learn. In my case this would be “To compare algorithms X, Y and Z on dataset D”. Then comes the hypothesis, or what I think will happen and why. In this case, I think that algorithm X will trail behind algorithm Y in terms of accuracy, and that due to the parameters set within Z, it will edge out algorithm Y.

The website I am using, <http://teachertech.rice.edu/Participants/louviere/Lessons/les1.html> is a writeup guide for a physical experiment using materials and dependent on a fixed procedure. I do not have these to worry about, so will ignore the procedure. My procedure is simply to run the algorithms on the dataset within WEKA.

Once I have some results, the next step is to analyse them. This involves thinking more about why the results are the way they are, and what this means in terms of the project. For instance, if the results do indeed match the hypothesis, then why, and if they do not match the hypothesis then how big is the difference and what in the inputs could have caused this. Finally, I need to draw a conclusion from the analysis, so the analysis needs to be tending towards a conclusion. The conclusion could be that algorithm Z is better for dataset D, and therefore shall be tested on dataset E as well, while algorithm Y, due to its inferior performance, will not be included.

## Saturday 4<sup>th</sup> February

Today, I looked into how WEKA is constructed, and how to use it as an external library for my own Java code, in preparation for combining the data mining model and the web crawler, and adding a user interface. I considered that the first step in learning how to use the system in this new context is to dive straight in – I’m familiar with use of external libraries, so I should be able to build a throwaway java program that runs an algorithm on one of my datasets.

Using weka’s library was simple enough – create an Instances object using a DataSource object representing the arff file to pass into the model. Everything seemed reasonable up until it was time to cross-validate the trained model. WEKA’s API offers a `Evaluation.crossValidateModel()` method, but it doesn’t output or return anything. Perhaps the Evaluation class can then be used to classify new instances?

After looking further into the inner workings of the `crossValidateModel()` method, which accepts a classifier, the number of folds, and a dataset, and takes care of the work of training and testing the classifier on the dataset, it appears to work almost like the OpenGL state machine I worked with in Computer Graphics last semester. I would pass in a classifier, and then I could use that classifier – in this case, J48, but any algorithm should, in theory, work – to classify new examples.

When classifying a new instance, I first create an Instance object and associate it with the dataset, to give the new instance information about its attributes. Then I set each attribute value, passing in made-up data for the purposes of this test. Finally, I tell the J48 algorithm – already pre-trained and tested from earlier – to classify this new instance.

Unfortunately, I get a less than helpful error message in response.

```
Exception in thread "main" java.lang.NullPointerException  
    at weka.classifiers.trees.J48.classifyInstance(J48.java:311)  
    at WekaTest.main(WekaTest.java:39)
```

This tells me that there was an exception when classifying the instance, caused by some variable being null when it shouldn't have been, but not much else.

After some digging, I found the cause of the problem. The cross-validation did NOT result in a built classifier – rather, the results of the testing – predictive accuracy, f-measure, ROC curves etc – were loaded into the Evaluation class. As such, I needed to build the classifier again, and then I could classify the made-up instance.

After the instance was classified, I wanted to then add it to the dataset, re-do the cross validation, which should update the predictive accuracy, then use that new value as the baseline accuracy for future runs of the software. Unfortunately, adding the new value to the dataset and re-running cross validation did not change the predictive accuracy. Perhaps the addition of a single new instance simply did not make a big enough difference to the accuracy? I will try again, but instead of adding a single instance, I will combine two of my datasets. Hopefully that will cause a bigger change in the predictive accuracy.

### **Monday 6<sup>th</sup> January**

Today I had another meeting with Dr Bryant, in which we talked about how to write up experiments. Turns out I was writing mine up all wrong, and when doing preliminary algorithm testing to get some baselines, I don't need to go to the trouble of writing it up formally. However, what I have done is still useful, if it can be rephrased to refer to setting a baseline and doing some exploratory data mining, which I'm fairly certain shouldn't be that hard.

Furthermore, I can start my literature review in a similar manner to my progress report – I can talk about Maven as a build tool now that I'm thinking more about the software, and go into more detail on why open source programs are better than closed source.

Another thing that I brought up during the meeting was that I had tested the predictive accuracy of one of my datasets using WEKA's API, and I was advised to ensure that calling the cross validate function from the API gave the same results as doing it from the explorer interface. Fortunately, it did indeed.

### **Thursday 9<sup>th</sup> February**

Today I worked primarily on my dissertation, finding references for my literature review. My project context in terms of existing research in the progress report was limited to a single paragraph, so I need to do better within the dissertation itself. Fortunately, I have Islam and Abawajy as an example and a guide. The title is shown below, on its own line, for easy reference should I need to find the paper again.

“A multi-tier phishing detection and filtering approach” ( Islam and Abawajy, 2012)

I mentioned this paper in an early log as being the source of a veritable mountain of papers for reference, and so I will need to start digging through this mountain in order to add to my current collection and find relevant papers.

I attended a seminar on doing a literature review last semester, and I learnt that a basic, well-formed structure is to start with the broad area the paper's about – in this case using machine learning to detect phishing – then narrow the scope a little – maybe to detecting phishing websites specifically – then narrow the scope further, perhaps to detecting phishing websites using certain methods. These methods can then be critically compared to mine, thereby identifying the gap in the available literature that is filled by my project.

Strictly speaking, I do not NEED to fill a gap in available research, because that is MSc and PhD level work, but if I can find a way to phrase the conclusion of my literature review as “this is a thing noone else has tried before” that is sure to stand me in good stead come marking.

As such, I trawled through the reference list of the Islam and Abawajy paper, looking for papers both vaguely related and specific to my project's area. I will read the abstracts, introductions and conclusions of each paper before deciding if a specific paper is worthy of inclusion in my dissertation.

### **Saturday 11<sup>th</sup> February**

Today I worked more on the literature review for my dissertation, beginning to move beyond citing as many papers as I can find that relate to a certain area or superset of my project, and looking at honing in on specific uses of supervised classification data mining to classify websites as phishing or benign. As stated earlier, my hope is to find papers that do nearly the same thing as my project, so I can then talk about how their methods differ from mine, and how my project fills in the gaps in the literature.

Effectively then, my work today and tomorrow will be to actually do what I have talked about in the previous entry with regards to the literature review. I hope to finish this part of the project context this weekend, and move on to discussing the various technologies I've used, the alternatives, and why I chose the technology I have, which will comprise the second part of this section.

### **Monday 13<sup>th</sup> February**

Today I start work on the user interface. During my meeting with Dr Bryant, he asked when he could expect to see a working prototype of the system. At the time, I had not yet considered prototyping, and thought to simply demonstrate the system when it was complete. However, after thinking about it, I could throw together a quick UI, set it to generate random predictions with random accuracy levels, and demonstrate that.

In my progress report, I said that I wanted to use GWT as the UI framework. However, for the purposes of this demo, I will build a quick and dirty user interface in JavaFX. Unfortunately, the java JDK I had installed did not appear to contain JavaFX, so I needed to do some troubleshooting. After installing the package, I had an access restriction on the javaFX classes I was hoping to use. The reason for that is that the Java licensing agreement states that nothing other than Java itself may



use package names with 'java' as a top level name, and eclipse mistook the javafx package for an external library. Deleting the JDK and re-adding it fixed the problem.

For today, I simply created 3 labels, 3 text boxes and a button, and aligned them correctly. Tomorrow I will make the button actually do something, by generating a random prediction and displaying it.

## **Tuesday 14<sup>th</sup> February**

Today I added an event handler to the button, along with two extra labels – one for displaying the prediction itself, and one for displaying the predictive accuracy. As of right now, random predictions are generated with random accuracies, with no extra formatting. I will colour and enlarge the prediction messages, and perhaps add some flavour text depending on the prediction and accuracy. Then, I will add some simple URL filtering, refusing to generate a prediction if the URL text is empty or doesn't start with "http(s)://".

## **Thursday 16<sup>th</sup> February**

During my last meeting with Dr Bryant, I asked about dissertation feedback. He told me that he only generally did one round of feedback in order to manage workload, and asked me to send a copy of my contents page, since that's effectively the structure of the dissertation and can be used to glean useful information, such as if I'm going in completely the wrong direction or not.

This made me realize that I had not actually written a contents page, I had just started writing parts that I knew would be required, such as the literature review and preliminary data mining results. As such, I decided to write the contents page today. I made an attempt to follow the project guide when writing it, but needed to make some changes due to the nature of my project as a software development/data mining hybrid.

I had an introductory chapter, then a chapter for the literature review, that was divided into sections for the actual literature review – referencing papers and putting the project into context against existing research – and the technical review – justifying all the various pieces of technology I used against the alternatives.

Then I had a requirements and design chapter, splitting requirements and design into separate sections and separating each of those into subsections for each subsystem and another subsystem for the system as a whole.

Then came an implementation chapter, which I split into two parts. Part one was talking about the web crawler, sectioning it off into code and unit tests, with subsections for each attribute to gather, followed by a subsection on the crawler as a whole.

In between the two parts was a chapter on the data mining, which had a section for the data gathering, a section for the preliminary data mining – algorithms such as 1R, Naive Bayes and K-NN – a section for the later data mining – SVM, RF, J48, Bayesian Networks – a section for the experiments – which is preliminary at this stage, since I have not yet done any experiments, and only have the results from the two iterations of data mining. I could, however, write up experiments on the number of trees in random forest, or the K value in K-NN. Then there was a section on the final decision on the algorithms to incorporate into the final system. I am considering splitting this

chapter into two because of its size, but cannot yet see where a logical place to make the split would be.

Next, there is the second part of the implementation and testing. This is split into building the UI, then integrating the three parts together into a coherent whole. This is further split into combining the crawler and the data mining model, then combining the crawler and the UI, then testing the complete system with positive and negative tests.

Finally, there is a conclusions chapter, divided into the critical evaluation and conclusion as defined in the project guide.

### **Monday 20<sup>th</sup> February**

Today, I had a meeting with Dr Bryant. He asked about the design documents for the user interface, which I had not done, and any accessibility concerns I had taken into account, which I had not yet done, as this version of the user interface was just a prototype. However, this has given me something new to think about – namely, what kinds of accessibility features I will add. Large fonts are a must – I will likely offer at least four different font sizes. Furthermore, I found a Java text to speech API, but with the user interface as it is, it would not be very helpful – therefore, documentation explaining how to use the software should be built into the software itself, as well as available as javadoc.

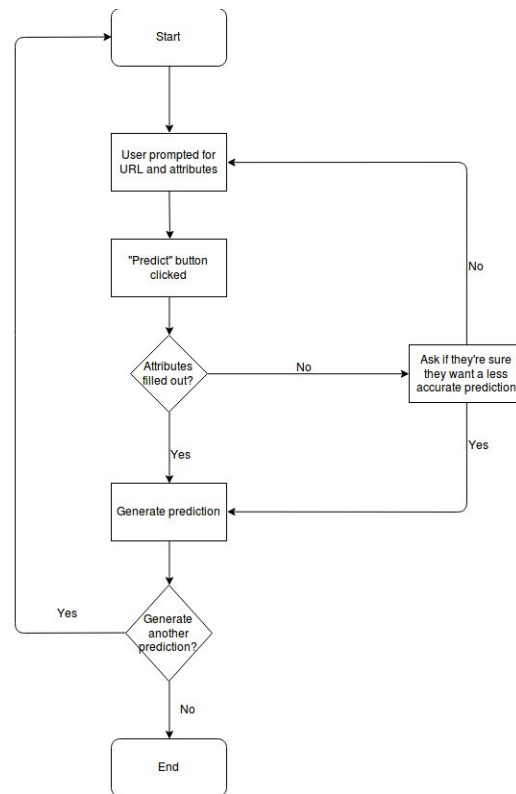
Furthermore, we read through some MSc dissertations for inspiration on how to write up empirical experiments, and while I'm still at somewhat of a loss on WHAT to write up, I now understand very well HOW to write it up.

Therefore today I will focus on creating options for the user interface, first for varying font sizes, then writing up the help page(s).

I created a simple menu bar at the top of the screen, that at present has an options menu allowing users to change font sizes. I offered 4 separate font sizes, which I hope is adequate as a starting point. Furthermore, I started work on the help page/system documentation, making sure to keep in mind that this system is targeted at non-technical laypeople.

Tuesday 21<sup>st</sup> February

Today I considered the design of the user interface. I built a flowchart of the expected event flow:



Next, I will create a simple wireframe diagram of the user interface, to be presented to Dr Bryant at the next meeting. I also need to block some time aside for dissertation work, lest I fall behind – I’m visiting Manchester’s John Rylands library tomorrow with a friend, who says its a good place to knuckle down and get some work done.

URL:	<input type="text"/>	This site is/is not a phishing site
Offers:	<input type="text"/>	
Look and Feel:	<input type="text"/>	
<input type="button" value="Predict"/>		
		PhishGuard is XX.XXXX% confident in this prediction.

## Thursday 23<sup>rd</sup> February

Today, I finished the help documentation for the program. I gave the system a tentative name – PhishGuard – and wrote the help page as if I was a corporate entity writing a manual for a system. Hopefully, this system will be easy to use, but if it is confusing, then the manual page should help.

I also attempted to get a text-to-speech function working. I tried two APIs – FreeTTS and Google’s TTS translator API. Unfortunately, both of these libraries threw an exception somewhere within the body of the library during testing, and the source code was not available in either of the jars I was using, so I could not examine the code to see if I have used it wrong.

I asked a friend what other accessibility mechanisms I could use in case TTS truly doesn’t work out, and they recommended background colour changes to help colourblind people. As such, I will implement 3 basic “skins” - black and white, white and black, default. Furthermore, in an effort to help dyslexic people, I will change the font used to be a dyslexia-friendly one. I know they exist – comic sans is one. While comic sans has a reputation of being a “silly” font, unsuitable for serious academic projects like this, I know vaguely of others. I will do further research tomorrow.

## Saturday 25<sup>th</sup> February

Instead of doing research into accessibility, I decided to finally start the process of hooking up the data mining model to the user interface. Firstly, I merged all 3 of my datasets together to create a large dataset with 600 examples. Then, I needed to think about the algorithm to use on the data, or if I should use more than one. Hence, the table of algorithm results on this merged dataset is presented.

Algorithm	Predictive Accuracy	Notes
ZeroR	67.7796 %	Baseline accuracy
Random Forest	90.6511 %	100 trees - default
K-NN	87.4791 %	K = 1 – default
1R	85.6427 %	Minimum bucket size – 6
Naive Bayes	86.8114 %	
Bayesian Network	87.813 %	
SVM	87.4791 %	No logistic models, c = 1, checks, tolerance = 0.001
J48	89.8164 %	

Therefore, random forest appears to give the highest predictive accuracy, and the number of trees in the forest can be varied to see which gives the best results. I decided to use my newfound understanding of WEKA’s API, and automate this variation.

Unfortunately, the GUI version of WEKA and the API are different – 3.6 and 3.8 respectively. While this has not caused any problems up to now – in fact, I did not realize until now – there are differences in the implementation of random forest that makes the setting of the number of trees confusing. As such, I upgraded the GUI version and will re-run the above algorithms, and if random forest is still on top, I will try again.

Algorithm	Predictive Accuracy	Notes
ZeroR	67.7796 %	Baseline accuracy
Random Forest	90.1503 %	100 iterations - default
K-NN	87.4791 %	K = 1 – default
1R	85.6427 %	Minimum bucket size – 6
Naive Bayes	86.8114 %	
Bayesian Network	87.813 %	
SVM	87.4791 %	No logistic models, c = 1, checks, tolerance = 0.001
J48	89.8164 %	

As can be seen, the two tables are identical except for random forest, which has dipped slightly in accuracy but is still the top ranker. J48 is also very high up, and K-NN could potentially be, with different values for K. As such, I will automate RF with every 10 iterations and K-NN with the odd numbers from 1-13.

Within the following results tables, the first number is the variable being changed and the second is the predictive accuracy.

Random Forest:

10 89.1486%  
20 89.6494%  
30 89.4825%  
40 89.8164%  
50 89.6494%  
60 89.6494%  
70 89.8164%  
80 89.9833%  
90 89.9833%  
100 90.1503%  
110 90.3172%  
120 90.1503%  
130 90.3172%  
140 90.1503%  
150 90.1503%  
160 90.4841%  
170 90.4841%  
180 90.3172%  
190 90.1503%  
200 90.3172%

K-NN:

1 87.4791%  
3 85.9766%  
5 85.9766%  
7 86.1436%  
9 87.4791%  
11 87.6461%

Performing these experiments has shown that Random Forest consistently out-performs K-NN on this merged dataset, and while there are performance gains to be had from varying the number of iterations (which I think roughly corresponds to the number of trees in the forest? Instead of generating X trees once, generate one tree X times?) the gains are slight.

The true question now is will this predictive accuracy continue to remain top of the list when the dataset grows, and as the ratio of phishing to benevolent sites changes? Ultimately the answer to that can only be known after the fact, and while I eventually want to have multiple algorithms that are kept up to date and cross-validated, Dr Bryant asked for a prototype, and it would be in keeping with the agile methodology I am using throughout this project to have a working system using a single algorithm before moving on to support for multiple. When multiple algorithms are supported and themes are changeable, I will consider the practical portion of the project complete and can move on to the dissertation.

### **Sunday 26<sup>th</sup> February**

Today I began the process of integrating the model and the UI. WEKA offers a facility to save a generated model as a file and then reload it, so I brought the saved RF model into my codebase. Then I started thinking about what was necessary in order to classify a new instance. The new instance would need context – it would need to be able to say “I am part of THIS dataset, treat me as such”, so I brought in the combined data.

Then, I needed to modify the way I gather data. Previously data was gathered from a file or an unchecked crawl, and the data was written to a file. I needed a crawler that would gather data from a single site, and make it available to the UI. Hence, I wrote in the necessary functionality to crawl in this way. The gathered data cannot be returned to the UI directly, so instead I had a single instance of a class representing the gathered data that could be written over. Then I could access that data from the UI. This does have some problems in an industry context – in industry, there would likely be one powerful server with this program on that all the users access, and if data is changed by another user before the first user is finished, then incorrect data is returned. However, this is not an issue for me, as PhishGuard as it is designed to be used by a single user. Just something to mention in the critical evaluation section of the dissertation.

After that, I set the crawler to crawl a single site, given by the provided URL. This would assign values to the ArffData class, which can then be picked out and used to create a WEKA Instance. This Instance would then be classified using the saved model, with the result being outputted to the screen. Unfortunately, the application crashed when testing the new functionality so I needed to do some debugging.

The problem turned out to be that missing values had to be set explicitly – if the value is not valid, then I have to call `setMissing()`, otherwise `setValue()`.

Then, I had another problem – my exception handling hinged on any exceptions occurring in crawling the page – which I anticipate to be the most common cause of any issues – does not allow exceptions to be propagated upwards. Any errors must be dealt with within the crawler itself. This is considerably at odds with my requirements, which stipulate that any exceptions are propagated up to the UI, which will display an error message.

This leaves me with few options. I can do the connection manually – now that I am accessing one website at a time, I do not need any of the complex crawler functionality. This will let me write my own exception handling, which I need. However, it feels like throwing away a significant portion of the project – the crawler is one of its three subsystems. However, that can be rationalized away in that what was REALLY important was the specific data I am gathering, and how exactly this was tested for. The crawler architecture merely let me automate this data gathering, in order to do it on the scale necessary for this project.

I will take this to Dr Bryant tomorrow, and get feedback on this. I know the actual connection part of this will be simple – java offers a simple framework for doing this.

## **Tuesday 28<sup>th</sup> February**

Today, I will modify the prediction buttons event handler so that it connects to a single site manually, as was recommended at the last supervisory meeting yesterday. As part of this, I will separate the crawling functionality from the data gathering functionality by moving the data gathering into a new class. The problem there is the redirection, which will likely need manually reimplementing. Again. Ugh.

A java URL object is set to either follow redirects or not. If follow redirection is set to yes, then the redirection is performed silently – there is no indication to the user whether or not there has been a redirect. If follow redirects is set to no, then I can ascertain whether a redirection has taken place by getting the response code, but then I need to reset the URL and reconnect, and doing this manually does not account for chained redirections. However, on reexamining the attributes to gather, I do not need to worry, as redirections are the only attribute that require a straightforward Java connection – Levenshtein can be ran without any connections at all, and spelling errors uses Jsoup's connection mechanism.

Once all this was achieved, the unit tests needed to be refactored and changed to test the data gatherer instead of the crawler. Fortunately, this was simply a case of calling methods from the DataGatherer class instead of the FYPCrawler class, and all tests passed. I then tested this new functionality on the UI, and received a prediction, with no exceptions. Using an empty URL field gave me the error I expected, too. From there, I did some small code restructures in an attempt to make the code more readable, as the contents of the predictButton's handle method were rather complex. It remains so, but some of the work has been offloaded.

## **Thursday 2<sup>nd</sup> March**

Today I wanted to integrate the help documentation I wrote in with the program itself. I wanted this to be in a second window, so I created a second UI class that subclassed Application, in the way that Javafx requires. Then I set the new class to open a HTML file I made out of the help documentation. Unfortunately, the file would not open, and the help screen crashed.

Turns out I was using a method more appropriate to opening web URLs than local HTML files. When I changed my method to account for that, the help screen opened just fine.

Next, I removed the requirement that a popup would appear asking the user to confirm if they have not entered any offers or look and feel data, reasoning that giving this data a missing value is just as valid as any other value. If I were to extend this project, however, I probably would implement this check. Another thing to talk about in my dissertation.

### **Friday 3<sup>rd</sup> March**

Today I resolved to finish the practical portion of the project. Since the last thing to do was the theme changes, this seemed very doable. I implemented three themes – black text on a light background, white text on a dark background, and white text on a dark blue background. Furthermore, I changed the font size changer to change the size of everything – labels, choiceboxes and textboxes.

With this added accessibility consideration, the practical portion of the project is effectively complete. I will show what I have done to Dr Bryant at my next supervisory meeting on Monday and incorporate any feedback, but this weekend I will start on my dissertation again. Hopefully I can get to 2,500 words by Monday.

### **Sunday 5<sup>th</sup> March**

I achieved my word count goal and then some, writing almost 1200 words in two days. I achieved this by working my way down the dissertation structure I wrote several weeks ago in a linear fashion, just as I did for the progress report. I wrote on the motivation for the project, aims and objectives of the project, and then confused the planned approach with the methodology. However, after seeing that there was a section for the planned methodology as part of the literature review, that part was moved into that chapter, and I started talking in some more detail about how I would achieve the project objectives. (TODO ask CHB if that's a good way to go about things for that)

As part of this, I talked about the data mining model, and the methods by which it is measured and tested. I should ask Dr Bryant tomorrow if the way I have done it, where I have compared predictive accuracy against other forms of measurement and cross-validation against other forms of testing is appropriate for the introduction or should be moved to the review section.

Furthermore, the UI section of the project was described as optional, since it does not directly lead towards the aim of the project getting met. I think this is a good plan, but it may not be a good idea to retrospectively assume such things, especially at this point in the project, where the practical work is finished.

### **Monday 6<sup>th</sup> March**

During my weekly meeting with Dr Bryant, I demoed the current version of the user interface. He found it lacking, identifying several areas for improvement before the next meeting. Firstly, the URL input box needed to be lengthened, as even short URLs overflow it. Then, previous predictions should be wiped away as soon as the user types in the URL box, as that is a sign they want to generate a new prediction. Furthermore, the labels assigned to each input were cryptic, to such a degree that a user who'd just looked at the system UI once would have no idea what it was meant to do. Finally, he found it slow, with little indication of progress.

In order to rectify this, I will implement a progress bar at the bottom of the screen, that updates when events occur. I will also rewrite the labels for the inputs, and lengthen the URL box. Furthermore, I will implement some kind of responsive design, so that the URL box always stretches to the end of the screen. I will need to do that for text fields too, in the same way. Vertical resizing can be handled by adding scrolling.



When thinking about the progress bar, an important topic is where it should be and how big it should be. I want to have it at the bottom of the screen, however that requires dealing with resize problems since I don't want the progress bar to be resized – it should move with the resizing application window.

Another potential solution is to have the progress bar fill the entirety of the width of the screen, minus space for a label. Hence, the label describing what's going on would be at the bottom left, and the progress bar would fill the rest of the screen.

Once this layout is considered, the next problem is making it work. The problem is that javafx event handlers are synchronous – the `handle()` function that contains all the functionality hangs the UI until the prediction is ready. Therefore, the solution is to run the `handle` button code in a separate thread. Luckily, javafx provides a simple means to do this. Unfortunately, I have never used concurrency like this before, so I am having to slowly go through and understand how concurrency works within javafx.

The key is an abstract class called `Task`, which is overridden anonymously in the same way that the event handler is. From the looks of the tutorials, I need to override `Task` in my overridden `EventHandler`, move the functionality from `EventHandler` into `Task`, and start the `Task` inside the `EventHandler`.

This led to problems, since the UI could not be directly modified from the new thread. Instead, I needed to call an `updateProgress()` method, and implement a `ChangeListener` that would modify the UI for me. The problem with that is that nothing maps neatly – I would need a lookup table within the `Task`, and the UI class is bloated enough as is!

At this point, I took a break and asked a question on [stackoverflow](#), seeking to understand the concurrency mechanism a little better. Concurrency works in javafx by use of a `Task`, which has certain properties bound to it. Properties can be updated in the `Task`'s `call()` method. `ChangeListeners` can be defined that execute when the value of one of these properties changes. The `ChangeListener` runs in the main javafx thread, and so can modify the UI. As such, binding for example the progress bar's progress property to the `Task`'s progress property and updating the `Task`'s progress within `call()` causes the progress bar to be set to the same progress. The same happens with the progress label. `ExceptionProperties` exist to deal with exception handling, showing an error message in the UI. Finally, a `Result` is returned. This is a new class containing a boolean and a double, and is used to represent the prediction – phishing, or benevolent – and the predictive accuracy. This result is used with another `ChangeListener` to display the final result to the user.

Finally, I rewrote the labels so as to provide a more complete overview of the system, and to give a better understanding of what each option does.

## **Sunday 12<sup>th</sup> March**

Today I prepared for my meeting with Dr Bryant, tomorrow. I will need to demo the new version of the user interface, and show how I have improved it since last week. The complete list of improvements made is listed.

- Moved project to GitHub, implementing version control
- Rewrote labels, providing more information on what system does
- Added progress bar and made it work using concurrency

- A certain degree of responsiveness was implemented – cannot do complete responsiveness because everything must be done manually

## **Monday 13<sup>th</sup> March**

I was given a lot to think about during the meeting today. Firstly, the “measure of confidence” within the user interface – in actuality the predictive accuracy – should be removed, as predictive accuracy only holds any relevance during the testing phase, and when the UI is being used, this is deployment. This also means that the model will not self update and be re-generated – what was created during testing is the final model, with no adjustments further down the line.

With no information other than the prediction itself being displayed, assessors during the demonstration may ask for a justification of the prediction, which would manifest as a visualization of the model. My model was created using Random forest, which generates a prediction using the average results from 100 decision trees and as such may not be possible to fully visualize. Looking through Weka, there is no built in visualization method ,which is annoying.

Maybe random Forest is not the best suited algorithm after all? I was also asked about confidence intervals for the accuracy estimates, which I had not previously thought about. Confidence intervals are a measure of spread, and determining whether or not the predictive accuracy obtained from my datasets is close or far from the true accuracy can have an impact on the true accuracy of the generated model. Ideally, I want to be able to justify changing the model to one that is visualizable in some way, or generates a probability instead of a binary prediction. Looking back to my table of predictive accuracies for the combined training data, I noticed that J48 is very close behind Random forest – perhaps the confidence intervals will provide a justification for swapping Random Forest for J48, which IS visualizable.

Unfortunately, the basic visualization is not particularly efficient, being a list of nodes and the paths down them. This string can be parsed in order to generate a genuine tree model, and the Prefuse toolkit – an extension of WEKA - exists to do this.

Hence, my plan for the week is as follows. I will remove the self-updating features and predictive accuracy measures from the system, perform some minor UI tweaks such as moving labels around, then perform confidence interval calculations on some of the algorithms used in the test to generate the model – exact number and choice TBD – then, if I can justify changing over to a different algorithm, add Prefuse into my application to visualize the resultant model. If I cannot justify a change in algorithm, I will need to explain to Dr Bryant that random forest does not allow for any visualization to be made.

Later today, I modified the system to remove any reference to the predictive accuracy, and added identifier labels to the URL box and choice boxes. Furthermore, I changed the choice boxes default option to say what exactly the user is choosing, instead of just having a question mark.

## **Tuesday 14<sup>th</sup> March**

Today I will be attaining confidence intervals for the highest predictive accuracy measures. For reference, here is the table of accuracies again:

Algorithm	Predictive Accuracy	Notes
ZeroR	67.7796 %	Baseline accuracy
Random Forest	90.1503 %	100 iterations - default
K-NN	87.4791 %	K = 1 – default
1R	85.6427 %	Minimum bucket size – 6
Naive Bayes	86.8114 %	
Bayesian Network	87.813 %	
SVM	87.4791 %	No logistic models, c = 1, checks, tolerance = 0.001
J48	89.8164 %	

Hence, I will obtain confidence intervals for Random Forest, J48, Bayesian Network, SVM and K-NN, as these gave the top 5 accuracy ratings. Random Forest will have 160 trees, and the value of K will be 1, as these give the highest accuracies.

Hence, the next step is to consider how confidence measures will be attained. Dr Bryant recommended the lecture he gave on this as a starting point, and the Witten book as further reading, so I'll start with the lecture.

The relationship between  $p$ , the true accuracy of the model, and  $f$ , the accuracy attained from WEKA, depends on  $N$ , the size of the test set. However, this project is not using independent test sets, but rather cross-validation.

Fortunately, the people who programmed WEKA foresaw this, and included a means to solve this in the Experimenter interface. I set up a new experiment, added a dataset, an algorithm, and set the testing method to 10-fold cross validation, then ran the experiment. When analysing the experiment, I need to understand what it's telling me. I think there's something there that looks like a confidence measure, but I'm not sure. Back to the lecture it is!

After some research, I learn that the Experimenter is best suited for comparing algorithms against each other, so I should try again, with all of the algorithms. Furthermore, I realized that it's performing a paired t-test with all of the results, and not attaining confidence intervals. A paired t-test gives something called a t-value, which can be mapped onto a statistical table to get a p-value. I have heard of things like ( $p > 0.05$ ) as a measure of statistical significance, but have not understood what it means. I shall look through some resources and attempt to understand.

A p-value appears to be the likelihood of seeing a random difference of 5% or more in the results. That is, for a p-value of 0.04, you could expect to see random noise 4 times out of 100, which is good enough to count as statistically significant for most purposes. In my situation, the confidence measure is 0.05 which, since the measure is two-tailed, means there is p% chance of a random difference of 10% of the results. I think.

Changing the confidence measure does not appear to change the results at all. This may be a direct comparison of results across ten times ten-fold cross validation, which is not what I want. Furthermore, all else I seem to have is the standard deviations for each value, so I'm effectively right back where I started.

However, upon reading the Witten book's section on paired t-testing, and retrying with radically different confidence measures, I notice that some of the values have asterisks by them and some do not, and this number increases if the confidence value increases. I remember reading a website on WEKA's Experimenter interface that links the presence of these asterisks to significantly different values, where significance is defined by that confidence value.

As such, the results show that K-NN, OneR and ZeroR all have statistically significant differences in their predictive accuracies from Random Forest. This is to be expected, especially for ZeroR and OneR. The others do not show a statistically significant difference in results. Does this mean I can swap Random Forest for another algorithm, like J48, that is visualizable?

**Test output**

```

Tester: weka.experiment.PairedCorrectedTTester -G 4,5,6 -D 1 -R 2 -S 0.05 -V -result-matrix "weka.experiment.ResultMatrixPlainText" -mean-prec 2 -stddev-prec
Analysing: Percent_correct
Datasets: 1
Resultsets: 8
Confidence: 0.05 (two tailed)
Sorted by: -
Date: 14/03/17 22:16

```

Dataset	(1) trees.RandomFo	(2) trees.J48	(3) lazy.IBk	(4) bayes.Bayes	(5) bayes.Naive	(6) functions.S	(7) rules.OneR	(8) rules.ZeroR
websites	(100) 89.81(3.84)	89.38(3.85)	87.08(3.65) *	87.75(3.75)	87.06(3.93)	87.48(3.81)	85.54(4.03) *	67.78(0.75) *
	(v/ /*)	(0/1/0)	(0/0/1)	(0/1/0)	(0/1/0)	(0/1/0)	(0/0/1)	(0/0/1)

Key:

```

(1) trees.RandomForest '-P 100 -I 160 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1' 1116839470751428698
(2) trees.J48 '-C 0.25 -M 2' -217733168393644444
(3) lazy.IBk '-K 1 -W 0 -A \"weka.core.neighboursearch.LinearMNSearch -A \"weka.core.EuclideanDistance -R first-last\" \"\" -3080186098777067172
(4) bayes.BayesNet '-D -Q bayes.net.search.local.K2 -- -P 1 -S BAYES -E bayes.net.estimate.SimpleEstimator -- -A 0.5' 746037443258775954
(5) bayes.NaiveBayes '' 5995231201785697655
(6) functions.SMO '-C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K \"functions.supportVector.PolyKernel -E 1.0 -C 250007\" -calibrator \"functions.Logistic -R 1.0
(7) rules.OneR '-B 6' -3459427003147861443
(8) rules.ZeroR '' 48055541465867954

```

Attached is the table of results from WEKA's Experimenter interface, showing the results of the paired t-tests, which tested Random Forest against each other algorithm in turn. The columns represent algorithms, and the first row is the algorithm names, the second row is the predictive accuracy for ten times ten-fold cross-validation, the standard deviation in brackets and, if the difference in results between the given algorithm and Random Forest is statistically significant, an asterisk.

I will swap Random Forest for J48, and implement the Prefuse package into my system to visualize the resultant tree. I will also keep the Random Forest model around, in case I have made an error in the statistics and it turns out Random Forest is indeed the best model.

## Thursday 16<sup>th</sup> March

Today I looked at implementing the visualization of my model. In order to do this, I would need to change it to one that can be visualized easily, so I changed to J48, since the previous experiment on statistical significance showed J48's results are not statistically different from Random Forests.

For visualization, I initially went with WEKA's own tree visualizer toolkit, which has its limitations – the resultant tree is static, and the size is fixed, which means using a window of insufficient size results in overlap, to the point of unreadability. Therefore I needed to make the new window very

large. Prefuse is a much better looking extension, and exists as Java source code, and logically must work with WEKA since it is a WEKA plugin, but I could not figure out how to get it working within my application.

Secondly, there is no way to justify any new predictions by means of, for example, highlighting the path taken down the tree to reach the new prediction. I do not know if Prefuse does this, and from a brief play with it through WEKA's explorer interface, it does not allow for any prediction justification. As such, the default visualizer will have to suffice for now. I have asked on StackOverflow if there is a way of doing what I need, but no answers yet.

Furthermore, I wrote more of my dissertation, this time focusing on the technology review – justifying the use of Maven against other build tools like Ant, and VirtualBox against VMWare Player.

### **Monday 20<sup>th</sup> March**

Today I have another meeting with Dr Bryant. There is a lot to discuss at this meeting – I will start with the confidence experiments I have done to demonstrate that the difference in results between J48 and Random Forest is not statistically significant ( $p < 0.05$ ). I will ask for confirmation that I am doing the right thing, and these results do in fact mean what I think they mean.

Then I will talk about visualization, and show the J48 tree. I will need to justify there not being a way to show the path down the tree that was taken for a specific classification, as this does not exist within WEKA and I cannot find any code examples of how Prefuse generates its trees from WEKA.

Lastly I will ask questions about my dissertation. Namely, should the visualization be another, optional objective, or is it too late to change the plan? And how is the summary of the plan written, given that everything specifically mentioned in the plan is already discussed or about to be?

The meeting ended up being significantly focused on the confidence measures, and Dr Bryant took me through an old lecture where he teaches about the paired t-test I have used. Effectively, it was a convoluted way of saying that the methods I used were sound, and the decision to switch to J48 was a well reasoned one.

Furthermore, he concurred that I should not work more on the practical work, focussing instead on the dissertation. He gave one caveat – the visualization tree should be visible all the time, not just when a classification has been made. Furthermore, the visualization button should be explained. These are simple changes, and therefore can be done before considering the project truly complete.

Tonight, I will continue with the dissertation. I hope to get to 7000 words by wednesday evening.

### **Thursday 23<sup>rd</sup> March**

Today was a much more productive day than I had hoped. Now that the practical portion of the project is complete, I can focus solely on the dissertation, and start empirically measuring the quantity of work that I do using the number of words written.

Obviously this is not a perfect measure – it says nothing about the quality of the work, for instance – but prior feedback leads me to believe that the quality of my writing overall is good. I know that

my structure is good, save maybe for the chapter on data mining, where I will be largely planning as I go. But I'll deal with that when I come to it.

Today, I wrote approximately 1,500 words, bringing my total up to 7,500. I wrote about the requirements and design chapter, and almost completed it. I need to finish the subsection on design of the user interface, and then I can move on to implementation.

### **Friday 24<sup>th</sup> March**

Today I completed the user interface design subsection, and with it the chapter on requirements and design. I needed to recreate the wireframe diagram, as the structure of the user interface has changed considerably from my first design. My current word count is approximately 8,250, which means that I am ahead of schedule.

### **Monday 27<sup>th</sup> March**

These log book entries have become shorter and less detailed now that I've moved on to the dissertation, and it is worth explaining why. The practical portion of the project is complete, and all I am doing now is, in effect, the writeup. I'm not trying out new ideas or doing complex, in-depth thought so much anymore, so all there really is to write about is progress updates on the parts of the dissertation I am currently writing about.

Today, I have another meeting with Dr Bryant. I need to ask questions about the dissertation now, since they have gone unanswered for so long while we discussed the user interfaces. I need to ask about the plan summary and how to write that – what I've done is gone through the resources and timeplan, since everything else is either already written or just about to be.

Lastly, with the data mining chapter coming up soon, I need to think about how to write that. Dr Bryant focused on the experiments, but that is only a small part of how I arrived at the choice of algorithm. First I used a few algorithms that we learnt in AIDM on 3 datasets of different ratios of phishing to benevolent. There's an experiment writeup there.

Then, I did the same for the algorithms I found during my research. Another writeup. the problem with these is comparing each algorithm against each other for each dataset. Perhaps I could do a set of graphs for each algorithm's accuracy on each dataset? But then I would need to do the same for the later algorithms, and then for the values of K and trees in RF.

During the meeting we discussed this question, and he agreed that my idea of starting out vague and honing in on the algorithm to use seemed like the best plan. Plus side is it's also a chronological account, which is easier for me to write.

### **Wednesday 29<sup>th</sup> March**

Today I worked on my dissertation more, writing the first section of chapter 4, wherein I discuss the implementation details of the web crawler. The second part is the unit tests, which as I look back on, appear somewhat lacking. I will need to implement further tests in order to have enough to discuss during this part.

Total word count so far is approximately 10,000 words. I suspect I may be writing a bit too long-windedly, so I'm on the lookout for places to cut words out.

## Sunday 2<sup>nd</sup> April

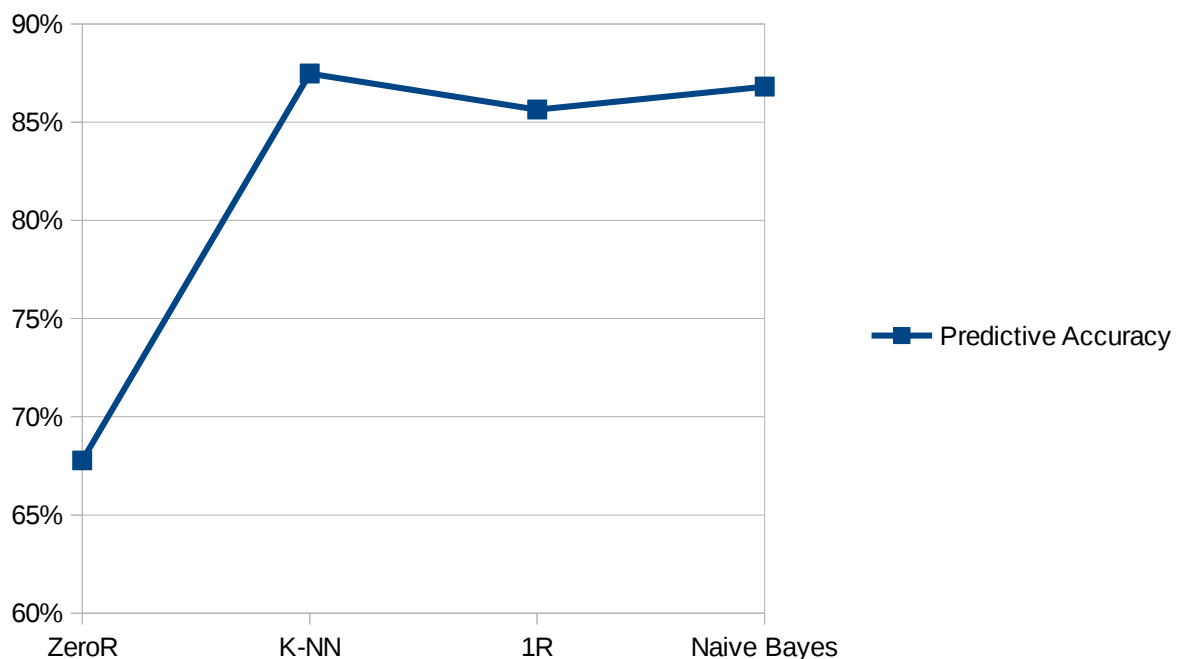
I want to get the dissertation as close to finished as I can by the 12<sup>th</sup>, as that is the deadline for feedback from Dr Bryant. As such, I will be concentrating on it for the next few days. Today I wrote another 800 words, finishing off the section on unit testing the web crawler and starting the chapter on data mining. I have decided that the experiments will be interleaved with the preliminary and later data mining, so as to integrate them more closely with the chapter as a whole, rather than shunt them off into their own section.

## Monday 3<sup>rd</sup> April

I have decided how to lay out the experiments. I will only consider the final, merged dataset, and build experiments from there. I will be laying out the results from the experiments as a graph, created from a table of results in LibreOffice. As such, here is the table of results for the preliminary mining:

Algorithm	Predictive Accuracy
ZeroR	67.7796 %
K-NN	87.4791 %
1R	85.6427 %
Naive Bayes	86.8114 %

And the graph, generated from this data:



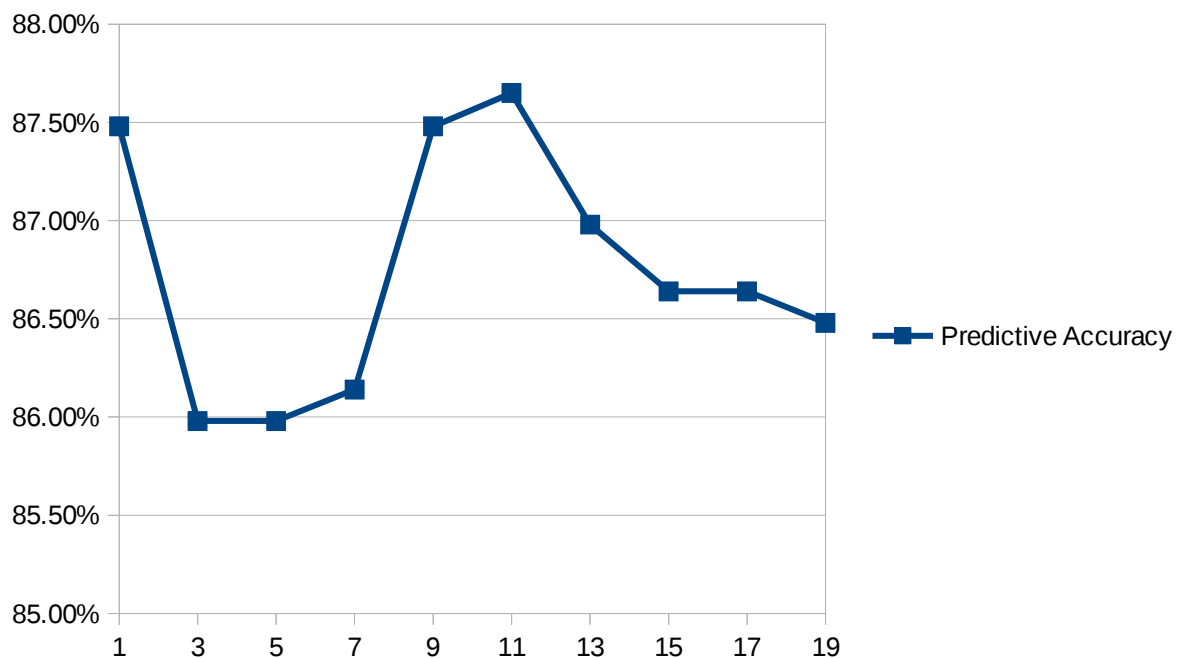
The axis are not labeled, but the line of results is. Hopefully this will meet Dr Bryant's standards.

Notice that the scale of the Y axis needed editing – this is because the default starts at 0%, which made the subtle differences between 1R, KNN and Naive Bayes difficult to spot.

The next experiment is on the various values of K on the combined dataset. These were not done on this dataset originally, therefore need to be done now.

K	Predictive Accuracy
1	87.48%
3	85.98%
5	85.98%
7	86.14%
9	87.48%
11	87.65%
13	86.98%
15	86.64%
17	86.64%
19	86.48%

And the graph generated from these results is below.



Thirdly, a graph containing all the algorithms and their default accuracy values is required.



Algorithm	Predictive Accuracy
ZeroR	67.7796 %
Random Forest	90.1503 %
K-NN	87.6591 %
1R	85.6427 %
Naive Bayes	86.8114 %
Bayesian Network	87.813 %
SVM	87.4791 %
J48	89.8164 %

### **Thursday 6<sup>th</sup> April**

Today I finished off Chapter 5 of my dissertation, and made a start on chapter 6, which covers the implementation of the user interface, and how this is integrated with the rest of the system. Today was a much more productive day than I had hoped, pushing me forward to 13,000 words. This gives me hope that I will not have to send an incomplete draft of my dissertation to Dr Bryant, which will give me the best feedback opportunities possible.

I will continue my dissertation tomorrow, and aim to finish chapter 6. Maybe I can even make significant headway on chapter 7 – I can finish chapter 6 in 500 words, and 1,500 words for the critical evaluation and conclusion seems like a good target to set.

### **Friday 7<sup>th</sup> April**

Today I finished chapter 6 of my dissertation with talking about how the subsystems integrate with each other, and how the full system was tested.

Once that was done, I started chapter 7, concluding the dissertation with firstly a critical evaluation of the project itself, then an evaluation of the project's wider context, followed by considering other legal, professional, ethical and social issues. The only professional issue I was able to think of was the issue of not using concurrency in the data gathering subsystem, which is only an issue in a multi-user system which it is not in its current state.

In terms of social issues, there is the fact that my opinion is subjective and yet informs so much of the project. This could lead to genuinely benevolent websites being misclassified as phishing, which could hurt their reputation. But this again depends on the system becoming much more popular than it is ever likely to be.

### **Tuesday 18<sup>th</sup> April**

I sent my dissertation draft to Dr Bryant in the interim between this entry and the last one, received some feedback, and am going to correct my dissertation in light of this feedback. I need to correct the abstract to include the findings of the project, and how effective the application is at detecting

phishing sites. Then, I need to correct the first chapter – there are many disparate things to fix, so I shall not be going through them all in detail.

One of the things I need to do is to change the graphs, so that they are bar graphs instead of line graphs. Fortunately, this is very simple to do.

I also need to merge the reference lists – this lead to slight confusion, since the appendices each have their own individual references, so does the combined list go before or after the appendices? In the end I put it before them, which was mostly an arbitrary choice.

## **Appendix B. The project plan**

### **FINAL YEAR PROJECT PLAN**

PROJECT TITLE: Project to distinguish phishing websites from benevolent ones

AUTHOR: Sophie Brown

SUPERVISOR OF PROJECT: Dr C. H. Bryant

DATE: 19/10/2016

#### **INTRODUCTION:**

The aim of this project is to design, develop and implement a data mining application used for determining if a website is a phishing site, using a mixture of data gathered by a web crawler and data provided by a human, and to investigate whether the selected attributes are good indicators of whether a given website is a phishing site.

#### **BACKGROUND:**

“Phishing is the attempt to obtain sensitive information such as usernames, passwords and credit card details (and sometimes, indirectly, money) often for malicious reasons, by masquerading as a trustworthy entity in an electronic communication.” (Wikipedia, 2016)

Attributes of phishing websites that will be used to form a dataset in this project are as follows:

- They often use URLs that are similar to, but not quite, dictionary words or trusted website names. Similarity will be measured using Levenshtein distance (Levenshtein, cited in Wikipedia 2016)
- Pop-ups may appear, claiming that the user has won a prize, or needs to claim something. The number of popups will be recorded as an attribute.
- If the user is trying to get to a different site, and that site instead redirects the user to a different site, that site is likely to be a phishing site. This redirect can be recorded and used as an attribute.
- The actual site is likely to emulate an existing, trusted website. This could be recorded as “identical”, “very similar”, “similar”, “different”, “very different” and “totally different”.
- The website is likely to offer some kind of reward in exchange for personal information, such as a cash payout, or peace of mind – as an example, consider phishing scams that pretend to be your bank, and require your details in order to "prevent fraud".

#### **OBJECTIVES:**

- To write a web crawler capable of recording the URL similarity, number of popups and whether a redirection was attempted.
- To write unit tests to measure the performance of the crawler on test websites

- To write a user interface consisting of a form that allows a user to submit a URL, similarity to an existing website, and whether or not the website in question offers something in exchange for personal information.
- To link the user interface to a back end, consisting of the crawler and an algorithm implemented within WEKA, such that the crawler will gather the remaining attributes and the algorithm will run on the resulting data.
- To configure the crawler to gather an approximate 1:5 ratio of phishing and non-phishing sites.
- To provide the human attributes in at least 90% of the training data.
- To evaluate algorithms selected for this type of dataset, and select one to generate a model.
- To estimate the predictive performance of this model.

## METHODOLOGY

Since my project contains a mixture of software engineering elements and data mining elements, I will blend together a software engineering approach with a data mining one. Since I do not have a full and complete understanding of what the crawler and application will look like when complete, an incremental software development approach is feasible, wherein I will build parts of the software, make sure that part is working, then move on to the next. Since I will need to build two small and simple pieces of software in the course of my project, I will employ a scaled-down version of the SCRUM methodology (Scrumalliance.org, 2016), featuring sprints twice a week.

This method will be combined with the CRISP-DM methodology (Smart Vision - Europe, 2016) for data mining. In particular, I will loosely follow the "data preparation" through to "deployment" phases of this methodology, because I already have a relatively solid understanding of what my data is going to look like, the ratio between the two class values, and overall size of the dataset, as well as the problem domain and what my project will end up being used for, therefore the "Business understanding" and "data understanding" phases are less necessary.

## RESOURCES

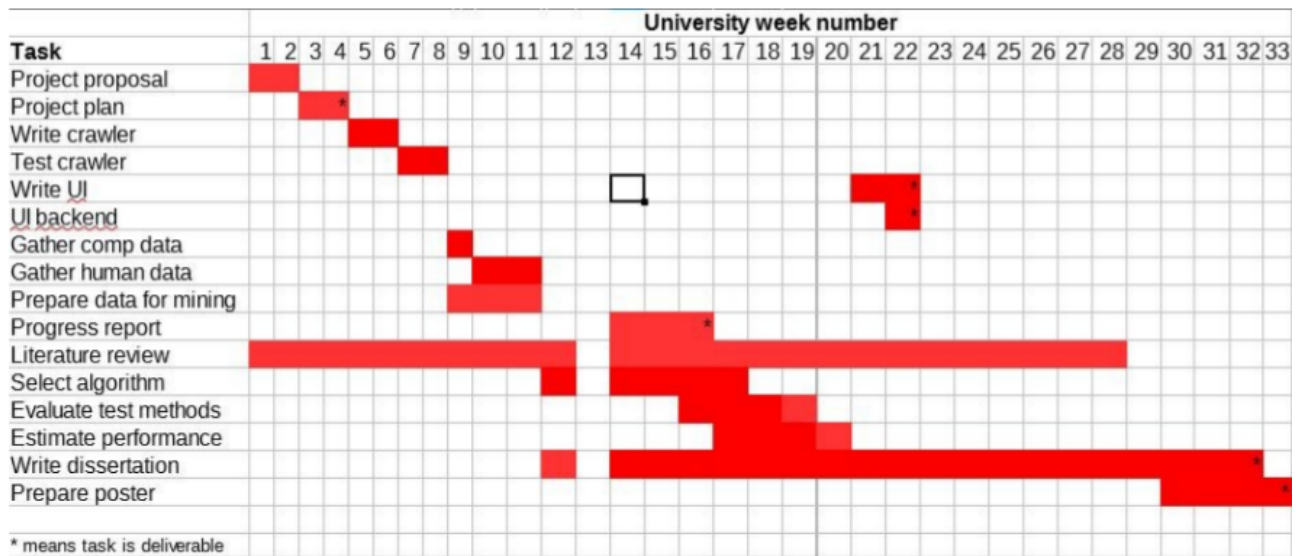
This project requires the following resources:

- A personal computer, with the following software installed:
  1. WEKA, a data mining tool written in Java. (Witten et al., 2005) I will use WEKA's explorer interface for the training and test phases of my project, and the Java implementations of its algorithms when it comes to writing the application for deployment
  2. Java, for writing the web crawler that will gather the training data and the application for future use.
  3. Eclipse, a fully featured Java IDE (Integrated Development Environment). I will develop the crawler and deployment application using Eclipse.
  4. VirtualBox. The web crawler will deliberately trawl the web looking for dangerous websites, and therefore needs to be run in an environment where the risk of viruses causing damage to the computer is minimal. Using a virtual machine means that if the crawler does give me a virus, that virus's effects are constrained to the guest OS running within VirtualBox, which can be wiped and re-installed from scratch with no loss to the host OS.

- Books, journals, papers, websites on the characteristics of successful data mining projects and how this success was achieved, so that I can study previous projects in this field for

further information on how mine can be a success.

## TIME PLAN:



## Final Year Project for CS

### Progress report

#### Project to distinguish phishing websites from benevolent ones

By Sophie Brown

#### **Abstract**

This project uses Data Mining methods to build a classification model of whether or not a given website is a phishing site or not. Training data is gathered using a web crawler written in Java, and a model is generated using WEKA. Later, a Java front end will be developed to allow users to specify a URL, which will then apply this model to classify the website as phishing or non-phishing, as well as give a measure of confidence in the prediction.

#### **Literature review**

When this project was first conceived, it was with the intent of learning more about how data mining is used for real-world projects. Hence, the first step was to look for previous data mining projects for inspiration. In the end, the project that was approved was inspired by (Nevonprojects.com, 2015a), with (Nevonprojects.com, 2015b) providing the idea to include a front-end to allow users to specify their own websites.

Once the project was proposed and approved, the next stage was to think about what kinds of technology would be used to achieve it. Data gathering would be necessary on a relatively large scale, so a web crawler would be used to fetch and process data about websites quickly and efficiently. When dealing with websites, it is common to use a server-side web language such as PHP. However, the crawler also needs to be connected to the front-end user interface, as well as a data mining package that will create a model for prediction. Hence, for ease of integrating these separate parts together, they should all be written in the same language.

During my industrial placement, I worked with a Java framework called GWT, which allows web applications to be written entirely in Java. (Gwtproject.org, n.d.) Hence, either a desktop or a web application can be written in the same language, and changing between the two requires minimal effort. Therefore, Java was a good language to write the front end in, since the choice of web or desktop application was not yet finalized.

When considering the data mining software, it would be folly to reinvent the wheel when software packages already exist to do it for us. Therefore, the objective is not to write data mining software but rather to use existing software in the context of detecting phishing websites. R is a programming language designed for “statistical computing and graphics”, (R-project.org, 2016) and therefore a data mining package built in R would appear to be a fine choice.

However, R does not offer anywhere near the support for UI building that more general-purpose languages such as C++ and Java offer – only one framework, Shiny, appears to offer the capability.

As such, and remembering that ideally the same language should be used for as much of the programming as possible, a data mining package written in Java would be the best thing to search for. This exists in the form of WEKA (Witten et al, 2011), which consists of a collection of data mining algorithms and methods for applying these algorithms on datasets. As such, WEKA meets all of the needs for the project, with no need to write anything from scratch.

The selection of Java packages for the data mining and front end subsystems limits what can be used for the web crawler. As such, crawler4j (GitHub, 2015) was chosen, by dint of being the only standalone java crawler with community support that was found. Other Java web crawlers include Nutch (Nutch.apache.org, 2009), which relies on other packages and is therefore overly complex for this project, and Heritrix (Webarchive.jira.com, 2010), which was updated only once in 7 years.

Furthermore, manually visiting potentially harmful sites will be necessary, and therefore security needs to be addressed. However, effective security could potentially focus on preventing risks before they are noticable, and therefore are unsuitable for this project. As such, all visits to nefarious sites were carried out within a virtual machine in order to contain any damaging effects. The two major packages are largely equivalent in terms of sandboxing, as they are being used here, and so the choice of virtualization software was primarily made on ethical grounds – VirtualBox (Virtualbox.org, 2007) is open source, whereas its primary competitor, VMWare Player (Vmware.com, 1999) is closed source. Therefore, VirtualBox was the VM software of choice for this project.

This project deals with a widely studied problem in data mining (He et al., 2011) (Hadi et al, 2016) (Xiang et al., 2011) and many different approaches have been developed to tackle it. This project constitutes an attempt to identify modern phishing websites, using a mixture of attributes gathered using automated methods, and attributes gathered by a human user. Studies have been carried out using features of the website itself to detect phishing sites (Xiang et al., 2011), and detection of phishing URLs is also a topic of some research (Basnet and Doleck, 2015). This project attempts to combine the two approaches, and use traditional data mining methods to detect phishing sites.

## **Aim and Objectives review**

The aim of this project is to “design, develop and implement a data mining application used for determining if a website is a phishing site, using a mixture of data gathered by a web crawler and data provided by a human, and to investigate whether the selected attributes are good indicators of whether a given website is a phishing site”. This effectively shows a double aim, considering the data mining application and the attribute evaluation as separate but closely related parts. In effect, the aim of this project is not just to deliver an application that can predict the likelihood of phishing sites, but to deliver one that can consistently make accurate predictions.

When planning the project, an important factor is the methodology that will be used to proceed through the plan. As this project combines software development with data mining, elements of both data mining and software engineering methodologies will need to be used. When developing the software, a scaled-down variant of scrum (scrumalliance.org, 2016) has been used, in which sprints occur in a matter of 2 to 4 days, instead of the weeks that enterprise implementation of scrum typically implements.

By contrast, the data mining section of the project requires an appropriate data mining methodology. A common methodology is CRISP-DM (Smart Vision - Europe, 2016), and this project implements a subset of it. Since there has been an understanding of the purpose of the data from the start of the

project, the remaining phases are all that is necessary to constitute the methodology for the data mining elements of the project.

The project objectives have been translated into the project plan, with each task on the chart being a single objective. This means that the progress through the project in the best-case scenario is a linear path down the list of objectives. Figure 1 shows a table of each objective, along with its status as complete, in progress or not started.

Objective	Status
Write a web crawler capable of recording the URL similarity, number of popups and whether a redirection was attempted.	Complete
Write unit tests to measure the performance of the crawler on test websites.	Complete
Write a user interface consisting of a form that allows a user to submit a URL, similarity to an existing website, and whether or not the website in question offers something in exchange for personal information.	Not started
Link the user interface to a back end, consisting of the crawler and an algorithm implemented within WEKA, such that the crawler will gather the remaining attributes and the algorithm will run on the resulting data.	Not started
Configure the crawler to gather an approximate 1:5 ratio of phishing and non-phishing sites.	Complete
Provide the human attributes in at least 90% of the training data.	Complete
Evaluate algorithms selected for this type of dataset, and select one to generate a model.	In Progress
Estimate the predictive performance of this model.	In progress

Figure 1. status of project objectives

As presented here, progress through the project has not been very linear at all. However, when presented in tandem with the gantt chart, it can be seen that progress through the project adheres reasonably closely to the plan.

With regards to how the objectives have evolved over the course of the project, the biggest difference is in the data gathering. Instead of a single dataset with a particular ratio of benevolent to phishing, three datasets of differing ratios were gathered, with a 1:1, 3:1 and 6:1 ratio. This was done in order to ensure that high predictive accuracy obtained using just one dataset was not simply a fluke, and to provide additional data points in order to gain a greater level of confidence in the experimental model, which is discussed in the next section.



## Progress Evaluation

The web crawler gathers certain data on aspects of websites and prints it to an ARFF file (Cs.waikato.ac.nz, 2008). The data gathered was initially the URL similarity to a known list of benevolent URLs, whether or not the website caused a redirect, and the number of popups that the website generates. However, counting the number of popups turned out to be a computationally difficult problem, as popups are loaded in the client and the crawler operates on the server, and was replaced with counting the number of spelling errors in the main body of the site.

Once the crawler was written, the next step was to write unit testing for it, to be assured that the crawler gathers the right data from the websites it visits. Four tests were written – one for testing the gathering of each attribute, and one to run the crawler through the process of gathering data from a single site. Once all of these tests passed, the process of using the crawler to gather real data was begun.

A web crawler operates by designating a “seed” site, that is then visited. Any hyperlinks are then followed, and the sites they point to are visited, and the process repeats. Therefore, the choice of seed is important in that it effectively chooses the kind of website the crawler gets its data from. As such, multiple datasets needed to be compiled to show the effects on the results caused by crawling from, for example, academic websites or webcomics or commercial sites.

An unexpected pitfall in this approach is that, since legitimate websites are unlikely to link to phishing websites, a crawl is ineffective for gathering phishing data. SPAM archive (Untroubled.org, 1998) is a collection of spam emails sent to a honeypot address from 1998 to 2017. Emails were downloaded from this archive, and a BASH script was written to extract URLs from these files.

```
#!/bin/bash

for file in spamUrls/*; do

    url=$(grep -Eo 'https?:/[^\"]>]+' "$file")
    echo $file + ": " + $url
    echo $url >> results.txt
done

sed -i 's/\s\+/\n/g' results.txt
sed -i 's/<\./.*//' results.txt
```

Figure 2. the BASH script written to parse URLs from spam emails

Once a list of URLs is gathered, the crawler can then be set to visit each site in the list, providing phishing data.

Finally, once the datasets were compiled and the human attributes – how similar the site looks to any known site, and whether the site makes any offers to the user – are added to them, several algorithms were run on the data using WEKA. The predictive accuracies of these results, in appendix 2, provide a baseline for choosing the algorithm that will be used in the deployment phase. Any algorithm chosen must logically provide consistently higher results than the algorithms used here, which sets a high bar, appropriate for achieving the second part of the project aim.

## Timeplan revision

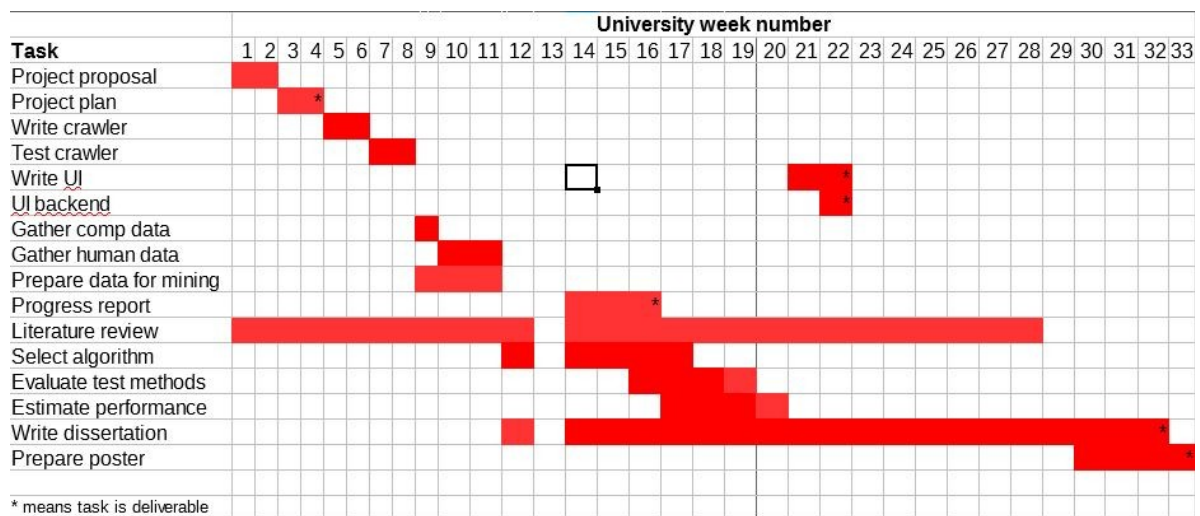


Figure 3. the initial timeplan for the project

The time plan as presented at the start of the project took into account potential schedule slips, and a planned holiday of one week. The downside to this is that there were some tasks that really should have been performed sequentially were planned as if they were in parallel. Notably, gathering and preparing data should have taken up different time slots, as the preparation depends on having some data gathered.

At the time of writing this report, it is week 17. As a result of doing preliminary data mining, the tasks that deal exclusively with that area have been lengthened, as they are stretched throughout a longer time span. Research has been done in search of an algorithm to use during the deployment phase, with (Akanbi et al, 2014) indicating that K-NN is a good choice for small datasets, and the preliminary results obtained from this project support this.

However, due to the need to confirm that K-NN is the best choice, this and related tasks cannot end early, so the project should proceed on schedule. No progress has yet been made on the dissertation, so that has been revised accordingly. The resultant time plan is one that has seen small revisions, but is fundamentally the same.

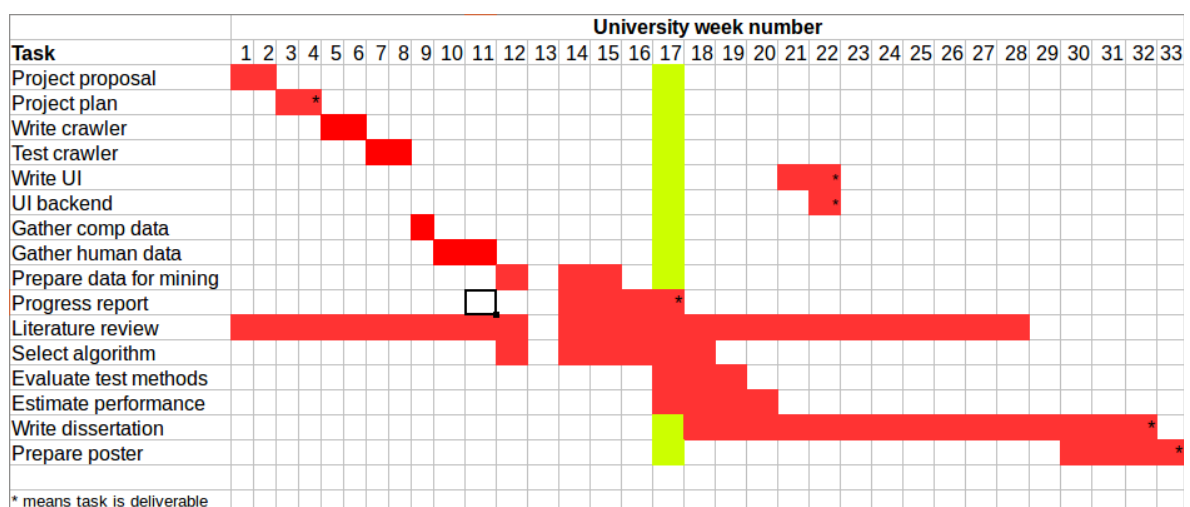


Figure 4. the updated timeplan, showing the current week

## **Appendix D. The ARFF dataset headers**

@RELATION websites

@ATTRIBUTE urlsimilarity NUMERIC

@ATTRIBUTE redirection {true,false}

@ATTRIBUTE spellingErrors NUMERIC

@ATTRIBUTE lookandfeel

{identical,veryclose,similar,different,unique}

@ATTRIBUTE offers {yes,no}

@ATTRIBUTE phishing {yes,no}