Development of a New User Interface for Weka, a Collection of Data Mining Algorithms in Java

Andrew Jones
BSc (Hons) Computer Science
University of Bath
2006
DEVELOPMENT OF A NEW USER INTERFACE FOR WEKA, A COLLECTION OF DATA MINING ALGORITHMS IN JAVA

Submitted by Andrew Jones

COPYRIGHT

Attention is drawn to the fact that the copyright of this dissertation rests with its author. The Intellectual Property Rights of the products produced as part of the project belong to the University of Bath (see http://www.bath.ac.uk/ordinances/#intelprop).

This copy of the dissertation has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the dissertation and no information derived from it may be published without the prior written consent of the author.

DECLARATION

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of this work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university of institution of learning. Except where specifically acknowledged, it is the work of the author.

Signed: ..........................................................
ABSTRACT

Data mining is a process that involves a number of techniques designed to analyse large data sets, locate patterns within them, and then use this information to predict future trends and behaviours (Thearling, *No Year*). Weka is an open source Java software package designed to perform the task of data mining. It is made up of a collection of machine learning algorithms to perform the data mining processes, and provides a graphical user interface to allow the user to access and execute them. The Weka user interface is highly unintuitive and suffers from many usability issues. In particular, it has a very steep learning curve for new users. This project aims to develop a new user interface for the Weka software that is more user-friendly and does not suffer from the usability issues found in the current interface. The development process will be discussed in detail, followed by an analysis of whether the interface achieves its aims.
ACKNOWLEDGEMENTS

I would like to thank my project supervisor Dr. Alwyn Barry for his assistance and encouragement throughout the project. I would also like to thank my friends and family for their continued support and help during the interface evaluations. Finally I would like to thank the Weka user community for providing feedback during the evaluation stages of the project.
Contents

1. Introduction............................................................................................................. 1
   1.1 Problem Description ........................................................................................... 1
   1.2 Project Aims........................................................................................................ 2
   1.3 Project Outline ................................................................................................... 2

2. Data Mining............................................................................................................. 4
   2.1 Introduction.......................................................................................................... 4
   2.2 What is data mining? ......................................................................................... 4
   2.3 Supervised Learning Techniques ....................................................................... 6
      2.3.1 Linear regression.......................................................................................... 6
      2.3.2 Nearest neighbour ....................................................................................... 6
      2.3.3 Decision trees.............................................................................................. 7
      2.3.4 Neural networks........................................................................................... 8
   2.4 Unsupervised Learning Techniques ................................................................... 8
      2.4.1 Clustering..................................................................................................... 8
      2.4.2 Association rules......................................................................................... 9
   2.5 Other Data Mining Concepts ............................................................................. 10
      2.5.1 Input and output of data mining techniques............................................... 10
      2.5.2 Preprocessing ............................................................................................. 10
      2.5.3 Estimating the accuracy of a prediction model........................................... 11
      2.5.4 Visualizing data mining results................................................................. 12
   2.6 Conclusions....................................................................................................... 12

3. User Interface Design............................................................................................. 13
   3.1 Identifying and Understanding the Users......................................................... 13
      3.1.1 Why are users important? .......................................................................... 13
      3.1.2 Who are the users in this project? ............................................................... 13
   3.2 Usability............................................................................................................... 13
      3.2.1 What is usability?....................................................................................... 13
   3.3 Design Process ................................................................................................... 14
      3.3.1 Identify users and tasks............................................................................... 14
      3.3.2 Use existing design ideas............................................................................ 14
   3.4 Evaluating user interfaces.................................................................................. 15
      3.4.1 Evaluation by users.................................................................................... 15
      3.4.2 Evaluation without users............................................................................ 15
   3.5 Conclusions....................................................................................................... 16
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>17</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Explorer</td>
<td>17</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Experimenter</td>
<td>18</td>
</tr>
<tr>
<td>4.3.1</td>
<td>The process</td>
<td>19</td>
</tr>
<tr>
<td>4.3.2</td>
<td>The results</td>
<td>19</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Techniques</td>
<td>20</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Selection of tasks</td>
<td>20</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Cognitive walkthrough results</td>
<td>22</td>
</tr>
<tr>
<td>4.4.4</td>
<td>Heuristic evaluation results</td>
<td>24</td>
</tr>
<tr>
<td>4.5</td>
<td>Conclusions</td>
<td>27</td>
</tr>
<tr>
<td>5.1</td>
<td>Introduction</td>
<td>28</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Development of use cases</td>
<td>29</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Use case analysis</td>
<td>30</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Functional requirements</td>
<td>30</td>
</tr>
<tr>
<td>5.5</td>
<td>Conclusions</td>
<td>31</td>
</tr>
<tr>
<td>6.1</td>
<td>Introduction</td>
<td>32</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Overall Design</td>
<td>33</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Main Screen</td>
<td>34</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Menus</td>
<td>37</td>
</tr>
<tr>
<td>6.2.4</td>
<td>Wizard</td>
<td>38</td>
</tr>
<tr>
<td>6.2.5</td>
<td>Experiments</td>
<td>39</td>
</tr>
<tr>
<td>6.3</td>
<td>Conclusions</td>
<td>40</td>
</tr>
<tr>
<td>7.1</td>
<td>Introduction</td>
<td>41</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Tools used</td>
<td>41</td>
</tr>
</tbody>
</table>
1. Introduction

1.1 Problem Description

Weka is an open source Java software package designed to perform the task of data mining. It is made up of a collection of machine learning algorithms to perform the data mining processes, and provides a graphical user interface (GUI) to allow the user to access and execute them.

Data mining is a process that involves a number of techniques designed to analyse large data sets, locate patterns within them, and then use this information to predict future trends and behaviours (Thearling, No Year). With technological advancements, we are able to store more and more data all the time, and so this is a field of research that is becoming increasingly important.

Weka contains tools for the key data mining processes of data pre-processing, classification, regression, clustering, association rules and visualisation (Witten and Frank, 2005). It allows the user to select a data set and then apply various algorithms to it to analyse the data. Weka offers the user access to many different data mining algorithms as well as allowing them to implement their own into the system, and so offers a great deal of functionality.

However, one of the potential downfalls of the program is the fact that the user interface provided is far from intuitive. For new users especially, it has a very steep learning curve, as it does not use the conventions common to many computer applications that most users will be used to.

The existing interface is split into four completely separate sections:

- The simple command line interface allows a more experienced user to input commands directly into a command line.
- The Explorer interface is a graphical interface that allows the user to perform the main data mining tasks and to visualise the results in a number of forms.
- The Experimenter interface allows the user to compare and analyse the performance of different data mining algorithms.
- The Knowledge Flow interface is offered as an alternative to the Explorer interface but does not contain all the functionality offered by the Explorer. It is based around a data flow style of layout.

Both the appearance and interaction methods of each of the four interfaces are significantly different, and this adds to the complexity of the overall design. The system may seem far more intuitive if there was more consistency through the different functions offered by each of the sections. Whether each of these four separate interfaces are needed, or whether it is feasible for some or all of them to be merged into one more standardised design is a question that will need further investigation.

One of the areas that particularly suffers is the application of the various algorithms to the data. Algorithms are simply selected from a list, and very little description of what they do is provided.
As the software is released as open source code, it is possible for users to write their own algorithms or additions to the system. Using the current interface, there is no simple way for these additions to be added and applied to a data set. If they could be added and applied using an intuitive “plug-in” feature, it would enhance this potentially powerful functionality greatly.

Weka is a program with a great deal of functionality that could prove very useful for a wide range of users. However, using the system can be both confusing and frustrating, thanks to a highly unintuitive user interface.

1.2 Project Aims

The aim of this project is to design and implement a new front end for the software. The new interface will need to be more user-friendly while still offering all the functionality that is available through the program. The design process will need to include the application of user interface design standards to ensure that the interface is as effective as possible. It will be important to ensure that the interface makes the software far more accessible for new users, whilst not making it frustrating for more experienced users to continue using.

Weka supports a strong user community, and having developed a suitable new interface, I hope to include them in the evaluation of the design. Using the feedback I receive, I will then aim to enhance the interface further based on their opinions and ideas.

1.3 Project Outline

We will begin with a discussion of the background topics that will be key to the success of the project. In Chapter 2, the fundamentals of data mining will be covered, including a discussion of the main processes and common techniques used. Understanding these will be vital to ensuring the new interface allows the user to perform their desired tasks in a natural and intuitive way.

Chapter 3 will cover the background work completed on the broad topic of user interface design. This is a field of study that has been the subject of a great deal of research, but, due to its nature, is one where conclusive results can be difficult to achieve. Taking advantage of the work previously performed in this area will be a key factor in ensuring the new interface is as user-friendly as possible.

To ensure the new interface eliminates as many of the problems from the existing interface as possible, a full evaluation of the current interface will be carried out. This is covered in Chapter 4.

Chapter 5 focuses on the requirements gathering process, a task that will play an important role in the final design of the new interface.

A thorough examination of the design process will be discussed in Chapter 6, including the different options considered in creating the final design, and the reasons for the design decisions made.
Chapter 7 will cover the implementation stage of the project, discussing the issues encountered during this process, and the reasons behind the implementation decisions that were made.

The new interface will need to be fully evaluated to analyse whether it achieves its aims, and to identify possible improvements. The techniques used, results, and improvements made will be discussed in Chapter 8.

Finally, an evaluation of the project as a whole will be given in Chapter 9. Here we will discuss the conclusions drawn from the project, an overall performance evaluation and a discussion of potential future improvements to the interface.
2. Data Mining

To successfully design and implement a new interface for Weka, it is vital the all the functions offered by the software are first understood. Therefore, a full background knowledge on data mining (also known as knowledge discovery) and the techniques it involves will be required to successfully achieve the goals of the project.

Although a great deal of literature is available on the subject of data mining, it is often not written in particularly “simple” terms. One of the potential challenges in compiling such a summary is first becoming familiar with the terminology used (which tends to vary with different authors), and then explaining the potentially complicated topics in a coherent and most importantly understandable manner.

2.1 Introduction

Advances in technology have lead to dramatic increases in the amount of information we are able to store. Databases are larger and more complex than ever before, and contain far more data than can analysed by a human. A data set is made up of many records or instances and each has a number of attributes, also known as features or variables (Becher et al. 2000).

Making sense of so much potentially useful information is a significant problem, and is one that has led to the development of the area of research known as knowledge discovery in databases (KDD). Fayyad et al. (1996) defines this process as follows:

Knowledge Discovery in Databases is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data.

The initial steps of the knowledge discovery process include selection of the most relevant attributes, and performing preprocessing techniques on the data to ensure it does not contain noisy data and is in the correct format to be used as input into an algorithm. The data mining step of the process involves the application of data analysis and discovery algorithms to the data. The final step involves an analysis or evaluation of the results of the applied algorithm to interpret the information that has been extracted.

Weka includes functionality to apply each of these steps to users’ data sets; however, the main focus of the software is on the process of data mining.

2.2 What is data mining?

Thearling (No Year) defines data mining as “the extraction of hidden predictive information from large databases”. A key point about these methods is that they allow the extraction of previously unknown and potentially useful information from data in databases (Chen et al. 1996); in other words, we are able to find out things from the data we have that we did not know before, and could not have been found by simple inspection of the data.
The process of data mining has two basic aims:

- To recognise patterns and trends in data.
- To make predictions based on patterns and trends found.

The ability to predict future events based on past data is one of the factors that makes data mining such a powerful tool. Companies who have large collections of data about how their customers have behaved in the past will be extremely interested in how this data can predict what potential customers are likely to do in the future.

Another important factor that many data mining techniques offer is a measure of the confidence in the accuracy of the predictions given. Some predictions can be made with almost complete confidence in their results, whilst others will have values predicted that only have a small chance of actually occurring. This can be largely dependent on the nature of the data and how strong the trends and patterns in it are. Companies will be far less willing to invest money in predictions with a low confidence level.

Many different data mining techniques have been developed, and in general these can be grouped into two categories: supervised and unsupervised.

Which of these learning techniques should be used depends on the type of information that is to be analysed, and the desired outcome from the analysis. Hastie et al. (2001) gives a concise description of the goals of each of the learning techniques:

In supervised learning, we have an output measurement that we wish to predict based on a set of features. We have a training set of data, in which we observe the outcome and feature measurements for a set of objects. Using this data we build a prediction model, or learner, which will enable us to predict the outcome for new unseen objects. A good learner is one that accurately predicts such an outcome.

In unsupervised learning, we observe only features and have no measurements of the outcome. Our task is rather to describe how the data are organized or clustered.

Many data mining techniques are able to “learn” over time by refining the algorithm used to create more and more accurate predictions. This is known as machine learning. Nilsson (1996) gives the following description of machine learning:

A machine learns whenever it changes its structure, program, or data (based on its inputs or in response to external information) in such a manner that its expected future performance improves.

A feature that is used in the generation of predictions is known as a predictor.

We will now discuss the most common data mining techniques, including their basic principles, how effective they are, and how they compare to other methods.
2.3 Supervised Learning Techniques

2.3.1 Linear regression

Linear regression is one of the simplest forms of data mining. A single feature of the data set is mapped against each of the related prediction values in a two dimensional space. The line that minimizes the error between the actual values and their corresponding point on the line can then act as a boundary to predict output values of future data (Berson et al. 1999). Confidence in the accuracy of a result could be measured by its distance from the boundary line.

The most commonly used method to find the best position of this line is to find the minimum sum of the squares between the actual values and the predicted values. This method effectively finds the line that has will result in the least error for the training data.

Hastie et al. (2001) tells us that “a linear decision boundary is unlikely to be optimal, and in fact is not”. Almost all data sets will be better represented by a non-linear decision boundary. This is an outcome that is far more difficult to achieve. One possible way to attempt to create a more accurate prediction model is to include more than one predictor (and hence more than one dimension for the model). This will result in a more complex and hopefully more accurate prediction model.

However, as Clyde et al. (1996) tells us, the selection of predictors to be used in a model is “typically difficult, and also crucial in influencing predictions”. There will often be a large selection of predictors to choose from, and transformations to combine predictors are often considered as possible choices as well. Making the choice of the most appropriate predictors, and those that will give the most accurate predictions therefore becomes a non-trivial task.

2.3.2 Nearest neighbour

The Nearest Neighbour technique makes predictions for new data by looking at the output value of the record that is “nearest” to it. Luisa et al. (1996) tells us that:

In this method, the sample is classified in the same class as the prototype which minimizes a predefined distance function with respect to the sample.

This is another relatively simple technique but will again not be especially accurate in many situations. For example, any anomalous results could easily affect the results of predictions.

An improvement on this technique is the K-nearest neighbour method. Rather than simply taking the value of the one nearest object, the average (or most popular in categorical data) of the nearest ‘K’ values will be taken. The value chosen for K needs to be picked carefully, as taking too many will result in too general an average, while taking too low a value will not be a great improvement on the single nearest neighbour algorithm.
Berson et al. (1999) tells us that the confidence in predictions made by nearest neighbour methods can be assessed by the distance of the neighbour(s) from the object. We can have more confidence in neighbours that are closer together. In the K-nearest neighbour method, the more of the nearest neighbours that give the same result, the more confident we can be in the prediction. Nearest neighbour methods are most effective with data where features are statistically independent (Singh et al. 1999).

In techniques such as nearest neighbour and clustering, where data is assessed based on how “near” it is to other data, certain dimensions can be “stretched” to increase the importance of a particular predictor. The distance between objects will therefore be increased in this dimension and so the weighting given to them will also increase (Berson et al. 1999).

2.3.3 Decision trees

Quinlan (1990) tells us that decision trees “employ a top-down, divide-and-conquer strategy that partitions the given set of objects into smaller and smaller subsets in step with the growth of the tree”. The aim of a decision tree is to classify the data based on decisions made at each level of the tree-structured model created. At each level of the tree, a test is run on the data item, and the data will be split between the branches based on the result of the question. An example of a test may be whether or not the data item is greater than a particular value. When a data item reaches a leaf of the tree it is placed in the class of data items corresponding to that leaf.

The growth of a tree model provides a model that is easy for the user to understand, as they are able to see how the data is classified by examining the questions that are asked of the data.

Decision trees will often suffer from the problem of overfitting. This occurs when the data continues to be divided after the optimum set of results has been achieved, and too many different classes are generated. If a particular sub-tree in the prediction model constructed does not improve the classification error rate it becomes unnecessary and may be removed using a process known as pruning (Fournier and Crémilleux 2002).

A significant problem in the generation of decision trees is ensuring the questions asked at each level split the data in a useful manner. Quinlan (1990) identifies a number of heuristics that can potentially be used to decide which tests to use the data, based on the following criteria:

- Information – tests are ranked based on apparent information they provide regarding which classifications members of the training set are assigned.
- Error – tests are ranked on the level of error in the training data if the data was classified at this point.
- Statistical significance – tests are chosen based on how relevant the attribute is to the overall data set.

Choosing the method that will provide the most accurate and useful results can be difficult. One solution is to build all possible trees and then select the best one to keep. However, this can be very computationally expensive.
2.3.4 Neural networks

According to Hastie *et al.* (2001), “[a] neural network is a two-stage regression or classification model, typically represented by a network diagram”. Neural networks are made up of nodes connected together with links. All data in a neural network must be numerical.

Data is passed into the input nodes (one node for each predictor) of the neural network and then follows the link onto the next node. Each link has a numerical weighting, and when values are passed along a link they are multiplied by the weighting of the link. At the next node they are then added to any other values that are passed into the node. At the final node an output value is calculated and a prediction is made based on this value. Intermediate nodes are known as hidden nodes as they cannot be seen by the user (Berson *et al.* 1999). The values on these nodes have no real meaning to the user; they are simply constants in the polynomial used by the network to calculate the output value.

To train the network, data is passed through it, allowing it to “learn” and therefore predict more accurate output values. As the training data is passed through, its predicted output is compared to its actual output. The weightings on the links are then updated depending on the result, making the overall process more accurate.

Berson *et al.* (1999) tells us that while neural networks often give very accurate results, they do not offer any explanation of why the data is behaving in the way it does. The network effectively boils down to a complex mathematical equation and is not much use to a user who wishes to know why data is behaving in a certain way rather than just the prediction values assigned to it. This is in contrast to decision trees that can offer detailed explanations to the user.

2.4 Unsupervised Learning Techniques

2.4.1 Clustering

Chen *et al.* (1996) defines clustering as “the process of grouping physical or abstract objects into classes of similar objects”. In this technique, a model is created that classifies data by grouping together objects with similar characteristics. By splitting the data into these clusters we can identify patterns within the data. This can allow us to gain a high level view of the major groups within a given set of data, or to highlight outliers that do not fit into any particular group (Berson *et al.* 1999).

In general, clusters are created by identifying objects that have similar values for a subset of their characteristics. This process is similar to the nearest neighbour method, as objects are identified as being “near” each other. Here however, the goal is to provide a set of grouped data as the result rather than aiming to predict future data. Another way to group objects is by focussing on finding clusters of objects that have coherent behaviours rather than objects that are physically close to each other (Wang *et al.* 2002). This allows us to identify groups of objects that behave in a similar way without their characteristics being necessarily similar.

The number of clusters generated is key to how useful the results of the algorithm are to the user. Too few clusters will not split the data enough, but too many will make it
difficult to identify patterns. The sensible number of clusters to choose is often based on the desired outcome.

Two types of clustering are available: hierarchical and non-hierarchical. In non-hierarchical clustering, the user needs to specify the number of clusters to be created. Many algorithms will then offer an option to increase or decrease the number of clusters if the initial results are not as desired.

Hierarchical clustering does not require a number of clusters to be specified. A hierarchical representation of clusters is produced, with one cluster containing all the data at the highest level, and clusters containing just one record each at the lowest level (Hastie et al. 2001). This allows us to see a range of cluster sizes. Different hierarchical clustering algorithms build the hierarchies in different ways. The most common method, agglomerative, builds the tree from the bottom up, beginning with one record in each cluster and joining them together. The divisive method builds from the top down, splitting the entire data set into increasingly smaller clusters.

### 2.4.2 Association rules

According to Chen et al. (1996), “in many applications, interesting associations among data items often occur at a relatively high concept level”. Association rules are rules that can be extracted from the data from all levels. The algorithm goes through the data set, picking out any patterns that emerge. These rules are then presented to the user, usually in order of the percentage of the time that the rule is correct and how often the rule occurs.

A rule consists of two halves. The first part is called the antecedent and this is the condition or conditions for the rule to occur. The second part is called the consequent and this is what happens when the rule occurs. There is usually just one condition for the consequent (Berson et al. 1999). For example, a rule could be structured as follows:

\[
\text{If Condition 1 and Condition 2 then Consequence}
\]

Each rule also has a measure of confidence, the percentage of the time the rule applies, and support, the number of times this rule occurs.

An association rules algorithm will find all the rules within a data set. This will often be a very large set of rules, and many of them will be useless. It is therefore important for an algorithm to present the most “interesting” rules to the user, allowing them to take useful information from the data.

As well as support and confidence, there are many other factors that can be used to define how “interesting” a rule is. Liu et al. (2000) identifies two key measurements as unexpectedness and actionability. Rules are deemed more interesting if they go against the users’ expectations, and if the user can do something with them to his/her advantage. Achieving effective implementations of these factors is a difficult task, and will often vary greatly depending on the input data and desired outcome.
2.5 Other Data Mining Concepts

2.5.1 Input and output of data mining techniques

Features of data in databases can be either numerical (have a particular value) or categorical (fall into a particular category).

Each different data mining technique will have different possible input types and may also have a number of parameters to specify certain characteristics of the algorithm. The new interface will need to deal with each of these possibilities in an appropriate and intuitive manner. It will also need to provide helpful error messages for data that is incompatible with the chosen algorithm or function.

Frawley et al. (1992) tells us that the activity of knowledge discovery “involves two processes: identifying interesting patterns and describing them in a concise and meaningful manner”. Without a clear and informative output that the user can understand, the data mined from the database will be useless. It will be important to ensure that the information output from the algorithms is presented in a useful manner to the user.

The output of each algorithm is likely to differ slightly dependant on the type of algorithm selected and any parameters specified by the user, but there will be a number of features that are common to most algorithms.

In general, the results of running the algorithm will be displayed in textual form, along with an assessment of the level of error in the results.

The current Weka interface splits the algorithms into different categories and display certain features of the results for each one. The new interface will need to display similar data:

- Classification algorithms display a ‘confusion matrix’ that identifies which data items in the training set have been classified correctly and incorrectly. It is possible to view the classification error in a visual form.
- Clustering algorithms display the list of clusters and how many items appear in each one. It is possible to see a visual representation of the clusters.
- Association rules algorithms display the rules generated according to the users’ specifications.
- The attribute selection algorithms display the attributes that have been selected as the most valuable.

Each individual algorithm will also display statistics dependent on it’s particular features.

2.5.2 Preprocessing

Data may need to go through a preprocessing stage before a data mining algorithm can be applied to it. This involves steps such as ensuring the numerical values are in the correct range. For example, Berson et al. (1999) tells us that neural networks require a great deal of preprocessing, as all the data must be normalised to lie between 0.0 and 1.0.
Often the amount of data in the data set will be too great, resulting in the creation of a model being too computationally expensive or less accurate models being created. Becher et al. (2000) identifies a number of exploratory data analysis techniques as one possible way to reduce the data to a more manageable size in an effective way:

- Identifying inappropriate and suspicious attributes. Attributes are removed if they contain too few values, are nearly constant or are not effective predictors.
- Selecting the most appropriate attribute representation. Target dependency analysis is used to measure the associations between source and target attributes. They are then grouped to give a set of discrete categorical attributes to make assessing their overall benefit easier.
- Creating derived attributes. Transformations are applied to groups of attributes to combine them if the combined value will be of more worth.
- Choosing an optimal subset of attributes. The goal of attribute selection is to select a subset of attributes without significantly affecting the overall quality of the resultant model.

2.5.3 Estimating the accuracy of a prediction model

Many different algorithms are available for each data mining technique, and so we need a method to choose the most appropriate one for the data set we are analysing. Moore and Lee (1994) tells us:

Model selection is an important aspect of many areas of supervised learning… The best model is the one that is expected to give the most accurate predictions for future data.

Comparisons of the performance of algorithms in Weka can be performed using the functions available in the Experimenter section of the current interface.

To perform this comparison we need to get estimates for the accuracy of the potential prediction models. Hastie et al. (2001) tells us that:

Ideally if we had enough data, we would set aside a validation set and use it to assess the performance of our prediction model. Since data are often scarce, this is usually not possible.

One technique that avoids this problem and is implemented in Weka is cross-validation.

Rather than having one set of data to test the prediction accuracy of the model against, K-fold cross-validation splits the data into ‘K’ subsets. One of these subsets is left out of the model while the algorithm ‘learns’ its predictions. The subset that has been left out is then used to test the accuracy of the prediction. This process is repeated for each of the subsets in turn, and the average of all the accuracy measurements is taken to give a final result.
2.5.4 Visualizing data mining results

Weka contains functionality to visualize the data in simple two dimensional plots. In the ‘Visualize’ section, each feature of the data is plotted against each other feature. The results of certain algorithms can also be plotted in this way.

More complex methods of visualizing data mining information are available and research into these techniques continues today. Visualizing the results of data mining allows the user to see results without the need to understand complex mathematical equations and can easily deal with highly noisy data (Keim 2002). Displaying large amounts of data with more than three dimensions is a challenge, but if implemented well can provide an excellent overview of the data to the user. Keim and Kriegel (1996) tells us that another important factor is the ability for the user to directly interact with the visualizations.

Weka does not implement these complex visualization techniques and does not facilitate a great deal of interaction with the visualization it does provide. However, a significant element of this project is to allow features such as this to be easily incorporated using the new interface, and so it will be important to ensure this is possible through the design that is created.

2.6 Conclusions

Following this investigation into the key features of data mining, a far greater understanding of the processes involved in using Weka has been achieved. This will allow a deeper insight into how the users of Weka will wish to use the software, and therefore will contribute significantly to the design process. It has also emphasised the fact that Weka provides functionality for a great deal of the data mining techniques, and can therefore be used for a great many applications.

Each of the data mining techniques is implemented in different ways, and each will have different input and output information, as well as different settings defining how they can be run. Unifying the different techniques into a single, consistent interface will be one of the biggest obstacles to overcome during this project.

As noted earlier, different authors often use different terminology for similar concepts, and so one of the challenges will be deciding on the most appropriate language to use when referring to data mining terms within the interface. Having attempted to explain these concepts in fairly simple terms should also make it easier to provide assistance to users of the system who are new to data mining.

Data mining has many powerful and wide reaching benefits, and, with a more user-friendly interface, Weka could be an extremely useful tool that allows users to easily take advantage of this fact.
3. User Interface Design

As Hix and Hartson (1993) tells us, “to users, the interface is the system”. In general, the user will not be interested in how the functionality of a piece of software is implemented or how it works; they will just want to be able to use it for their needs. Clearly, if the user interface gets in the way of this task then it is not doing its job.

3.1 Identifying and Understanding the Users

3.1.1 Why are users important?

One of the key principles in developing an effective user interface is to identify and understand the users who will be using it. Lewis and Rieman (1993) tell us that no matter how useful a piece of software seems to you, unless you can find a specific example of a user who will want to use it in the way you anticipate, it will usually turn out to be useless.

3.1.2 Who are the users in this project?

Unfortunately, in this project the users will be wide ranging in both background and location, a problem that means it will be a challenge to identify user tasks that are representative of all possible uses of the software. Being able to meet users during the course of this project is an unlikely prospect due to the fact that they are spread all around the world. This will make certain elements of the project, such as identifying requirements, far harder.

In general, users are domain experts who have an in-depth knowledge about the subject area. As my interaction with users in this project will be restricted, it is very important that to compensate for this, a full understanding of all the topics discussed in the Data Mining section (Chapter 2) of this document has been achieved.

3.2 Usability

3.2.1 What is usability?

The usability of the system is inherently linked to the design of the user interface, and will be a key measure of how successful the project is. However, attaining a measure of the usability of a system is not an easy task. Hix and Hartson (1993 cited Shneiderman 1992) define usability as a combination of the following user-oriented characteristics:

- Ease of learning
- High speed of user task performance
- Low user error rate
- Subjective user satisfaction
- User retention over time

In each aspect of the interface design created, it will be important to ensure that each of these features are considered.
3.3 Design Process

Planning is a key element in the design of any piece of software, and this is especially true in the design of a user interface. There are many different approaches that can be taken, and choosing the correct one to suit this particular project could become a key decision in affecting the overall success. Time constraints mean that if any significant errors are made in the early processes of defining requirements and initial designs, complete redesigns will not be possible.

3.3.1 Identify users and tasks

As discussed earlier, identifying the users for a project is a vital task and so needs to be completed as early as possible in the project. Once users have been identified, it can be very helpful to identify representative tasks that the system will be used to accomplish (Lewis and Rieman 1993). These tasks should cover as much of the system as possible.

Hix and Hartson (1993) emphasises the distinction between the behavioural and constructional domains in the development of a successful user interface design project. Descriptions of the two domains are given as follows:

In the *behavioural domain*, interaction is described abstractly (i.e., independently of software) in terms of the behaviour of the user and the interface as they interact with each other.

In the *constructional domain*, software engineers implement the software that implements the behavioural design.

Although this project will be completed individually and will not have different people working on different domains, it is still important that the stages of the project are considered in the correct way. For example, a view of the user interface design from the constructional domain will only consider how the coding of the software can be implemented, and may neglect some of the key usability issues that need to be addressed. Therefore, as the process of identifying tasks is based in the behavioural domain, it will need to be considered within this context.

3.3.2 Use existing design ideas

When ready to begin work on the actual design of the interface, an excellent way to start off is to adapt existing interfaces into the design and build on them. Lewis and Rieman (1993) emphasises the importance of “intelligent borrowing” in the design of a new user interface for the following reasons:

- It is unlikely that the ideas you can come up with will be of higher quality than the best examples of user interface design around today.
- If ideas from popular interfaces are used, it is likely that the users will already be familiar with the concepts, reducing the time it takes to learn how to use the interface.
- Borrowing ideas can also save a great deal of time it would take to invent new ones.
This project will aim to take advantage of this idea, as it should help to increase the usability of the new interface, and the time constraints will limit the amount of time available to invent new ideas. Common concepts used in Windows-based programs will be a good place to start, for example, the popular “Windows, Icons, Menus, Pointers” paradigm. This will also extend to using similar menu names and having similar functions within the menus. Most users will be familiar with these interaction techniques already, thus increasing the ease of learning.

It is vital, however, to ensure that existing interfaces are not replicated too closely, or there may be the danger of becoming guilty of plagiarism and possibly infringing on copyright regulations.

3.4 Evaluating user interfaces

Evaluation of the usability of a user interface will be a vital area in this project. An evaluation of the existing interface will highlight problems that exist in the current system. I will also need to fully evaluate the interface I design to assess its success and to identify further improvements.

3.4.1 Evaluation by users

Evaluation by users gives an excellent insight into what the people who are actually going to use the system think of the interface. Users will often identify problems not apparent to the designer of the interface.

It is unlikely that I will be able to meet and talk to users directly, and so I will invite them to evaluate the interface using an online questionnaire developed by Perlman (No Year). A number of questionnaire styles are available, but the one I have deemed most appropriate for this task is the Computer System Usability Questionnaire developed by Lewis (1995). The questions in this questionnaire are explained in simple terms that users who do not have a great deal of knowledge of user interface terminology will be able to understand. There are 19 questions on this questionnaire, and I feel this is an appropriate number. It is enough to give me the users’ opinions on the interface, but is not too many to put off users who will only be prepared to spend a limited time filling in such a questionnaire.

3.4.2 Evaluation without users

Users’ time is limited, and therefore a great deal of the evaluation will be done without them. The users of Weka are unlikely to be experts in user interface design, and so only completing techniques involving evaluation by users would be highly likely to miss many usability problems. Also, experienced users will now be used to features of the interface that are potentially problematic for new users, and so would not highlight them as issues.
Lewis and Rieman (1993) describes three approaches to evaluating user interfaces in the absence of users:

- A cognitive walkthrough is the process of going through the system imagining the thoughts and feelings of a first time user. We then attempt to create a believable “story” about each action the user has to take to complete a given task.
- Action analysis evaluates how long it will take a user to complete a task by using simple estimates for each mental action a user will have to make.
- Heuristic evaluation evaluates the interface against a set of guidelines. One of the most famous lists of usability heuristics was developed by Nielsen (1994) who identifies ten guidelines that an interface should adhere to.

To effectively identify all usability problems of an interface, at least two of the techniques should be applied, if not all three. These techniques are particularly useful for analysing prototypes and designs during the development stage of the project and so they will need to be applied over the iterations of the design and implementation periods of the project.

### 3.5 Conclusions

One of the key points to take from this investigation is the importance of using elements that are common in user interfaces of other software products. Users are likely to have experience in using these programs and should therefore feel comfortable using many of the features in them. The usability of common interaction methods such as menus has been thoroughly tested and so including these in the new interface will help make it more user-friendly.

Another important point taken from the analysis is the importance of testing the system for usability issues. This process will need to be performed for both the current interface and the new interface once it has been implemented. The evaluation of the current interface will highlight problems that need to be eliminated in the new interface, and help begin the requirements generation progress. The evaluation of the new interface will identify potential improvements to be made and ultimately help decide whether it is successful in its goals.

Finally, the importance of users in the design of a user interface has also been highlighted. While it will be important to apply recognised usability standards to the new interface, an interface that covers all the “scientific” requirements will still be a failure if the users cannot operate it effectively. Therefore it will be vital to get as much user input as possible to enhance the chances of making the interface a success.
4. Evaluation of Current Interface

4.1 Introduction

In any major redesign of a user interface, it is vital that a full evaluation of the current interface is completed before the design work begins. This should identify usability problems in the existing interface to ensure that the new design improves these features. It should also identify areas of the current interface that do work well in order to ensure that these are carried over to the new interface.

4.2 Overview of the current interface

When Weka is started up the user is given four options for the interface they wish to use to interact with the system. The first is the command line interface that simply allows users to type in commands to perform the system functions.

The next option is the Explorer interface. This is a graphical user interface and is the main interface of the system. It allows the user to apply algorithms to data sets and to visualize results.

The third option is the Experimenter interface. Again, this is a graphical user interface, but the function of this part of the system is to compare the performance of different algorithms.

The final option is the Knowledge flow interface. This can be used as an alternative to the Explorer interface, with slightly limited functionality. It aims to allow the user to perform the same tasks but with a more graphical representation to show the flow of information.

The following is a brief overview of how the three graphical interfaces are implemented and how they can be used. An overview of the interfaces along with screenshots of the most important screens can be found in Appendix A.

4.2.1 Explorer

The explorer interface is based around six tabs that are accessed from the top of the screen. At the bottom of the screen are three consistently visible elements of the interface. The ‘Status’ bar that gives a textual description of the current status of the system, and next to this is the ‘Log’ button that allows the user to access the log of messages stored by the system. The final element is an image of a bird (of the “Weka” species). This signifies whether or not the system is currently processing a task. When the system is idle, the bird is pictured sitting down and stationary. While the system is performing a task, the bird will be shown standing and moving back and forth. A number next to the bird signifies the number of processes currently running, with ‘x1’ representing one running task, ‘x2’ representing two and so forth.

The main body of the window shows the contents of the currently selected tab.
The first tab is opened when the Explorer interface is first selected and is labelled ‘Preprocessing’. From this tab the user can open a data set, view details about the attributes and apply filters to remove certain elements of the data.

The algorithms available in Weka are split into categories, and the user must select a different tab for each type. These tabs are as follows:

- Classify – classification algorithms to build models that predict the class of future instances are applied from here.
- Cluster – clustering algorithms to group records into clusters are applied from here.
- Associate – algorithms to generate rules based on the data set are generated from this tab.
- Select Attributes – algorithms to select the most relevant attributes are applied from here.

The final tab is the ‘Visualize’ tab. This tab offers the user various options on viewing two-dimensional plots of the attributes in the data set.

4.2.2 Experimenter

The Experimenter interface is based around three tabs, again accessed from the top of the screen. The first tab is the ‘Setup’ tab, where the user can select the data sets and algorithms they wish to evaluate. They can also set the parameters of the experiment.

The ‘Run’ tab simply has ‘Start’ and ‘Stop’ buttons at the top, and a text window to log the progress of the experiment.

The ‘Analyse’ tab allows the user to view an analysis of the results of either an experiment that has just been run, or one that was loaded from a file.

4.2.3 Knowledge Flow

The Knowledge Flow interface offers the user an alternative to the Explorer interface. The functions of Weka are split into components that can be added to a workspace and connected together. For example, a data set in ARFF format is loaded by placing an ‘Arff Loader’ component in the workspace and configuring it. An algorithm can be applied to the data set by placing the component for the required algorithm on the workspace and connecting the Loader component to it.

The main area of the screen contains the workspace, with a Status bar and ‘Log’ button at the bottom. At the top of the screen is the toolbox containing each of the available components. The components are split into categories, and tabs are used to select between the different categories.

At the top left of the screen are located ‘Save’, ‘Open’ and ‘Stop all execution’ buttons.
4.3 Evaluation by Users

Users are in an ideal position to evaluate the interface as they have experience of using it and may well have experienced problems with it in the past. They are domain experts who understand the functionality offered by the system and so any problems they experience should be down to the interface of the program rather than not understanding how to perform a task.

4.3.1 The process

As talking to users directly was not possible, they were invited to evaluate the interface using the Computer System Usability Questionnaire developed by Lewis (1995) and implemented online by Perlman (No Year). The questionnaire contains 19 questions allowing the user to give their opinion on certain elements of the interface, and also gives users the opportunity to highlight the most positive and negative aspects of the interface.

This request was made via the Weka mailing list. With a questionnaire such as this, different users may interpret the levels of responses (ranging from agreeing strongly with a statement about the interface to disagreeing strongly) slightly differently. This problem is reduced with a greater number of responses, so the more completed questionnaires received, the more accurate the results will be. The number of responses received was slightly limited, with only eight being returned, but this was enough to give a basic insight into the areas of the interface that caused the most significant issues for current users of the system.

4.3.2 The results

The full details of the user testing results can be found in Appendix D.

The inability of users to recover from mistakes they made was highlighted as a significant problem with the current interface. Weka does not feature many opportunities to "go back" a step, and it is often difficult to undo mistakes. In some cases the only way to fix a problem may be to completely start again, which could prove extremely inconvenient if, for example, a lot of parameters have been set up for an algorithm.

Error messages were identified as an area of the interface that is significantly lacking. The error messages in Weka are not especially helpful, and do not assist users in recovering from situations in which they have encountered problems.

Another problem highlighted by the users’ responses was the inability to find the information they required. There is often a great deal of information on screen at one time, and this can make it harder to locate the particular feature that is required. A lot of information, if available at all through the interface, is often hidden in areas not necessarily expected by the user. This can make it a lot harder for a user to perform processes that they wish to.

The fact that the information provided by the system was not easy to understand suggests that users may have had some problems understanding some of the results
output as well. Sometimes symbols are used to represent results but are not fully explained and so this could be part of the problem.

A number of users indicated that their preferred interface to use was the command line interface. This points to an over-complicated graphical user interface as the only alternative, or possibly that for experienced users who often need to complete similar tasks regularly, the command line interface could prove more efficient. As a possible improvement to the graphical user interface, it could be made easier to repeat common tasks, for example by allowing easy saving of combinations of settings, or being able to set default parameters that load when the software is started.

4.4 Evaluation without Users

Users are unlikely to be experts in user interface usability concepts and therefore may miss some significant problems in the design. Also, if a user has been using the system for a long time, they may have become used to some of the problems and will no longer identify them as issues. Therefore it is important to perform additional evaluation without the assistance of users.

4.4.1 Techniques

Two techniques were used to evaluate the current interface without users. The first was the cognitive walkthrough technique. Lewis and Riemann (1993) define the cognitive walkthrough as “a formalized way of imagining people’s thoughts and actions when they use an interface for the first time”. A set of tasks likely to be performed by users is developed and the tester works through each of these tasks using the interface. With each action taken, the tester imagines what the user will be thinking and what step the user is likely to take next. The aim is to create a believable “story” of how a user will complete a task and what they are thinking at each step.

The second technique used was the heuristic evaluation technique. This technique involves comparing the interface with a series of guidelines or heuristics in order to identify usability problems that exist within the interface. Ideally, this kind of evaluation should be performed by multiple expert evaluators. Unfortunately this was not possible within the scope of this project, as evaluators with the necessary level of experience were not available to participate. A single evaluation was carried out, and therefore a number of usability problems may have been missed. However, in combination with the cognitive walkthrough, it was felt this was sufficient to highlight the main shortcomings of the existing interface. The heuristics used were the ten usability heuristics developed by Nielsen (1994).

4.4.2 Selection of tasks

An initial step in the evaluation of the current interface is to select a number of representative tasks that will need to be performed by the user. The tasks need to be selected to ensure they cover the entirety of the system so that all areas of the interface can be evaluated.
In the Explorer interface, the application of each different type of data mining algorithm is performed from a separate tab, but requires a similar process for each one. Due to the fact that the application of each type of algorithm should be largely the same, it could be considered as just one task to cover all the types. However, it was decided that the most effective approach would be to consider each different type of algorithm as a different task. This will ensure that problems in each algorithm type will be detected, and it would highlight key usability problems that may occur throughout the interface. In general, tasks should be considered independently from the process of completing them, but it was felt that in this situation the requirement to discover all usability issues should take priority.

The new interface will aim to standardise the process for applying algorithms as much as possible, and may, depending on analysis of the most appropriate design, make it possible to apply all types of algorithm using just one function. In this case just one task could be considered for the application of an algorithm to a data set.

To ensure that as many potential situations that could occur as possible are tested, these tasks may need to be applied multiple times to different data sets and different examples of algorithms. For example, some algorithms may require data to be of a certain type and it is important to test how the interface reacts in a situation where the data is not of the correct type.

Tasks were selected for each of the three interfaces. The tasks selected were as follows:

4.4.2.1 Explorer

1. Open data set.
2. Apply preprocessing filter to a data set.
3. Apply classification algorithm to a data set.
4. Apply clustering algorithm to a data set.
5. Apply association rules algorithm to a data set.
6. Apply algorithm to select most relevant attribute to a data set.
7. View a visual relationship between two attributes of a data set.

4.4.2.2 Experimenter

1. Run an experiment.
2. Analyse the results of an experiment that has been run.

4.4.2.3 Knowledge Flow

1. Open data set.
2. Apply preprocessing filter to a data set.
3. Apply an algorithm to a data set.
4. View results of an algorithm.
4.4.3 Cognitive walkthrough results

We will now discuss the key issues identified by the cognitive walkthrough evaluation.

4.4.3.1 Explorer

When a user wishes to select an algorithm to apply to the data, no descriptions of the algorithms are available. This means that if a user does not know the name of a specific algorithm they wish to use, they will have to guess which one they should select. Once they have selected an algorithm, they can access a lot more data on it. However this involves a number of extra clicks and if they have to go through this process for each available algorithm it will be very time consuming.

When an algorithm has been selected, its name is displayed in the bar towards the top of the screen. It is also followed by the default parameter settings in the form of a hyphen character followed by the letter code to represent the parameter and then the value. For new users, this could be very confusing, as they may not even realise these are parameters of the algorithm, and will seemingly just be a random string of letters and numbers. If the user does realise that these are parameters, unless they know what all the codes mean, it will still be difficult to understand what parameters they represent.

Changing the parameters could also cause problems too. There is no indication that clicking on the name of the algorithm will allow the user to perform this task. If the user does not realise this, they will be unable to change the parameters and will not be able to access the further information about the algorithm and what each of the parameters does.

When the system is busy, for example when a user applies an algorithm that will take some time to process, the bird at the bottom of the screen becomes animated and the Status bar may give a description of what the system is processing. However, this may not be obvious to the user. When they start an algorithm, they will be unlikely to focus on the bottom of the screen and therefore may not realise that the system is still processing their instruction. They may become concerned that what they have done has not worked for some reason and attempt to rectify it as a problem.

An unintuitive element of the system is the fact that results of algorithms cannot be copied in a way that might be expected. If a user wishes to insert the results of an algorithm into a report, they are likely to instinctively want to use standard copying and pasting methods. In most programs, this would involve selecting the text they wish to copy and then selecting ‘Copy’ from the ‘Edit’ menu or right clicking on the selected text and clicking ‘Copy’ from the menu that appears. In this interface, there is no Edit menu, and right clicking has no effect. The common ‘Ctrl+C’ shortcut key combination does work, although many users may not be aware of this.

Similarly, saving the results of an algorithm is not an intuitive task. The common method would be to use ‘Save’ from a ‘File’ menu, or use a ‘Ctrl+S’ shortcut key combination. Instead, the user must either right click on the name of the results in the results list on the left hand side of the screen and click ‘Save results buffer’ or use the unusual key combination of ‘Alt+Shift’ and click the mouse button to open the save dialog.

When a filter or algorithm is applied to a data set, the attribute to be used as the class can be selected using a drop-down menu. This drop-down menu is unlabelled and is therefore prone to confuse the user or simply be ignored. For the preprocessing filters
this is especially problematic as it is located in a completely different section of the screen to where the filters are applied.

When algorithms are run multiple times on different data sets, the results for each are listed in the ‘Results list’ on the appropriate tab. It could become confusing for the user if they have a long list of results from different data sets, as the name of the data set is not listed. They may have to click on each result and check the name of the data set at the top of the results to find one they are looking for.

The ‘Ignore Attributes’ on the Clustering tab and ‘Select Attributes’ on the Visualize tab could be confusing for the user, especially those who have limited experience with computers. To select more than one attribute from the list at the same time, the Ctrl key must be held down, or the Shift key can be used to select multiple consecutive records. Although these are fairly common techniques for selecting multiple records, they may not be the most appropriate here as this is the only function of this segment of the interface. If users do not realise how to do this they will be unable to use this function effectively. It is also inconsistent with the preprocessing tab where records can be selected using tick boxes.

One of the first steps a user is likely to want to take after opening a data set is to select the most useful attributes in the data. Although it is possible for them to manually select the attributes they want from the Preprocessing panel, the algorithms designed to perform this task are located on a completely separate tab, which is a little counter-intuitive. Having run an algorithm, the user must then remember the attributes that have been selected, go back to the Preprocessing tab, and manually remove them.

4.4.3.2 Experimenter

In setting up an experiment, the user will be aiming to compare the performance of different algorithms on one or more data sets. Therefore it seems natural that the initial steps a user will want to take is to select the algorithms and data sets to be used in the experiment. However, the interface components to allow them to do this are located at the bottom of the screen underneath the various other experiment settings.

Another problem is that data sets can only be added from files, and not URL’s and databases as are available in the Explorer interface. If a user’s only access to a data set is through one of these routes, they will need to open it in the Explorer interface, save it as a file and then go back the Experimenter interface. This is inconvenient and time consuming.

When using the ‘Advanced’ experiment settings, the algorithms to be used are specified using the ‘Generator Properties’ section. However, the manner in which settings can be made using this functionality is highly unintuitive and does not allow the user to set both the algorithms and the other settings, as changing one of the other settings cancels any algorithm selection the user has performed and vice versa. Another issue is that when a user selects the option to specify the algorithms, the default algorithm (‘ZeroR’) is placed in the list. If the user does not wish to use this algorithm they must remove it from the list, select a new algorithm and then add it to the list.

Once the user has finished setting up an experiment they would expect to be able to run it using a button at the bottom of the screen, as they will have naturally worked their way down the screen through the settings. However, the user must use the ‘Run’ tab accessed from the top of the screen to begin the experiment.
Once an experiment has been run, the user can use the ‘Analyse’ tab to analyse the results of the experiment. However, when the user opens the tab, they are not able to analyse an experiment they have just run straight away, they must first click the ‘Experiment’ button in the top right of the screen. This button is not immediately obvious as it is located next to the ‘File’ and ‘Database’ buttons, and therefore the user may become confused at how to analyse an experiment they have just run. It would seem far more intuitive to automatically set the experiment just run as the default so that the user does not have to click the ‘Experiment’ button.

Once an experiment has been analysed, the results are not set out in a particularly clear way. The column headings to compare the performance of the algorithms are too small and as many algorithms start with similar sub-names it can be difficult or impossible to distinguish between them. There is also no obvious indication as to what some of the symbols used represent.

4.4.3.3 Knowledge Flow

One problem that immediately became obvious was the lack of ‘New’ button to compliment the ‘Open’ and ‘Save’ buttons. There is no way to easily start a new, blank workspace. Closing down the Knowledge Flow interface does not even solve this problem, as the components still appear in the same positions, although cannot now be used. Weka must be completely shut down to achieve a blank workspace.

Running the algorithms involves selecting ‘Start Loading’ from the menu that appears when the user right clicks on the data set loader component. This does not seem particularly intuitive; it may seem more sensible to have a more obvious ‘Process’ button. However, as it is possible to have more than one data set on a single workspace, there would still need to be a specific ‘Process’ option for each loader component.

Another potential problem is that it is not immediately obvious whether a process has been successful or not. Some of the algorithms have their component icons animated when they are processing, but when other processes are in progress it is impossible for the user to tell whether or not their command has been successful. Error messages tend not to appear, and the user has to click the ‘Log’ button to check whether any errors have occurred.

4.4.4 Heuristic evaluation results

Outlined below is a selection of the key usability issues identified using the ten heuristics identified by Nielsen (1994).

4.4.4.1 Visibility of system status

The main issue identified under this heuristic was the lack of significant indication as to when the system is processing something. For example, when an algorithm is run on a data set, the system may take a little while to complete the algorithm, but this fact may not be obvious to the user. The only indicators are the system status bar and bird icon located at the bottom of the screen and these may be missed by some users, especially those who are new to the system.
4.4.4.2 Match between the system and the real world

In general the system uses common data mining terms, and so users with data mining experience should feel comfortable. However, if a user is not particularly experienced in the field of data mining, there are certain areas where an alternative description in simpler language may be a benefit. For example, some of the names of the algorithms do not give much clue as to what they do, and no further description is available when the user is choosing which one they wish to use.

4.4.4.3 User control and freedom

If users make a mistake, there is no option for them to undo their last action, and so in some situations it may become difficult for them to recover from mistakes they make. Although a lot of the actions users take will to be change settings, and so mistakes should be easily rectifiable without an undo function, there will still be a number of situations where it becomes an issue. For example, if a user accidentally changes a setting and then cannot remember what its original value was.

4.4.4.4 Consistency and standards

The fact that there are separate interfaces for the Explorer and Experimenter shows a lack of consistency in the interface. There are some small similarities between the two interfaces but they are mostly significantly different. The experimenter is particularly inconsistent, with significant layout differences between just the ‘Simple’ and ‘Advanced’ experiment options.

Within the Explorer, the tabs for applying different algorithms are fairly similar to each other, with some minor differences to reflect the differences in the algorithm types. However, the preprocessing tab is quite different. For example, all algorithms are applied by clicking the ‘Start’ button on the middle left hand side of the screen, while the preprocessing filters are applied by clicking the ‘Apply’ button located at the top right of the screen.

The interface lacks many of the standard user interface components such as menus that are used in many popular programs. This can mean that users will feel less comfortable using the interface as they may instinctively look to certain components to perform their desired action.

4.4.4.5 Error prevention

An error that can easily occur is the application of an algorithm to a data set that is not compatible with it, for example if it has the wrong input type. This is a difficult error to avoid, as it is dependant on the data set and algorithm that are chosen by the user. However, clearer descriptions of the algorithms could help alleviate this problem.

4.4.4.6 Recognition rather than recall

Details of the data set, such as number of instances, are only displayed on the Preprocessing tab of the Explorer interface. This means that if a user needs to check a detail of the data set, such as the data type of one of the attributes, when they are applying an algorithm they will need to go to the preprocessing tab, remember the detail, then go to the algorithm tab and make a decision based on their memory. This can lead to mistakes and generally makes the task less easy for the user.
Having run an algorithm to choose the most appropriate attributes from the ‘Select Attributes’ tab, the user must then remember the results of the algorithm, click on the Preprocessing tab, and manually remove the attributes. If this involves a long list of attributes, remembering them all will be an extremely difficult task, and users are likely to have to switch between the two tabs multiple times.

4.4.4.7 Flexibility and ease of use

The current interface does not make use of accelerator keys. These can improve the speed at which users, especially experienced ones, are able to perform common tasks.

Each time Weka is started, there is no record stored of the directory in which the last data set opened was stored. This is a fairly standard process in many programs, and if all a user’s data sets are stored in the same directory it can save them a lot of time and effort, especially if the directory is quite a few steps away from the default Weka directory. Similarly, the options a user selects are never remembered when Weka is restarted. If a user regularly uses the same options, and has to set them up each time they begin the program it could become time consuming and frustrating.

4.4.4.8 Aesthetic and minimalist design

There is often a lot of information displayed on-screen at one time, as there are many different options available to the user to set the parameters of their work. This can be confusing to the user, as it becomes harder to pick out the piece of information that they require. The interface would probably benefit from less information on-screen at any one time, and the information being displayed more clearly and more spread out.

4.4.4.9 Help users recognize, diagnose, and recover from errors

Error messages in Weka are generally quite limited, offering just a basic description of the error, but not necessarily giving any advice to the user about what is likely to rectify the problem.

In certain situations, such as the application of an association rules algorithm to data with numerical input, no error message appears to the user. In the ‘Status’ bar, a message telling the user to ‘See error log’ appears. The user is unlikely to notice this as it is not obvious and is not highlighted in any way. There is no other indication that an error has occurred, and the user could be left waiting for the results of the algorithm to be displayed, as they have not realised that an error has occurred. If they do notice it they must then click the ‘Log’ button where they will be told that the rule cannot be applied to numeric attributes.

4.4.4.10 Help and documentation

A lot of help files, tutorials and documentation regarding Weka can be found online. However, it can be quite difficult to find and often requires looking through a great deal of text to find a particular piece of information that a user may require.

Within Weka the help available is limited. This means that users will often have refer to the online documentation which can be a time consuming activity and not ideal if the just need to find a small piece of information. It would be far easier to have a ‘Context
Help’ function offering immediate information on the screen the user is currently viewing.

4.5 Conclusions

The large number of usability issues identified by the evaluation of the current interface gives added justification to the need for the user interface to be redesigned. Problems in the interface were made apparent by the users themselves, and were reinforced by the two techniques used to analyse the interface without users.

Using the three different methods to collect potential usability problems appears to be a well-founded decision, as each one identified issues that were not highlighted in the others. Although it is possible further problems could have been identified through further evaluation techniques such as action analysis, it was decided that the time taken would outweigh the benefits of such a process within the constraints of this project.

Having identified the key usability problems in the current interface, the requirements gathering process will need to translate these issues into requirements for the new interface. This will ensure it does not suffer from the same problems.
5. Requirements

5.1 Introduction

Requirements are a vital part of any software-related project. Failure to set out a clear set of objectives for a piece of software can result in the final product not meeting the goals that it needs to achieve. The requirements will need to be thorough and unambiguous to ensure that misinterpreted or omitted requirements are not translated to the final design.

Requirements can be grouped into many different categories, and these can often prove useful in the development of a full system. However, as this project is only concerned with the user interface of a system, it was decided that the requirements identified could be satisfactorily classified as either functional (those requirements firmly linked to the functionality of the interface) or non-functional (requirements that are not directly functional, such as usability standards).

Developing requirements for a project such as this can prove tricky, as there are limited sources to gather them from. As it is not possible to communicate directly with current users of Weka (due to their widespread locations), it is not possible to clarify the exact requirements of the new interface with them. This is a significant problem, as users are in an ideal position to clarify exactly how they wish to use Weka, which is a key factor in the development of requirements for a user interface.

To ensure that a full and accurate set of requirements is developed despite the lack of users, a careful development plan is needed. The sources for the requirements will include the current interface, background work on standard usability requirements and the development of use cases to model the users’ interaction with the system.

The full list of requirements and use cases can be found in Appendix B.

5.2 Development of Requirements

Basic functional requirements were developed using the functions of the current interface. It is vital that the new interface offers all the functionality of the existing interface, but in a more user-friendly way. Therefore, the abilities to perform the functions within the current system were the first requirements generated, along with some simple non-functional usability related requirements.

Further requirements for the new interface were developed from work completed in earlier stages of the project development. The evaluation of the current interface provided a key starting point in this area. Requirements could be developed through the need to remove problems identified in the current interface.

Uses cases were developed for key tasks to identify further requirements (see section 5.3 Use Cases below). Each of the use cases could be developed into a set of requirements for the particular user task they related to. As well as the basic requirement that the interface must allow the user to complete the relevant task, the steps involved in each use case helped clarify the more detailed functions involved in the user’s interaction with the system, and thus the requirements for each task.
The requirements document continued to evolve during the design process. As the designs were developed, additional requirements emerged that had not been previously considered. These needed to be added to the requirements document to allow a comparison between the final system and the requirements towards the end of the project.

Having compiled a list of requirements, it is vital to ensure that they represent a valid list and fully represented the needs of the new system. Following the stages of the requirements gathering process, validation reviews of the list were carried out. The requirements were checked using the five criteria suggested by Sommerville (2001) to ensure that they were valid, consistent, complete, realistic and verifiable. In general, these reviews only resulted in minor discrepancies, and making amendments proved to be a relatively simple task.

5.3 Use Cases

5.3.1 Development of use cases

Use cases are used to describe the users’ interaction with the system to achieve their goals. The structure used to present the use cases is the one suggested by Preece et al. (2002). A use case is created for each user task, and a numbered list is created describing the steps the user takes and the response of the system at each step towards achieving the user goal. Also described are the possible alternative courses of action. These are listed underneath the main list of steps and reference the point in the original steps at which the alternative course may become apparent. They list the condition that will trigger the alternative steps, the action taken in this eventuality, and the point in the original list to return to.

Developing the use cases intuitively encourages initial thoughts about the design of the new interface. Considering how a user is going to interact with the new interface naturally involves considering the best interaction methods for the various user tasks involved in the system. In this respect the development of use cases provides an extremely useful tool as a starting point to both the development of requirements and the eventual design of the interface. However, it also poses some potential problems as the majority of the design work has not yet been completed and therefore some of the interactions need to be ambiguous enough to avoid contradicting the final design ideas. Very specific use cases will either limit the design process, or will require re-development of the use cases at a later date.

To ensure this did not become an issue, specific features of the interface were only included where necessary, and where there was a limited number of design alternatives available. For example, Use Case 8 models the process of a user running an experiment. By simply stating that the user selects the ‘Experiment’ option, the use case is generic enough not to limit the design process. If the defined step had been to select the ‘Experiment’ option from the ‘Tools’ menu, this would have meant the design would be forced down this route, or the use case would have had to have been re-engineered at a later stage.
5.3.2 Use case analysis

Development of the use cases provided an excellent starting point in the generation of requirements, especially those relating to how the user interacts with the system.

A key feature of the new interface highlighted by the use case development was the method in which filters and algorithms are selected. Evaluation of the current interface had identified this area of the system as a potential problem, as all information about an algorithm or filter was only available after it had been selected, and even then was not easy to find. During the development of use cases for applying preprocessing filters and algorithms to a data set it became obvious that displaying descriptions of the algorithms whilst the user is choosing the one they want would be the best solution. Giving the user the option to change algorithm parameters immediately after choosing the algorithm they want also seemed to be a more natural step than the current process of having to click on the algorithm name to access this information.

This example also highlights how useful the use cases proved to be in the development of initial design ideas. Having created use cases for the steps a user will have to take to complete tasks such as these, the most effective interaction processes become more apparent, and this in turn makes the design process less effort.

5.4 Key Requirements

We will now discuss some of the requirements that have been identified as key to the design of the new interface and why these requirements were generated.

5.4.1 Functional requirements

As noted above, a key issue with the existing interface is the fact that descriptions of the filters and algorithms are not available until after a user has selected them. This led to the generation of functional requirements 2.1 and 3.1, stating that descriptions of filters and algorithms should be available to users when they are selecting them. This should allow them to make an informed decision on the algorithm they wish to choose.

Requirement 1 states that the new interface should combine the functionality of the Explorer and Experimenter interfaces in the current system to one single interface. This requirement is an important one, as it means that users will not have to switch between interfaces to perform different tasks, and should standardise the processes more, making using the interface far simpler and more user-friendly.

A key problem area identified by users was the inability to recover from mistakes made within the system. In order to counteract this problem, functional requirement 9.1 identifies the need for ‘Undo’ and ‘Redo’ functionality to be available to users. Functionality of this kind is regularly available in many commonly used programs such as the Microsoft Office suite of programs.

Another solution to the problem of being unable to rectify mistakes is to use a ‘wizard’ structure that allows users to move forward and back through the steps of a process, as identified in requirement 9.2. It was decided that this would be an important addition to the new interface to model the order in which users make use of the standard Weka
functions, and could prove especially helpful to new users who are unsure of what they are doing. This should be an improvement over the current interface, where users select different tabs to carry out different tasks, as this does not give a great deal of assistance in terms of the order in which tasks can be most effectively carried out.

One of the features lacking in the current interface that was identified very early on in the project was the inconvenient method required for users to incorporate their own code modules into the Weka system. Functional requirement 10 identifies the need to rectify this problem, and to make it as simple as possible for the user to perform this task. However, before the design stages of the development have been undertaken, it is difficult to identify exact requirements for how this problem should be solved.

## 5.4.2 Non-functional requirements

Key usability issues are addressed in the non-functional requirements. Requirement 4.1 specifies the need for the inclusion of ‘Tool Tip’ help to each of the components, which will assist the user in finding their way around the system.

Requirement 5 identifies the need to use common interface elements that users should intuitively know how to use. A great many computer programs use menus such as ‘File’ and ‘Edit’ to access common tasks, and new users especially are likely to benefit from components such as these being included in the Weka interface.

It will be important to use common data mining terminology in the interface as noted in non-functional requirement 6.1. If terms the users are not familiar with are used, the interface will not seem as natural and this will impact on the usability.

Non-functional requirement 7 identifies the need to ensure the new interface can be easily installed and incorporated into the Weka system. This is especially important within the scope of this project, as user responses will be needed to evaluate the new interface. As users’ time is already limited, it is unlikely they will bother to test the new interface if it is complicated to install.

## 5.5 Conclusions

Despite the difficulties identified due to the inability to communicate with Weka users, a relatively comprehensive set of requirements has been developed. The evaluation of the current interface provided a lot of the key requirements identified to ensure that the new interface does not suffer from the same usability issues apparent in the current one. The development of use cases also provided a clearer view of how the user will interact with the system and helped develop the requirements further.

The requirements generated will be vital in the design process. They will provide the basis from which the designs are built on, and can be consulted to assist in design decisions to ensure that the correct choice is made.

The requirements will also play a key role later on in the project as part of the validation and testing of the new interface. Comparing the implemented version with the requirements will identify any areas that do not meet the requirements and therefore will need to be reconsidered.
6. Design

6.1 Introduction

The design of any user interface is a key factor in determining the overall usability of the system. If the design that is implemented is flawed, it can be very difficult to rectify this problem at a later stage. Minor design issues can usually be resolved during the implementation stage, but the code is likely to be structured towards the design and so major changes would require a great deal of extra effort to rework.

The design process does not only include the design of how the interface will look and how the elements will function, it also includes the design of the structure of the implementation and helps define how the process of building the interface should be carried out.

6.1.1 Design Process

Initial design ideas were developed from the requirements and use cases. Certain aspects of the interface were defined by the requirements, and so ideas about the rest of the interface were built around these required components.

Interface designs were hand drawn to begin deciding on the layout of the interface. Initial designs were very rough and were used to decide on the general layout. As the designs were developed, they were made neater and more detail was added until the screens contained the vast majority of the detail that would be in the final program. An advantage of producing less neat drawings early on is that the designer will be more willing to look at them critically. If neat drawings that take a long time to complete are produced early on, the designer may feel less willing to change something they are proud of and usability problems may not be eliminated.

Following each design iteration, the designs were evaluated to ensure they did not pose significant usability problems. The cognitive walkthrough technique used in the evaluation of the current interface (see section 4.4 Evaluation Without Users) was used to assess the interface and uncover any particular areas that may not seem intuitive to the user. Whilst this technique will never be as effective as having actual users to evaluate the system, it still proved useful in giving a relatively quick evaluation of designs, and does go some way to helping assess the interface from the point of view of the user.

The designs were also compared to the requirements document to ensure that all the necessary requirements had been met. This was a vital step in validating the design and ensuring it met all the needed standards.

In the design of a user interface, it can become difficult to imagine how intuitive a particular function will be without actually having a working version. Whilst the designs were developed as much as possible, it was also important that a working version of the interface was produced as soon as possible to make it clearer how effective the designs would prove to be.

Once the general structure of the design had been decided upon, prototype screens were created. These placed the various buttons and menus in the correct place on the screen,
but without implementing the underlying functionality. This allowed simple cognitive
walkthrough testing to be performed to identify any usability problems that existed, so
that the design could be updated to eliminate them.

An examination of the underlying Weka code was also required during the design
process to discover how the new interface would link into the rest of the Weka
functionality. This could potentially have a bearing on how the interface is designed, and
would also highlight any areas of functionality that may limit the design in some way.

The first elements to be examined were the source code files for the existing user
interface. The process of studying the methods that are called when the buttons on the
interface are clicked gave an insight into how the existing interface links into the
algorithm functionality. It also helped to establish which information on-screen is
created by the algorithms themselves, and which is part of the interface. As some of the
methods are likely to be re-used in the new interface, the examination was also useful in
giving an insight into how they work.

Having established the basic ways in which the existing interface connects to the Weka
functionality, it was also necessary to examine this functionality in some detail. The
required input and output of these functions could affect how the new interface is
designed, especially the format of the results output by algorithms.

6.2 Final Design

6.2.1 Overall Design

To ensure the new interface does not suffer the same problems as the current interface,
some significant changes to the overall structure have been made in the new design. We
will now discuss these changes, along with the justification for the decisions made. This
will be followed by a more in-depth look at the key elements of the new designs.

The current interface is quite disjoint, with each panel designed slightly differently, and
a completely separate interface for the experimenting functions. The new design aims to
unify the various functions by providing a main screen from which all other functions
are accessed. This screen includes a menu bar and toolbar, two user interface elements
that are used in many programs.

The new interface introduces the idea of ‘projects’. This is the term to be used to
describe each task the user has performed. Each data set opened constitutes a new
project, as does each new algorithm run. This concept is used to help users keep track of
the tasks they have performed, with each project stored in the project list in the main
window (See section 6.2.2 Main Screen below). The aim of this feature is to allow the
user to compare the results of the tasks they have performed and to help manage which
information they wish to keep.

This should be a fairly natural addition to the interface. The current interface uses
‘Result Lists’ on each of the panels to list the results of each of the algorithms the user
has run. The new project list will be a very similar concept, but as there are no separate
tabs for each algorithm type, they will all be placed in the same list, with the name
reflecting the type of result.
The only element that will be added to the project list that was not included in the original results lists will be data sets. However, when a user opens a data set it will be added immediately to the list, and therefore this should be a fairly natural progression for the user. Without a full evaluation, it is difficult to say how effective this concept will be in practice, although initial testing in prototypes seemed to suggest that it would be a fairly intuitive process.

The main processes in Weka will often be used in a similar order. For example, the first step will always be to open a data set, next a user is likely to want to filter the data and select the most appropriate attributes, and then finally run an algorithm on the data set. This process ordering was modelled to a certain extent in the Knowledge Flow interface, but was not a particularly successful representation as it was highly unintuitive.

In order to model this ‘step-by-step’ idea, the new design will provide a ‘Wizard’ structure to guide users through the common tasks. A wizard is a common user interface interaction tool that aims to assist the user in completing a particular task by taking them through the process one step at a time, and is usually controlled using ‘Next’ and ‘Back’ buttons at the bottom of the wizard dialog window. The wizard function should be particularly helpful for inexperienced users, as it will explain what is happening at each step. Smaller wizards will also be used for the main processes in Weka, effectively allowing the user to jump in and out of the appropriate points (see section 6.2.4 Wizard).

Although a wizard structure will mean a user will usually have to make more mouse clicks to complete a task, this will be outweighed by the fact that there is likely to be less information displayed on screen at one time. This was a particular problem in the existing interface, with the majority of information squeezed into just a few screens.

The main wizard will be primarily aimed at less experienced Weka users, who are more likely to require assistance in deciding which tasks they wish to perform. As they are taken through the tasks each step will be explained to them, and this will hopefully increase the speed at which they learn the Weka basics. More experienced users are likely to find accessing the main Weka functions through the menu bar more efficient, as this will involve them having to go through less steps. They are more likely to be confident about the task they wish to perform next, and so will be able to quickly select it from the menu.

To ensure the new interface does not feel completely alien to users, certain elements of the existing interface will be used in the new version. Often these elements do not have any usability problems associated with them, but have not been used in the most effective way in the existing interface. This also has the added bonus that modules of code from the existing system can be reused, saving time.

### 6.2.2 Main Screen

The main screen will be the point from which users control the data they are using and can access all the functions of Weka. The main screen will feature two sections:

- The left hand side (taking up roughly a quarter of the screen width) will contain the ‘Project List’ and project details.
- The right hand side of the screen will contain different methods for viewing information about the current project.
This layout has a number of similarities to the tabs in the current interface that allow users to run the algorithms. On the left hand side of these tabs the results list is displayed, and the right hand side contains the text panel used to display the algorithm results (See Figure 6.1). Selecting a result from the results list updates the information displayed in the text area so that it is relevant to the selected result. In the new main screen, selecting a project from the project list will update the information displayed on the right hand side of the screen.

![Figure 6.1 – Elements of current interface to be re-used](image)

The algorithm tabs were the most consistent elements of the current interface, with the layout of all four being generally similar. By making the main screen of the new interface similar to these, it should mean that the new interface does not feel completely alien to existing users.

At the same time the screen will also feature common user interface elements such as a menu bar and toolbar, and so new users should also feel fairly comfortable using the interface.

The project list is a new addition to the interface that will let users compare the outcomes of the different actions they take. When they open a data set it will be added to the list, and when they apply an algorithm the result will be added to the list. This provides the ability to have more than one data set open at a time, allowing users to compare the performance of algorithms on different data sets. It also allows users to easily keep track of the actions they have performed.

Selecting a project from the list will display the information related to that project. If the project was one created by running an algorithm on a data set the text output will be available to view, as well as any available visualization options.

The functions that can be applied to a data set will be selected from the menu bar. They will be applied to the data set of the currently selected project. This should be a natural process as users are likely to look at the details of a data set before wishing to perform an action on it.
Below the project list will be displayed details of the project such as the data set name. This will help users keep track of the project they are looking at.

The right hand side of the screen will feature a tab system to allow users to view different information about the currently selected project. A tab area will easily allow users to switch between views, and users of the existing interface should feel comfortable with it due to the use of tabs in the current interface. Three tabs will be used for the different information output by the software. These are ‘Data set details’, ‘Text’ and ‘Visualization’.

In the current interface the user must select different tabs to perform different functions, which could become inconvenient and sometimes confusing. In the new design all the functionality will be accessible directly from the main screen, and the tabbed area will only control the type of information the user is currently viewing, so the same problems should not occur. As each of these tabs shows a completely different view of the data, the user is unlikely to ever want to view them at the same time, so this should not be an issue either.

The data set details tab will be the tab shown initially, and will feature a number of the elements used in the current interface on the Preprocessing panel. The table listing the attributes of the data set, another table displaying details of the currently selected attribute, and a histogram displaying data set information will be included in this panel. This will allow users to explore the data set after they have opened it and will also allow them to remove attributes they immediately know they don’t want.

The text tab will display the text output from algorithms after they have been run. One of the issues identified in the existing interface is that the layout of the output text could sometimes be unclear. Some of the elements, such as the confusion matrix of a classifier, would be better off displayed in a format other than the standard text, such as in a table. Unfortunately, the examination of the source code showed that this output text is generated by the underlying Weka functionality. For example, the confusion matrix is generated as a string by calling the `toMatrixString()` method, which would not allow the format of it to be changed. As this method is located in the underlying Weka functionality, it will not be possible to alter this within the scope and time constraints of this project, and so this may be a potential limitation to the improvements made by the new interface.

The visualize tab will allow users to view plots featuring the data set data. Users will be able to choose the type of visualization they wish to view from the ‘Visualize’ menu on the menu bar. Initial options available will be viewing a plot comparing two attributes, or a matrix of plots comparing all attributes. Once a classification or clustering algorithm has been run, extra visualizations options will become available.

Initially, the visualize tab will show the plot used to compare two attributes of the data set, as this was deemed to be the most useful of the initially available plots. If a user is switching between projects, the visualize tab will continue to keep the same visualization option but will be updated with the relevant object information. This is important to ensure consistency within the interface, and will allow the user to compare similar plots for different projects. This will not be possible if a user is viewing a visualization option such as a Margin Curve that is specific to a classifier result, and then selects a project that does not contain a classifier result. In this situation, the visualize panel will return to the original plot between two attributes.
A button will allow users to view the contents of each of these tabs in a new window to ensure they are able to see all the information they require.

6.2.3 Menus

The main screen features a menu bar and toolbar. These are important features as they are two of the most common ways of interacting with an interface. Users are likely to immediately recognize the names of some of the menus such as ‘File’ and ‘Edit’, and will immediately have an expectation of the functions that will be contained in them. This should help make users feel more comfortable using the interface with the knowledge that they are already able to access certain common features.

To ensure the new interface takes advantage of these benefits, the new design will contain five menus that are standard to a large number of Windows based programs:

- **File** – includes functions for manipulation of data sets such as open and save.
- **Edit** – includes controls for making changes to the data.
- **View** – controls the view currently displayed.
- **Tools** – allows the user to access the functions that are available in Weka.
- **Help** – provides help to the user.

One additional menu that is included in the design is the ‘Visualize’ menu. This menu will be used to control the visual representation of the data that is displayed in the visualize tab.

To improve the speed with which users can access commands in the menus, accelerator keys will be used to call common features. For many of these, standard combinations that are used by convention in a great many programs will be used, for example the combination ‘Ctrl + Z’ to undo the last command.

However, not all the decisions were so obvious. For example, ‘Ctrl + S’ would conventionally be used for the ‘Save’ command. However, the design includes a command to save the current data set as well as one to save the text output. As the combination can only be used for one of the two commands, the decision was made to assign it to the ‘Save Output Text’ command, as this combination is commonly used in Word Processing software to save the current text file. Similarly, the ‘Ctrl + O’ combination is regularly used to open files. In Weka however, there are three open commands (‘Open File’, ‘Open URL’ and ‘Open Database’). In this situation the combination will be assigned to the ‘Open File’ command, as this will be the one most regularly used.

The commands available in the tools menu allowing the users to apply algorithms to the data also posed problems. The ‘natural’ combinations could not be used as ‘Classify’ and ‘Cluster’ would both require the combination ‘Ctrl + C’, which is conventionally assigned to the ‘Copy’ command. It was therefore decided to assign the combinations using letters across the middle row of a standard ‘Qwerty’ keyboard, which do not have any commands conventionally associated with them. By placing them all next to each other, users should become used to the slightly unnatural assignments more rapidly. Conveniently, the ‘F’ key can be easily used to call the first tool, ‘Apply Filter’. The remaining tools will be assigned the keys to the right of this.
The toolbar is also a common feature to many programs. By convention this is displayed just below the menu bar, and uses pictorial icons to represent the available options. The tools usually offered on the toolbar are the ones that are likely to be used on a regular basis, so that users have easy access to them, rather than having to select them from a menu.

### 6.2.4 Wizard

The wizard functionality in the new design will allow users to move through the processes of Weka in a simple step-by-step manner. One of the problems highlighted in the evaluation by users of the current interface was the difficulty in recovering from mistakes they have made. An added benefit of a wizard structure is the ability to go back and forward steps to make changes to decisions users have made previously.

The design of the wizard began with the construction of a tree structure to define the available paths through the wizard structure. This was developed by considering the different tasks a user can complete, and the points within the structure at which they can potentially choose to complete them.

The first stage of the wizard will be to open a data set, as no other tasks can be completed until the user has selected a data set to work on. In Weka, it is possible to open data sets from three sources; files, URLs and databases. Each of the three requires different options to specify the data set to open, and so it was important to consider the most appropriate design to allow users to do this.

Two design options were considered for this process. The first was to have an initial screen allowing the user to choose the type of data set they wished to open and then individual screens for each of the three types allowing the user to select the relevant options. The second option was to display all of the options on one screen, with the relevant ones becoming enabled depending on the radio button choice made by the user. The final decision was to choose the first option. Although there were not a large number of options for the three choices, it was felt that having all the information on one screen could become over complicated and confusing, and so despite the requirement for the user to make an extra mouse click in the first option, this was deemed the more user-friendly.

After opening a data set, the next step in the wizard will allow the user to choose which task they wish to perform next. They will be able to choose between the main Weka functions of:

- View data set details
- Select attributes
- Apply a filter
- Apply an algorithm
- Visualize data

Having selected their desired option, the user will be taken to the relevant screen according to their choice. Once they have completed their chosen task, they will be returned to the task selection screen to choose another task should they wish to do so. Alternatively, they will be able to select the ‘Finish’ button to exit the wizard.
One of the key requirements identified was to display descriptions of the algorithms while the user is selecting the one they wish to use, and then to allow them to set the parameters of the chosen algorithm. The wizard structure is able to fulfil this requirement without difficulty. Having chosen to apply an algorithm to the data set, the user will be presented with a list of available options on the left hand side of the screen, and a description of the currently selected algorithm on the right hand side. Having selected the one they wish to use, they will be able to use the standard wizard ‘Next’ button to take them to the next screen that will display the list of available parameters for them to set.

To offer additional assistance to less experienced users, an information panel has been included in many of the wizard screens. This will display additional information about the options the user has the choice between.

In addition to the main wizard that takes the users through the different options they wish to use, smaller versions of the wizard will be used when tools are selected from the main menu. When the user selects a tool they wish to use, they will effectively ‘jump into’ the main wizard at the relevant point, and will leave it again once they have completed their desired task. Once a task has been completed and the user leaves the wizard, the results will be added to the project list.

On occasion, the ‘Next’ button may need to be disabled until the user has performed a particular action on the screen. In other situations, clicking the ‘Next’ button may result in an error message being displayed is the user has made an erroneous selection. Although having these two different situations will make the interface slightly less consistent, it was felt that this was the best design for the two potential decisions. If the ‘Next’ button is disabled in certain situations it may not be obvious why and so an error message will be necessary. However, having too many error messages can become frustrating for the user, so in situations where an obvious choice must be made before progress can be made, the ‘Next’ button will be disabled.

One of the requirements is to inform the user of when the system is busy processing a task. This will need to be done within the wizard structure, as it is from here that the users run algorithms on the data sets. In the current interface, an animated bird in the bottom right hand corner of the screen signifies when the system is busy. However, this was not in a position that is obvious to the user, and they may not notice it. To help keep some familiarity for users of the existing interface, the bird will again be used, but it will be placed in a more obvious position, to the left of the wizard control buttons, and will be highlighted more clearly. In the creation of prototypes, the effectiveness of this positioning will be evaluated.

### 6.2.5 Experiments

One of the aims of the new interface is to integrate the experiment functions into the same interface as the rest of the Weka functionality. To keep the interaction methods throughout the design consistent, the experiment functions will also be accessed through a wizard structure.

In many cases, users are likely to want to perform experiments using data sets and algorithms they have already explored. As the current interface completely separates the explorer and experimenter interfaces, there is no easy way for the user to experiment on
the data they have just been using. This is an issue that the new design will aim to rectify.

When the experimenter wizard is started, the user will be asked to choose the algorithms and data sets they wish to use in their experiment. Here, options will be available to allow them to import both data sets and algorithms they currently have stored in their project list. This should allow users to easily run experiments using the information they have already selected, and will save them having to set the parameters of their chosen algorithms twice.

6.3 Conclusions

The designs created were developed from the requirements, and so should meet all the aims that have been asked of the new interface. Weka offers a great deal of functionality, but the key requirement of incorporating it all into a single interface has been achieved, whilst eliminating the usability issues that the current interface suffers from. One potential limitation is that the text output from algorithm results has not been redesigned. However, as it would not be possible to implement this design change due to the structure of the Weka code, it was not deemed necessary within the scope of the project.

By evaluating the designs following each iteration, usability errors could be discovered and eliminated, meaning that the designs were continuously improving. The cognitive walkthrough technique used proved to be particularly useful as it assesses the design from the point of view of the user, helping identify problems that may not have otherwise been obvious to the designer.

The designs have been developed so that both experienced Weka users and new users to the system should feel comfortable. Elements of the existing interface coupled with quick access to all the functionality from the main screen will appeal to existing Weka users, while new users should feel comfortable with the use of conventional interface elements such as menus, and the wizard to guide them through the common features. This is a key point, as if all users feel comfortable using the new interface, it can be considered a success.

Following the generation of simple prototypes, the positioning of the animated bird that indicates to the user when the system is busy in the bottom left hand corner of the wizard structure seems to be a sensible decision. As the wizard windows are smaller and contain less information, the bird image is far more obvious than it was in the old interface, and so should prove more effective. However, until the interface has been by users it will be difficult to tell whether this is actually the case.

The design took into consideration an examination of the Weka source code, and this, along with the initial prototypes built, will provide a good platform from which to begin the implementation stage of the project.
7. Implementation

7.1 Introduction

Careful planning is often key to implementing a large project such as this. Before work on the coding began, it was important to ensure a strategy was in place to shape the order of implementation and the methods used.

It was also important to consider the structure of the code and how the functions worked during the design process to ensure that the interface designs could be successfully integrated with the existing code.

As part of the planning process, an examination of the source code for the existing interface, and for the background functions of Weka was carried out. This helped give an insight into how the code was structured, and how many of the key functions would be implemented.

7.1.1 Tools used

Weka is written completely in Java, and therefore there was no great consideration needed to decide that the new interface also needed to be implemented in Java. Java is an object-oriented language, and the Java Platform offers a large number of predefined classes that most programs written in Java will rely on for a great deal of their basic functionality (Flanagan, 2002). Two such packages used significantly in this project are the javax.swing and java.awt packages, both of which offer key user interface functionality.

The vast majority of the implementation work was carried out in Borland JBuilder. This piece of software is an integrated development environment for the Java language, and was particularly chosen because of its interface design features, that allow components to be ‘dragged and dropped’ onto the user interface. This is a feature that is particularly helpful during the creation of complex interfaces, as implementing these simply through code can become very time consuming, especially during the testing and fine-tuning stages.

7.1.2 Development

Development work began with basic prototypes of how the interface would work. These were connected to a few of the simple Weka functions to test how the interface would interact with the background functionality. However, most of the prototypes had little functionality, and were simply used to assist in the final stages of the design process, to refine the designs and help identify potential usability problems.

Once the designs reached their final stages, the implementation stage was given the greatest priority and the development on the full interface began. The first sections to be implemented were the basic structure of the main window and the wizard framework. These were both areas that would operate as the basis for the vast majority of the functionality of the interface, so it was vital to ensure these were correctly implemented before the more complex functions were inserted.
In general, the development was completed in stages, with a “module” of the interface developed during each stage. One module usually encompassed a group of simple components, or a single more complex one.

7.1.3 Testing

As the time available to complete this project was limited, the decision was made to generate the testing plan “on-the-fly” as development of the software progressed, rather than creating a full test plan during the design phase. This approach did involve some risk, as there was a danger that certain features may not be tested, but it was felt that careful planning would minimize the risk, and the advantages of this approach would make it worthwhile. As this project focuses on implementing a new interface, the tests will be based around how the interface interacts with the underlying functionality, rather than testing underlying functionality itself, which will have already been thoroughly tested during the development of the released Weka software. This means that the elements to test will be visible on the interface, and will make generation of tests far simpler, unlike the development of a whole software program, where a great deal of the tests will be hidden in the underneath the surface.

This approach should save a great deal of time, as generating all the test cases from just the designs will mean going through every single screen very thoroughly to ensure each function has all the necessary tests. Instead, with each module of code developed (usually a particular element or small group of elements on the screen), the test cases can be easily generated based on the functions involved. However, it was still vital that some planning of the tests to be performed on an interface element was performed beforehand to ensure the correct tests were applied.

The majority of testing employed is “black box” testing. This process involves a series of test cases that state the action performed to initiate the test, and the expected output of the test in a situation where the system functions correctly. It also includes tests to ensure that the interface reacts in the correct way in situations when the user makes an erroneous selection. For most of the elements in the interface, generating these test cases should be a relatively simple procedure. For example, the test for the ‘Open File’ button would be to click the button, and the expected output would be an ‘Open’ dialog window being displayed. For some of the more complex elements of the interface it will become a little more time-consuming, but in general this process should be very efficient.

As well as the black box tests, certain elements required additional testing. Elements of the interface that allow values to be input need to be tested for “extreme values”. This involves inputting very high values that are just above and just below the threshold the program should accept, and the same for very low values, to ensure that the interface reacts in the correct manner.

Once these functional tests had been completed, any errors found will be fixed and then retested. The basic functionality of the module can then be signed off as complete. Integration testing can then be employed to ensure that the module interacts correctly with the other elements of the interface.

By applying all these tests to each module of the interface, we can be fairly confident that all elements of the interface will fully tested.
Further time should be saved thanks to the JBuilder software. This allows the syntax of code to be checked as it is written, a feature that should potentially save a great deal of time that might otherwise be spent running the code to eliminate basic syntax errors.

Backups will be taken at regular stages to make sure that in the event of any significant problem, only a limited amount of work will be lost. Each backup version will be given a version number, and a record of the main changes made in each version will be stored. Using software designed to implement source code control was deemed unnecessary in this project, as only one developer would ever be working on the code, and so there was no requirement to have a central repository of code. Simply creating backup versions after major changes was considered an effective enough in this situation.

7.2 Implementation Issues

During the implementation process, there were a number of decisions to be made about the methods and structure of the implementation, along with a number of problems that were encountered. The following section gives an overview of the key decisions made and solutions to the major problems.

Screenshots of the key screens in the implemented designs can be found in Appendix A.

7.2.1 Main Screen

The main screen was a very important part of the implementation, as it effectively links the various parts of the interface together.

The project list was implemented using the existing Weka ResultsHistoryPanel class used to list the results of algorithms in the current interface. However, additional methods needed to be implemented to add objects to this list. When a project is added, the panels in the tabbed area are updated with the project details, and the project just added is selected in the project list.

Whenever a new project is added, its corresponding data set is added to an array. When the user selects a project from the project list, the current data set needs to be updated from the array according to the project selected. As projects can also be deleted, the array needed to be updated to take account of this.

The right hand side of the screen, containing the tabbed area that provides the different views of the data, was implemented using a simple JTabbedPane. As the size of the panels in this area could become limited, and the information often benefited from being viewed in an as large an area as possible, a ‘View in a new window’ option was included. This created a new frame with the contents of the currently selected tab in it. As the data set details tab allows attributes to be removed from the data set, the two windows needed to be synchronized. To perform this task, an additional function to update the contents of the tab with the new data when the new window is closed was added.

To allow the text output from the results of algorithms to be printed, the panel that displays the text output needed to implement the Java Printable interface. As the
results from certain algorithms can become quite long, the number of pages that it was to take up is calculated based on the size of the text output compared to the size of a printed page. Following tests of this initial implementation, it was discovered that for multiple page printouts resulted in the calculated number of pages being printed with the same first page content on each. To rectify this problem, the Graphics object used to define what is printed needed to be translated the distance of a page down the output text for each page.

7.2.2 Wizard Structure

One of the initial implementation decisions that needed to be made was selecting the structure for the wizard part of the interface. Implementing the code to perform this functionality from scratch could prove time consuming, as there were many factors that need to be considered.

The wizard needed to allow users to move through the screens in order, often branching off in different directions depending on the choices made. A user also needed to be able to move back to the previous screen when required, which should show the same options selected as when they finished with it originally. Information, such as the current data set, would also need to be passed on from one screen to another, and updated when filtered.

As a wizard is a fairly common user interaction tool, it seemed likely that packages implementing this functionality would have been created already. After some research, a number were found, with one in particular, from the Sun Java Wizard tutorial (2006), seeming the most appropriate for the needs of this system.

A choice needed to be made between using the existing wizard framework package and creating a completely new wizard framework. Although creating a new wizard structure would mean that it could be tailored exactly to the requirements of the system, it was decided that the extra effort required would not be worthwhile for this benefit, and the existing package could be modified sufficiently to suit the needs of the new interface.

The basic wizard frame in the implementation is a JDialog box, featuring the control buttons in the bottom right hand corner. The main body of the window contains a cardlayout structure, into which all the wizard screens are placed. When the ‘Next’ and ‘Back’ buttons are clicked, the relevant panel in the card layout is activated as the top card for the user to see.

Each screen in the wizard requires two classes; a panel containing the actual content of the screen, and a descriptor class that extends the WizardPanelDescriptor class included in the wizard package. The descriptor class is used to control how the panel interacts with the other panels in the wizard, for example by specifying the next and previous panels, and can also be used to call methods in the panel when it is about to be displayed. This is an important feature as it allows information such as the current data set to be added to the panel dynamically, as it may not be available when the panel is actually created at the wizard instantiation.
The **WizardPanelDescriptor** class offers a selection of abstract methods to allow information to be passed to the panel at different stages and to control the user's path through the wizard dynamically:

- `aboutToDisplayPanel()` is called just before the panel is displayed to the user.
- `displayingPanel()` is called just after the panel is set as the currently displayed panel.
- `getNextPanelDescriptor()` is called by the wizard controller to retrieve the reference to the next panel to display.
- `getBackPanelDescriptor()` is called by the wizard controller to retrieve the reference to the panel to display when the ‘Back’ button is clicked.

A number of issues with the wizard structure were encountered during its implementation.

For certain wizard panels, such as the choice of filter or experiment analysis screens, the ‘Next’ button needed to be disabled until the user had made a certain selection. The ideal way to implement this was to add a `PropertyChangeListener` function to the object that the required decision was based around, and then to check whether a valid choice had been made each time a `PropertyEvent` was generated. Unfortunately this caused conflict with the wizard controller class that also uses the fired `PropertyEvent` object to set up the next panel in the wizard structure. This feature is integral to the functionality of the wizard and so could not be changed to respond at only certain points in the system. In the event of the filter selection panel, this could be resolved using a `TreeSelectionListener` instead, but in the case of the experiment analysis panel, it was necessary to carefully consider each potential situation that could occur and ensure that the buttons were correctly set.

Certain screens required some simple error checking. For example, when a user chooses to run an experiment wizard, the first panel loaded requires them to select at least one algorithm. If no algorithms have been added to the algorithms list when the ‘Next’ button is clicked an error message should be displayed. However, this becomes a more complex problem than it initially seems, as the ‘Next’ button component, and the method called when it is clicked, are located in the main Wizard class, which is not accessible to the individual screen.

To solve this problem, a method was added to the panel that checked whether or not the list was empty. This was then called from the panel descriptor class when the wizard controller tries to obtain the panel that is to be displayed next. However, this presented a new problem, as the method to retrieve the next panel is called when the panel is about to be displayed, and at this point the list will still be empty so an error message was displayed before the user is able to do anything.

This issue was resolved by setting a boolean value within the panel descriptor class. This boolean value was only set to be true once the panel had been displayed, and the method to check whether or not the list was empty was only called when this boolean value was true. This ensured that an error message was only displayed when the user clicked ‘Next’ with an empty list.

This issue was still not fully resolved however, as the error message did not prevent the next panel being displayed. This required a change to be made to the wizard controller
class, allowing a next panel of ‘null’ to be returned, with the panel not changing in this case. This allowed the next panel to be set as the next screen of the experiment wizard in the event of no errors being encountered, and ‘null’ in the case where an error was encountered.

There are also a few more special cases that required slightly different situations to the majority of panels. One example is the ‘main choice’ panel in the main wizard where users are asked to choose what the next task they wish to perform is. The user selects the choice they want and then clicks ‘Next’. Each choice opens a new wizard to perform the task, but it was not possible to load this from the ‘Next’ button. Instead a new panel had to be added with no visible content that loads the wizard and then immediately returns to the main choice panel.

Similarly, at the end of the task wizards, the result needs to be added into the project list when the user clicks the ‘Finish’ button. Here an additional panel with no visible content was used to perform the add action and then close the wizard down.

Overall, the decision to use the existing wizard structure appears to have been a good one. Although it required some modifications, it provided the functionality required to create an often complex structure, and allowed data to be passed between screens. It is difficult to create an estimate for how long implementing a completely new wizard structure would have taken, so it would seem that the use of the existing structure was a far more risk-free solution.

7.2.3 Algorithm Selection

The method for selecting an algorithm and specifying its parameters is performed as part of the wizard process. The first screen in this process allows the user to select the algorithm they wish to use from a tree structure. The structure of the tree is generated from a properties file that needs to be read in by the AlgorithmChooser class.

A description of the algorithm currently selected is displayed in a text area on the right hand side of the screen. This information is generated by the ‘global info’ that is stored in each algorithm class. Each time the user selects a new algorithm a TreeSelectionEvent is fired, allowing the information text to be updated dynamically.

After the user has selected the algorithm they wish to use and clicked the ‘Next’ button, they are taken to a screen allowing them to set the parameters of the algorithm. The properties of the algorithm are extracted from the algorithm class, and listed down the left hand side of the screen, with the appropriate interface component to allow the parameter to be specified. The right hand side displays help text about each of the options that is also extracted from the algorithm class.

7.2.4 Experiments

One of the additional design features of the experiment wizard was the ability to import data sets and algorithm settings from the project list. The first stage of this process was performed when the user begins an experiment wizard. The lookupDataSets() and lookupClassifiers() functions in the DataSetList and AlgorithmList classes
respectively go through the items in the project list and add any objects they find to the relevant arrays. If the array is not empty, the ‘Import from project list’ button is enabled.

When the user selects this option, a new frame is opened that displays a list of the names of each object in the array with a checkbox next to it. The user is then able to select the ones they wish to import, and they are added to the list. For the algorithms, this was a relatively simple process, as the addNewAlgorithm() method could be used to simply add the classifier object to the list. With data sets it was little more complicated.

The data set list used in the new interface is adapted from the list in the current interface, which only allows data sets from files to be added. The list is linked into the current experiment and data sets are added by calling the addElement() method in the Experiment class. Simply calling this method on the Instances object (used to manipulate data sets in the code) results in the individual attributes of the data set being added to the list, which was not the desired effect. To get around this problem, the data sets that have been selected for import are stored in temporary files, and these temporary files added. The only downside to this method is that each data set added must wait for the file save thread of the last one to finish before it can begin its own, which can make the import of a large number of data sets slow. It was important to communicate this fact to the user.

If a user chooses to import more than one data set, a message is displayed warning them that it may be slow, and checking that they wish to continue. If they do continue, the cursor is changed to an hourglass to indicate that the system is busy. Ideally, the progress of the import would be communicated to the user via a progress bar, but this was not possible due to the fact that the file saving functionality is performed within a thread that does not allow the interface to be updated whilst in progress.

### 7.2.5 Visualization

Although the new interface uses the same visualization panels as the old interface, the structure for accessing them has changed dramatically. The visualization options are displayed in the Visualize tab on the main screen. When the user selects a project from the project list, the setVisualization() method checks which visualization options are available for the selected project by reading its associated objects. The appropriate menu items are then enabled. When the user selects a visualization option, the appropriate object is loaded and the visualization plot is created and displayed.

It was important to ensure that, whenever possible, the selected visualization type is maintained when the user switches projects to allow them to compare the plots. This was relatively simple, but it was also necessary to ensure that if a particular project did not have the selected visualization option available, the panel would show the simple plot comparing two attributes. This was also handled by the setVisualization() method, by changing the selected visualization mode if no objects are found when the user selects a project.

### 7.2.6 Undo and Redo

The requirements for the new interface included the need for ‘Undo’ and ‘Redo’ options to allow the user to rectify any mistakes they make. Within the wizard portions of the
new interface, this functionality is covered by moving backwards and forwards through
the wizard screens, but for the main screen specific functions were needed.

The current interface includes some limited undo functionality on the Preprocessing
panel, allowing users to undo the effect of a filter on a data set. This functionality was
re-used in the new interface but required some additional code to allow users to undo the
other task that was otherwise irreversible on the main screen – the deletion of a project
from the project list.

The existing undo functionality stored old data sets as objects in an array to allow
multiple undo operations to changes to the data set. Objects are added to the array each
time a change is made to the data set. This functionality was adapted to also accept
objects of a new HistoryObject type that contained the required information to
reinstate a project in the project list. When the user selects the undo button, the undo() method checks the type of the last object stored in the array and makes the relevant
change to the data set or project list.

The redo function largely performs the reverse of the undo function. A second array
stores the objects that are added back into the system and takes them out again when the
user selects the ‘Redo’ option. By convention the redo button becomes available when
the undo operation is performed, and becomes unavailable again in two situations. The
first situation is when the redo operation has been called on each object stored in the
array, and there are no more actions to redo. The redo action also becomes unavailable
when the user performs another ‘regular’ task that does not involve undo or redo. In this
situation performing redo on the last action would be unintuitive, and this is the standard
procedure in most programs.

Ensuring the redo action becomes unavailable again after an action performed by the
user required a resetRedo() function that disabled the Redo button and clears the
array so that next time Redo becomes available the correct action will be taken. This
method is called from each of the methods activated by the user performing an on-screen
action.

### 7.2.7 Error checking

One area that was highlighted in the evaluation of the current interface by users was the
fact that error messages were not always displayed, and often users were allowed to
make selections that were not appropriate to their current situation.

Solving these issues caused particular difficulties in the implementation of the new
interface. The data mining algorithms available are often only applicable to certain data
types, such as numeric data or nominal data, and some are unable to deal with situations
such as certain values being missing from a data set. Unfortunately, the algorithms have
been implemented in such a way that it is not possible to check these factors until the
algorithm is running.

Ideally, each algorithm would contain a selection of methods such as
acceptsNumeric() that would return boolean values to define whether or not the
chosen algorithm could be run validly on the current data set. This would have made it
possible to display an error message as early as the algorithm selection stage informing
the user that they could not select a particular algorithm and giving them the reason why not.

Within the scope of this project it would not have been possible to add this kind of functionality to the (often very complicated) algorithm implementations. However, this could provide a useful future development to the Weka system, and would help in making it a great deal more user-friendly.

7.2.8 Import

A key feature of Weka is the ability for users to import their own java classes into the system. This means that users are able to add additional algorithms to the list that are not included in the original Weka package. However, using the existing interface, there is no easy way to do this. Currently, this process must be performed manually, and involves placing the class file in the correct directory, and editing a properties file to include an additional line so that the program finds the new algorithm. For inexperienced computer users, this will prove a daunting task, especially as the documentation available on this process is very limited.

Another issue with the inclusion of users’ own code modules is the fact that the installed version of Weka does not automatically include the required directories needed to perform this task. They, along with the properties file are included in a compressed ‘jar’ file in the Weka directory, which must be extracted in order to make the required changes.

This poses an important implementation decision to be made about the new interface. To make the import functionality as simple as possible, the directory structure would need to be in place in an uncompressed state. However, this would make the size of the files required to run the new interface a great deal larger, and this will become an issue when users are asked to download the new interface during the user-testing phase. To ensure as many users as possible participate in the user testing, the file to download will need to be as small as possible, and the users will also need to be reassured that the new interface files will not affect their current installation in any way. Including the required directory structure could result in files from the existing installation being overwritten, and this would immediately put users off from testing the new interface. Therefore the decision was made to provide the interface as a compressed file, but to also provide detailed instructions on how to extract the files should the user wish to use the import functionality.

The actual import function is called from the ‘File’ menu on the main screen. This displays a new ImportFrame object that allows the user to browse to the class file they wish to import. They must also select the package of the new class from two drop down menus. This information needs to be specified to allow the class file to be placed in the correct directory, and the correct change to be made to the properties file.

Once the user has selected the ‘Import’ option, the import dialog window was closed and the addImport() method of the main screen is called. This takes the file selected by the user and moves it to the directory of the package specified by the user using the drop down menus. It also reads in the properties file, and by comparing sub-strings, locates the point in the file at which the classes of the same type are listed. It writes the new
addition to the file at the correct point. If the file is successfully added, a message will be displayed informing the user of this fact. If not, an error message will be displayed.

7.3 Conclusions

The implementation of such a large project can be a difficult task due to the complexity of ensuring the different elements interact successfully with each other. A key challenge was linking the new interface to the underlying Weka code, and this was handled fairly successfully thanks to the examination of the source code during the design stages, and the re-use of certain elements from the existing interface.

The decision to use an existing package to implement the wizard component of the interface also appeared to be a successful choice, and has allowed descriptions of the algorithms to be displayed during algorithm selection by extracting the information from the Weka algorithm code.

The functionality to allow users to import their own code was one area of the interface that could potentially have caused some problems, but this has been successfully implemented. This automatic process should prove to be a great improvement on the current system, which requires the user to manually place the class file in the correct directory and change the properties file.

One limitation to the implementation is that the error checking was not as comprehensive as hoped. The algorithms only generate error messages when they are being run, and there is no easy way to check whether the settings a user has chosen are valid until this point. Rectifying this problem would have involved amending the algorithms themselves, a step which was deemed outside the scope of this project.

Thanks to the comprehensive design process, the new interface does seem to have eliminated many of the problems identified in the current interface. However, before a full evaluation of the new interface has been completed, it is not possible to say how successful the new interface will prove to be. This evaluation is likely also likely to result in a number of improvements that will need to be made to the implementation.
8. Evaluation of New Interface

8.1 Introduction

Having implemented the new interface, a full evaluation needs to be performed to analyse how easy it is to use, and to identify any significant usability issues that it suffers from. Changes can be then be made to the design in areas that are identified as problematic, and the interface re-implemented to eliminate these issues.

One key factor that will determine the success of the new interface will be how it compares to the old interface. Evaluations will be performed to directly compare the two interfaces, and this will give an indication as to how successful the project will prove to be.

There are many different methods of evaluating a user interface. Often a combination of these is likely to provide the fullest evaluation and will identify a greater proportion of usability problems than just a single evaluation method. Access to existing users of Weka will again be limited to contact through an Internet mailing list, and so the only real option is to ask them to fill in a usability questionnaire on the new interface.

In the evaluation of the current interface, cognitive walkthrough and heuristic evaluation techniques were used for assessment purposes, but this will not be the case in the evaluation of the new interface. These techniques have been used at regular stages throughout the development of the interface to highlight problems that were then fixed. They are therefore unlikely to expose many more usability issues.

Instead, usability tests observing novice users will be carried out. These will not only aim to give a fresh view of the evaluation process, they will also provide a more detailed analysis than given by the techniques mentioned above. The strength of these tests lies in comparing the performance of two interfaces, and for this reason were not used in the original evaluation of the current interface.

8.2 Evaluation by Users

8.2.1 The process

As with the analysis of the current interface, the most accurate insight into how effective the new interface is will be given by the people who actually have to use it on a regular basis. If they have problems with it they will be unlikely to want to use it in preference to the old interface and the aims of the new interface will not have been fulfilled.

To make the comparison of the two interfaces as fair as possible, the same questionnaire used to evaluate the current interface was used to evaluate the new interface. This questionnaire was the Computer System Usability Questionnaire developed by Lewis (1995) and implemented online by Perlman (No Year). As well as the set of questions, the questionnaire provides the user with the opportunity to add additional comments. Users were asked to include comments on how they felt the two interfaces compared.

The response to the request for users to evaluate the existing interface was slightly limited, and the fact that a download is required may discourage users from participating
in the evaluation of the new interface as well. Downloading the required files will take up time, and users may be worried about their computer being infected by a virus. However, it is hoped that the prospect of a new Weka interface will be enough to tempt some users.

Two files are used to run the interface on a users computer. The first is a ‘jar’ file containing the code for the program and new interface. Jar files allow a package of java classes to be compressed into a single, smaller file. The second file is a batch file to run the program. These files will be offered as a single download in a compressed ‘zip’ file. Making the download as small as possible is vital to encourage as many users as possible to evaluate the new interface, as they are likely to be put off by a large download, and would also be put off by having to download multiple files.

Although running the new interface has been made as simple as possible, it is still important to offer clear instructions on how to download and run the program. Many of the users may not be particularly experienced computer users, and so these instructions will be made as clear and as simple as possible. It will also be important to make it clear to users that this download will not affect their current installation of Weka in any way, as if they are worried the new interface will completely replace the existing interface they are highly unlikely to download it.

Another important decision that needed to be made was the amount of help in using the interface offered to the user. The aim of the new interface is to be intuitive and user-friendly, and so from this respect a measure of how effective the new interface proves to be could be how well a user copes with using it without having to constantly consult a detailed reference manual. The new interface contains elements of the existing interface that the users should be familiar with from their previous usage, and also incorporates common interface components that should be fairly self-explanatory to the user. However, providing help documentation is considered good practice in user interface design, and providing no help whatsoever could prove problematic for users if they become stuck. Therefore a brief overview of the main features of the interface has been included and can be accessed through the ‘Help’ menu on the main window.

8.2.2 The results

The full details of the user testing results can be found in Appendix D.

Unfortunately, the response to the request was disappointing, with only three users filling in the questionnaire, but this is perhaps not surprising given the need to first download the software and then thoroughly test it. This process could take up quite some time and will have been likely to put off users. However, the responses gained do give an idea of how users felt about the new interface, and the results are generally positive.

Notably, the questions referring to the ease of use of the interface, and whether or not the users liked using the interface of the system received excellent results, as did the questions relating to how easy it was to learn the system and how comfortable the users felt using the interface.

One user did give lower scores for a few of the questions, although it is thought that this is due to the wording of the questions. The questions that were awarded lower scores related to the functions available in the system, and the ability to complete the users work using the system. The user may have felt that these were limitations to the Weka
software itself, rather than the interface, as the scores given to questions directly concerning the interface were very favourable.

The responses to all the questions relating directly to the user interface were above average, and in the vast majority of questions the average score for the new interface was higher than that for the current interface. Notable examples of this were in the questions asking whether the system was easy to learn and simple to use. Although drawing definite conclusions is hard due to the limited number of sources, this does indicate that the new interface would be popular among Weka users.

In addition to the results of the questions, users are also able to leave comments, and to highlight the most positive and negative aspects of the interface. Whilst no comments directly comparing the two interfaces were made, there were a few comments made about the new interface.

One comment made was that ‘Select all’ and ‘Select none’ options should be available when including data sets and algorithms from the project list in an experiment. This feature would save the user having to select each individual check box when importing a significant number of items.

The Weka users also highlighted some areas of the interface they felt were particularly positive about the new interface. One user praised the wizard functionality, whilst another commented that having the description of the algorithms available during the selection stage was a useful addition.

8.3 Usability Tests

8.3.1 The Process

To directly compare the usability of the two interfaces, a series of usability tests were carried out. To perform these tests, a selection of individuals were observed while using both the existing interface and the new one. The time taken to carry out a series of tasks was measured, and by “thinking out loud”, the users were able to convey their opinions on the interfaces, and any particular problems they encountered. The participants in these tests were novices to Weka, and a survey of their experience in both computer use and data mining was taken before each test.

Careful preparation for these tests was needed to ensure that they were as fair and accurate as possible, and that the participants felt comfortable whilst performing the test. As these users are not domain experts, the testing process began by giving them a description of the basic concepts involved in data mining to read. Any points that they were unclear on were explained in further detail. At the beginning of each test, the user was given a short amount of time to explore the interface and familiarise themselves with it.

To make the evaluation fair, half the users were asked to assess the current interface before the new one, whilst the other half were asked to assess the new interface first. It is expected that during the evaluation of the interface tested first, the user will gain knowledge about how the system works. They are likely to learn the meaning of certain features, and will therefore be more prepared when they come to using the second interface. Results for the interface tested second are expected to be slightly better than
when the same interface is tested first, but by switching the order for half the tests, a fair comparison of the interfaces can be achieved.

The users were given a set of tasks to work through. The aim of these tasks is to see how easily they are able to use the major functions of Weka with both the current interface and the new one. Therefore, the tasks chosen for the evaluation will be the most common used in Weka, in order to achieve a fair representation of the effectiveness of the interface. The list of tasks can be found in Appendix D.

As the users are volunteers, they will only have a limited amount of time that they will wish to spend participating in the evaluations. For this reason, the list of tasks was kept reasonably short to ensure that the testing process does not take too long. As the application of classifying, clustering and association rules algorithms are similar processes, classifiers will be the only type used in this evaluation.

Users were observed as they worked through the tasks, with the observer instructing them when to move on to the next task (if not obvious to the user) and providing help if the user became completely stuck at any point. The observer timed how long it took the user to perform the tasks, and will made notes on any particular problems or comments the user had.

Ideally, the observation of the users during the tests would be performed by two individuals. The first would guide the user through the testing process and assist them when needed, while the second made notes. However, within the scope of this project it was not possible to have two, and so a single evaluator performed both functions. Simple short hand coding was used to ensure that the observer is able to keep up with writing down all the required information, whilst still assisting the user as necessary.

The time taken to complete a task or group of tasks can be a clear indication of how user-friendly an interface is. If the user is comfortable with the interface and does not encounter any major problems while using it they are likely to complete their tasks far more quickly. It is important to note that this may not always be the case. If the user completes the task quickly but is frustrated by certain aspects of the interface, they may prefer the slightly slower but ultimately more user-friendly interface.

Following their completion of the tasks, the users will also be asked some simple questions on what they thought of the interfaces, to give an overview of the general opinions on the interface.

The format of the results is based around the one suggested by Preece et al. (2002). For each of the tasks, a rating is given as to how easily the user was able to cope with completing it. The rating system uses a scale of 1-3 where:

- A score of 1 represents an easy completion of the given task.
- A score of 2 indicates the user found completing the task OK.
- A score of 3 indicates the user found it difficult to complete the task.
- A score highlighted in bold indicates that the user needed help in completing the task.

These ratings are placed in a table comparing each user with each task. Following the completion of the usability tests, average results can then be obtained for each individual
user, and for each individual task, to highlight areas of the system that need particular attention.

Comments made by the user, and other difficulties that the observer picked up on will also be considered in the analysis of the results.

The tests were carried out by six users. This was deemed an appropriate number, as the results of these tests will contain a lot of detail and will therefore provide a comprehensive evaluation of the system from the point of view of a new user.

8.3.2 The results

The results of the usability tests can be found in detail in Appendix D.

In general, the new interface performed very well during the usability tests. The times taken to complete the tasks were consistently shorter for the new interface compared to the current one, indicating a marked improvement in usability. On top of this, the scores assigned to how easy users found it to complete tasks were also consistently higher.

There were, however, a number of comments made by users, and observations made by the evaluator that revealed certain usability issues that still existed within the interface.

One of the problems particularly noted during the usability testing was the fact that the help text displayed at the top of each wizard screen was too small, as users very rarely noticed it. The font used is no bigger than that used for the rest of the interface, and the users eye is naturally drawn to the centre of the wizard window where the choices they have to make are located. This meant that users sometimes seemed slightly unsure of what to do, a point that could have been clarified by reading the text.

A feature highlighted by users as inconvenient was the inability to use a data set opened in the main window when they began using the wizard. Users who used the wizard tended to first open a data set from the main window and then begin the wizard. They were then forced to open another data set, and commented that it would be far easier if they had the option to use the data set already opened.

Whilst completing the usability tests, users praised certain aspects of the new interface that they felt were particularly useful.

As the users had little or no previous data mining experience, they did not know which algorithm to choose when asked to select one. Those evaluating the new interface second commented on the fact that having the descriptions of the algorithms on the algorithm selection screen made this task far easier for them.

Users also praised the fact that they were able to access all the functions offered by the two Explorer and Experimenter interfaces in the current system from a single screen. Accessing commands through the menus located on the main screen was also highlighted as a natural step.

In general, users had a more positive reaction towards the new interface compared to the current one. This, coupled with the fact that the times and usability scores were consistently better for the new interface suggests that it has achieved the basic aim of a more usable interface.
8.4 Improvements

Based on the evaluations performed on the new interface, a number of improvements have been made to the implementation. As these were generally only small changes, they did not require as thorough a design process as was carried out during the main design stage, but it was still very important to consider all the ways in which new additions will impact existing elements of the interface.

‘Select All’ and ‘Select None’ options have been added to the import data sets and import algorithms functions in the Experiment section, as suggested by one of the Weka users. This allows all the items or none of the items to be selected with one mouse click, which could save the user time if there are a large number of items they wish to import.

To make using the wizard functionality more efficient, a new option has been added to the first screen allowing the user to use a data set that they currently have selected in the project list. Users performing the usability tests commented that the absence of this option was slightly unintuitive and could prove inconvenient. The additional option is disabled if the user has not opened a data set from the main screen.

The size of the help text at the top of each wizard screen has been increased, and a larger border placed around it to make it more obvious. This follows the observation of users, who very rarely looked at the help text, as they did not see it. Now that it is clearer, users are more likely to read it, and so it should help them if they are unclear about what their next action should be.

8.5 Conclusions

A slightly disappointing aspect of the evaluation of the interface was the limited input by the Weka users into the process. As they are the ones who would be using the interface, their evaluations would have provided the most valuable insight into the effectiveness of the interface. Given that there was a limited response to the request for evaluations of the current interface, which simply involved completing a questionnaire, it is perhaps not surprising that this response would also be small when a far more complex task was required of users.

However, the few responses that were received gave some indication as to what the Weka users may feel about the interface. The results were generally positive, giving pleasing scores and praising certain areas of the interface, such as the wizard structure and algorithm selection method.

Overall, the evaluation by Weka users gave a positive opinion of the new interface, but unfortunately we cannot be especially confident in this result due to the limited amount of response data.

The usability tests performed with novice users were generally more effective in delivering results. The technique used allowed users to directly compare the two interfaces, and in general the users favoured the new one. As one of the problems originally identified in the existing interface was the steep learning curve for new users, this is undoubtedly a good sign.
The observation technique meant that the evaluator could see what problems the user encountered, as well as highlighting the problems the user chose to comment on. The time taken to complete the tasks was also recorded to gain an idea of how the efficiency of the interfaces compares. Overall, this technique gave a very detailed analysis of how a user will experience the interface.

Although the evaluation with Weka users was a little disappointing, when combined with the results gained from tests with novice users, we can be far more confident of the outcome. Both evaluation techniques presented favourable results for the new interface, a fact that suggests the interface as achieved the main project aim of providing a more user-friendly method of using the Weka system.
9. Conclusion

9.1 Overall Evaluation

9.1.1 Evaluation of methods and procedures used

The aim of this project was to create a new user interface for Weka that did not suffer from the usability problems that are apparent in the current interface. A large number of problems were revealed by the evaluation of the current interface, with three different methods used to give a thorough analysis of the issues that existed.

To ensure that the new interface did not suffer from the same problems, a set of requirements was developed from the evaluation of the current interface performed, along with the developed use cases. The combination of these two processes, along with the usability principles taken from the background work completed on user interface design provided an excellent basis for the development of the requirements.

However, the development of the requirements was made more difficult due to the fact that contact could not easily be made with users to help clarify and validate the requirements. Although requirements were generated fairly successfully using the methods described above, the whole process could possibly have been made more efficient and robust if more research had been performed into other methods of generating requirements in the absence of users.

During the design phase, ideas for the design of the interface were created from the requirements and the use cases developed during the requirements gathering stage. The use cases proved especially useful in helping envisage the tasks that users will be performing using the interface, and allowed a task-based approach to be taken to the design, as suggested in the background work carried out on user interface design in Chapter 3. This work was also particularly useful in highlighting the most appropriate design evaluation techniques to use after each design iteration.

The implementation stage was one that could have potentially have caused problems, as it can be difficult to make an estimate about how long a piece of software will take to implement. It is very difficult to foresee the problems that will be encountered, and, depending on their nature, these issues can cause a significant delay to the progress of work. Fortunately, thanks to careful planning and thorough testing of each module, the interface was built and connected to the underlying code without any major problems.

Generating the test cases as the software was being implemented proved to be a good decision. Creating such a detailed test plan during the design stage without having the interface to work from would have been a difficult and time-consuming task, and so the method used almost certainly saved time. One potential disadvantage was that having to generate test cases in the middle of implementation broke up the flow of coding slightly, but after a period of time this became more naturally incorporated as an extra step in the coding process.

One of the key aims of this project was to have the new interface evaluated by the users of Weka. These users will have experience using the existing interface and are therefore in an ideal position to compare it to the new design. Although this aim has been achieved, with generally favourable results from the users, the evaluation was not as thorough as it possibly could have been. Due to the fact that the users’ locations are
spread all around the world, it was not possible to actually meet them and observe them
using the system, which would have been the ideal form of evaluation. Instead, they
were asked to fill in a questionnaire on their opinions of both the old and new interfaces.

Unfortunately, the response to these requests was a little disappointing, with less than
ten responding on each occasion. This would be a sufficient number if observational
techniques that provide a very detailed analysis of the users’ experience with the system
had been used, but to be confident in results from a less detailed questionnaire, ideally
more than this would have been received. However, whilst the evaluation was not as
thorough as it possibly could have been, the responses that were received were enough
to give an indication at least of the users’ opinions of the interface.

In hindsight, more research could have been performed into methods for encouraging
users to respond to questionnaires, such as offering incentives, and if the project were to
be run again, this would be one area that could certainly have received more attention.
One option may have been to send out an email earlier in the project asking for
volunteers to help with the development of the new interface. By communicating
directly to a group of users rather than the whole Weka mailing list, it may have been
easier to convince them to contribute their opinions to the project. It would probably
have also been possible to involve them in other areas of the project, such as the
requirements development, which did not seem appropriate when emailing the entire
mailing list.

9.1.2 Evaluation of final product

Due to the slightly limited amount of testing by users, it is difficult to say with absolute
certainty how usable the new interface will seem to current Weka users. However, we
can be fairly confident that new users will find the new interface far more user-friendly
than the current one, thanks to the positive results achieved in the usability tests with
novice users. As the steep learning curve for new users was identified as a significant
problem with the current interface, this is an important achievement.

The fact that both current Weka users and novice users commented on problem areas
particularly highlighted by the techniques used during the evaluation of the current
interface suggests that the cognitive walkthrough and heuristic evaluation techniques
used were effective in identifying problems that users are likely to encounter.
Confidence in the fact that the new interface is user-friendly can be taken from this, as
during the development of the interface it was tested repeatedly using these techniques.

One of the key conclusions taken from the background work completed on user interface
design was the importance of using elements of existing interfaces in a new design. The
new interface employs a far more standard layout than the current interface, with a menu
bar and toolbar located at the top of the screen. These are common features of a user
interface, which should help users become proficient at using the new interface far more
quickly, as they will not have to spend time learning how to use interface components
that they have not used before. This is very important as ease of learning was identified
as one of the key factors affecting the usability of an interface.

One of the problems with the old interface was that it attempted to fit a great deal of
information into a limited space, making it harder for the user to pick out the
information they need. One of the challenges of implementing the new interface was in
fitting this large amount of information into the same limited space, but it a far more
convenient manner. Using menus allowed access to many of the functions to be reduced to a much smaller area, and allowed them all to be accessed from a single screen. The wizard structure used also proved extremely effective in this context, as it allowed the information to be split up into separate screens without reducing the usability of the interface.

A key aim of this project was to remove the major usability problems that were identified in the old interface. In general, this has been achieved. One of the major problems identified very early on in the project was the fact that the interface was split into four different sections. In the new interface this problem has been rectified, with all the functionality from the Explorer and Experimenter sections of the old interface combined into one single, consistent interface. Another key problem was the lack of information available about the algorithms during the algorithm selection process. This problem has also been solved, with descriptions of the algorithms available while the user is selecting the one they wish to use.

Another key aim was the inclusion of an additional piece of functionality in the interface that allows users to automatically import their own code modules to the system. The old interface did not allow users to do this, and so code modules had to be manually added. The new interface achieves this aim through a simple function that allows the user to select the class file and specify the package in which it should be placed. The module is then automatically imported.

Thanks to the usability tests performed by novice users, and the cognitive walkthrough and heuristic evaluation techniques used, the limited amount of evaluation performed by current Weka users is not the problem that it could potentially have been. The results of these tests indicated that the new interface is far more usable than the old interface, and so we can conclude that the main aim of the project, to develop a more user-friendly interface for Weka, has been achieved.

In conclusion, the only major aim that has not been achieved to the level hoped was the limited amount of testing by the Weka users. Other than this, the new interface has achieved all the main aims that were set at the start of this project, and so in general can be considered a success.

9.2 Future Developments

Due to time constraints and limits to the scope of this project, there are still remaining areas in which the interface can be improved. We will now discuss these areas, and suggest how they may be improved and developed further in the future.

One of the areas that is lacking in the current interface is the results output from algorithms. The results are simply displayed as text, and this is often not the most useful format for them to be in. For example, a number of possible output types are displayed as tables within the text string. These would be far better represented in a table object that allowed the user to change the formatting such as column widths and row heights, as some of the information can become unclear in the fixed size used within the text. A table such as this would also allow the users to export the results to formats such as spreadsheets in a far more intuitive way. Unfortunately, the generation of the text output is not performed by the interface, but by the underlying Weka code, and was therefore
Another area that could be improved is the ability of the system to check for errors in the set up of an algorithm before it is actually run. For example, a user may be using a data set with only numerical attributes, but has selected a classifier that requires nominal attributes. The error will not be detected until the algorithm is actually run, which can be frustrating for the user. Again, this issue requires alteration of the underlying Weka code, and so was deemed outside the scope of this project. This is an area that could easily be improved in the future, by including a set of standard methods in each of the algorithms that allowed the interface to check the requirements for the algorithm, and compare it to the data set currently being used by the user. This would allow users to discover their errors as soon as they have made them, and would reduce this problem that can often become frustrating.

As highlighted in Chapter 2, there are many different ways of visually representing the different aspects of data mining, and Weka offers just a few of these. In addition to users being able to add their own algorithms to the system, Weka can also incorporate users’ own visualization modules. Although this has not been implemented in the new interface, it has been designed in such a way that will make it a fairly simple task to incorporate this in future development work. The ‘Visualize’ panel on the main screen can be set up to contain any new visual representation, and can be connected into the interface by adding an additional element to the ‘Visualize’ menu. Ideally, a function similar to the one used to import users’ own algorithms would allow users to do this automatically.

Another future development may be a further improvement of the use of plug-in functionality within the interface. Although users are now able to automatically import their own algorithms into the system through the user interface, removing them is still a fairly inconvenient process, involving deleting the actual file as well as changing the properties file. If this process were automated it would be far more convenient, although careful consideration would have to be taken to ensure users can easily recover if they accidentally remove an algorithm they did not mean to.

9.3 Personal Reflection

Taking on such a large project has presented a significant challenge, but it is one I feel I have coped well with and have learned a great deal from. As well as the improvements to my planning and organisational skills, a great deal of knowledge has been gained on the wide-ranging area of data mining – a topic of which I have had no previous experience.

One of the most interesting stages of the project was the design phase, which allowed me to use my creative skills to produce the designs. The Weka interface requires a great deal of information to be displayed, and designing the most effective way of fitting it all in whilst ensuring the interface remained user-friendly was particularly challenging.

Another challenge presented by the project was the inability to meet up with users. Users are key to any user interface related project, and not having direct access to them posed additional problems. However, by performing research into data mining, and using a
number of different evaluation techniques that did not require access to the users, I felt that I successfully compensated for the lack of direct contact with users.

It has been a rewarding experience working towards the creation of a product that may contribute to the future of the Weka software. If the new interface is incorporated into future releases of the software, it will hopefully go some way towards making the program more accessible for new users, and making using the software a generally more pleasant experience.

In general, a great deal of satisfaction can be taken from the project, as it appears to have successfully fulfilled the major aims that were set out when the project began.
10. Bibliography


ECKSTEIN, R., 2005. *Creating Wizard Dialogs with Java Swing*, available from Internet


THEARLING, K., No Year. An Introduction to Data Mining: Discovering hidden value in your data warehouse, available from Internet <http://www.thearling.com/text/dmwhite/dmwhite.htm> (14th November 2005)

Appendix A: Interface Overviews

Contained within this section is a brief overview of both the current Weka interface and the new interface designed in this project, to be used as a reference if access to the software is not available. A selection of screenshots of each interface are presented, along with a brief description.
A.1 Current Interface

A.1.1 GUI Chooser

The first step a user must take is to choose which of the four interfaces to use.

![GUI Chooser](image1)

Figure A.1 – GUI Chooser

A.1.2 Explorer

If the user selects the Explorer interface, they are presented with the Preprocessing panel (Figure A.2). Here they can open a data set, apply filters, and manually select attributes.

![Preprocessing Panel](image2)

Figure A.2 – Preprocessing Panel

To apply an algorithm to the data set, they must select the appropriate tab, for example the Classify tab (Figure A.3). Results are displayed in a textual form on the right hand side of the panel.

![Classify Panel](image3)

Figure A.3 – Classify Panel
Users can view visual representations of the data in the Visualize tab (Figure A.4).

**Figure A.4 – Visualize Panel**

A.1.3 Experimenter

When the Experimenter interface is selected, users are presented with the Setup panel (Figure A.5). Here they can select the algorithms and data sets to use in the experiment, and select their desired experiment options.

**Figure A.5 – Setup Panel**

To run the experiment, users must select the Run tab (Figure A.6) and click the Start button.

**Figure A.6 – Run Panel**
The results of an experiment can be analysed from the Analyse tab (Figure A.7).

A.1.4 Knowledge Flow

The knowledge flow interface (Figure A.8) offers the same functionality as the Explorer interface, but presents it in a different way. Components representing the Weka functions are added to the workspace and connected together.

Figure A.7 – Analyse Panel

Figure A.8 – Knowledge Flow Interface
A.2 New Interface

A.2.1 Main Screen

When the user starts up the new interface they are presented with the main screen. Once a data set is opened, the data set details are displayed (Figure A.9).

Algorithm results can be viewed using the Text tab (Figure A.10).

Visual representations of the data can be viewed using the Visualize tab (Figure A.11).
Algorithms can be selected and run using a wizard structure (Figure A.12).

Figure A.12 – Classifier Selection
Appendix B: Use Cases and Requirements

This section contains listings of the use cases and requirements developed for the new interface.
**B.1 Use Cases**

The following is a list of use cases for the interaction between the user and the Weka interface for performing key tasks offered by the software. As the Weka users cannot be split into easily defined groups, there are no specific actors that can be identified, and all use cases will therefore be assumed to have the general actor of “Weka user”.

The use cases have been structured in the manner set out by Preece *et al.* (2002).

**Use Case 1: User opens data set in ARFF, CSV or C4.5 file format**

1. User selects ‘Open File’ option.
2. A standard ‘Open’ dialog box is displayed.
3. The user selects the directory of the file they wish to open.
4. The user selects the data set they wish to open, or enters the name in the ‘File name’ text field.
5. The user clicks the ‘Open’ button.
6. The system opens the data set and displays the name and details of it on screen.

Alternative courses:

6. If the file name entered does not exist in the current directory.
   6.1 The system displays an error message.
   6.2 Return to step 4.

**Use Case 2: User opens data set from URL**

1. User selects ‘Open URL’ option.
2. An ‘Open URL’ dialog box is displayed.
3. The user enters the URL address in the ‘URL’ text field.
4. The user clicks the ‘Open’ button.
5. The system connects to the URL location and opens the data set. The details of the data set are displayed on screen.

Alternative courses:

5. If the URL does not contain a valid data set.
   5.1 The system displays an error message.
   5.2 Return to step 3.

**Use Case 3: User opens data set from database**

1. User selects ‘Open Database’ option.
2. An ‘Open Database’ dialog box is displayed.
3. The user clicks the ‘Open’ button next to the database field.
4. An ‘Open’ dialog box is opened.
5. The user selects the directory of the database they wish to open.
6. The user selects the database they wish to open, or enters the name in the ‘File name’ text field.
7. The user clicks the ‘Open’ button.
8. The system returns to the ‘Open Database’ dialog box with the selected database and location displayed in the Database field.
9. The user enters a username and password if required and a query on the database if required.
10. The user clicks ‘Ok’.
11. The system opens the database as a data set.

Alternative courses:

3. If the user knows the location and filename of the database.
   3.1 The user types in the relative path of the database.
   3.2 Go to step 9.
8. If the file name entered doesn’t exist in the current directory.
   8.1 The system displays an error message.
   8.2 Go back to step 6.
11. If the system cannot connect to the database.
   11.1 The system displays an error message.
   11.2 Go back to step 8.

Use Case 4: User selects attributes manually

1. User opens a data set.
2. Data set attributes are displayed.
3. User selects attributes they wish to remove.
4. User clicks ‘Remove’.
5. The attributes are removed from the data set and the interface informs the user of this fact.

Alternative courses:

5. All attributes were selected.
   5.1 System displays an error message.
   5.2 Go back to step 3.

Use Case 5: User selects attributes using algorithm

1. User opens a data set.
2. User selects ‘Select Attributes’.
3. A new window opens containing list of attribute selection algorithms on left hand side.
4. The user clicks on an algorithm.
5. Details of the algorithm are displayed on the right hand side of the screen.
6. User clicks ‘Ok’.
7. System displays list of parameters for chosen algorithm.
8. User sets desired parameters and clicks ‘Ok’ button.
9. The system displays the name of the algorithm and the parameter codes in the ‘Algorithm’ field.
10. The user clicks ‘Apply’.
11. Algorithm is run on data set and the system signifies that it is processing while algorithm is running. Results of the algorithm are displayed in the results window.
12. User clicks ‘Remove rejected attributes’.
13. The attributes are removed from the data set and the interface informs the user of this fact.

Alternative courses:

8. If the user clicks ‘Save Settings’
   8.1 Standard ‘Save’ dialog box opens.
   8.2 User selects directory and enters file name, then clicks ‘Ok’.
   8.3 Parameter settings are saved.
   8.4 Go back to step 9.

8. If the user clicks ‘Load Settings’
   8.1 Standard ‘Open’ dialog box opens.
   8.2 User selects directory and settings file, and then clicks ‘Ok’.
   8.3 Parameter settings are loaded for current algorithm.
   8.4 Go back to step 9.

9. If the user clicks ‘Cancel’ button.
   9.1 Go back to step 7.

11. If the user clicks ‘Edit Parameters’ button.
   11.1 Go back to step 9.

Use Case 6: User applies preprocessing filter to data set

1. User opens a data set.
2. User selects ‘Filters’.
3. System displays filters options.
4. User selects ‘Choose Filter’ button.
5. A new window opens containing list of filters on left hand side.
6. The user clicks on a filter.
7. A description of the filter is displayed on the right hand side of the screen.
8. User clicks ‘Ok’.
9. System displays list of parameters for chosen filter.
10. User sets desired parameters and clicks ‘Ok’ button.
11. The system displays the name of the filter and the parameter codes in the ‘Filter’ field.
12. The user clicks ‘Apply’.
13. Filter is applied to the data set and user is taken back to main screen. Details of new filtered data set are now displayed.

Alternative courses:

9. If the user clicks ‘Cancel’ button.
   9.1 Go back to step 3.
10. If the user clicks ‘Save Settings’
    10.1 Standard ‘Save’ dialog box opens.
    10.2 User selects directory and enters file name, then clicks ‘Ok’.
    10.3 Parameter settings are saved.
    10.4 Go back to step 9.

10. If the user clicks ‘Load Settings’
    10.1 Standard ‘Open’ dialog box opens.
    10.2 User selects directory and settings file, and then clicks ‘Ok’.
    10.3 Parameter settings are loaded for current filter.
    10.4 Go back to step 9.

11. If the user clicks ‘Cancel’ button.
    11.1 Go back to step 7.

13. If the user clicks ‘Edit Parameters’ button.
    13.1 Go back to step 9.

**Use Case 7: User applies algorithm to data set**

1. User opens a data set.
2. User selects relevant category of algorithms.
3. System displays algorithm type options.
4. User selects ‘Choose Algorithm’ button.
5. A new window opens containing list of algorithms on left hand side.
6. The user clicks on an algorithm.
7. A description of the algorithm is displayed on the right hand side of the screen.
8. User clicks ‘Ok’.
9. System displays list of parameters for chosen algorithm.
10. User sets desired parameters and clicks ‘Ok’ button.
11. The system displays the name of the algorithm and the parameter codes in the ‘Algorithm’ field.
12. The user clicks ‘Apply’.
13. Algorithm is run on data set and the system signifies that it is processing while algorithm is running. Results of the algorithm are displayed in the results window.

Alternative courses:

5. If the user clicks ‘Cancel’ button.
   5.1 Go back to step 2.

9. If the user clicks ‘Save Settings’
   9.1 Standard ‘Save’ dialog box opens.
   9.2 User selects directory and enters file name, then clicks ‘Ok’.
   9.3 Parameter settings are saved.
   9.4 Go back to step 9.

9. If the user clicks ‘Load Settings’
   9.1 Standard ‘Open’ dialog box opens.
9.2 User selects directory and settings file, and then clicks ‘Ok’.
9.3 Parameter settings are loaded for current algorithm.
9.4 Go back to step 9.

9. If the user clicks ‘Cancel’ button.
9.1 Go back to step 7.

12. If the user clicks ‘Edit Parameters’ button.
12.1 Go back to step 9.

13. If the selected algorithm cannot be applied to the chosen data set.
13.1 An error message is displayed.

Use Case 8: User runs experiment to compare two algorithms

1. User selects ‘Experiment’.
2. System displays experiment options.
3. User selects ‘Add Data Set’ button.
4. Standard ‘Open’ window is displayed.
5. User locates and selects data set and clicks ‘Open’.
6. Data set is displayed in ‘Data Sets’ list.
7. User selects ‘Add Algorithm’ button.
8. A new window opens containing list of algorithms on left hand side.
9. The user clicks on an algorithm.
10. A description of the algorithm is displayed on the right hand side of the screen.
11. User clicks ‘Ok’.
12. System displays list of parameters for chosen algorithm.
13. User sets desired parameters and clicks ‘Ok’ button.
15. User sets other parameters of the experiment, and clicks ‘Run’.
16. System signifies that it is running the process, and, when finished, displays confirmation of successful completion.

Alternative courses:

5. If the user clicks ‘Cancel’ button.
5.1 Go back to step 2.

7. If the user clicks ‘Add Data Set’ button again.
7.1 Go back to step 4.

11. If the user clicks ‘Cancel’ button.
11.1 Go back to step 4.

13. If the user clicks ‘Save Settings’
13.1 Standard ‘Save’ dialog box opens.
13.2 User selects directory and enters file name, then clicks ‘Ok’.
13.3 Parameter settings are saved.
13.4 Go back to step 12.

13. If the user clicks ‘Load Settings’
13.1 Standard ‘Open’ dialog box opens.
13.2 User selects directory and settings file, and then clicks ‘Ok’.
13.3 Parameter settings are loaded for current algorithm.
13.4 Go back to step 12.

14. If the user clicks ‘Cancel’ button.
14.1 Go back to step 7.

15. If the user clicks ‘Add Algorithm’ button again.
15.1 Go back to step 8.

16. If the selected algorithm cannot be applied to the chosen data set.
16.1 An error message is displayed.

Use Case 9: User analyses results of an experiment

1. User runs an experiment.
2. User selects ‘Analyse Results’ option.
3. Analyse results options are displayed.
4. User sets the options they desire and clicks the ‘Run Analysis’ button.
5. The system indicates that it is running the analysis and signals when it is finished.
6. The results of the analysis are displayed in the results window.

Alternative courses:

4. If the user clicks ‘Load File’ button.
   4.1 An ‘Open’ dialog box is displayed.
   4.2 The user locates the experiment file they wish to load and clicks ‘Open’.
   4.3 Name of loaded experiment is displayed.
   4.4 Go back to step 3. Analysis will now be run on loaded experiment.

Use Case 10: User views visual representation of data set

1. User opens data set.
2. User selects ‘Visualize’ option.
3. System displays two-dimensional plot of first two attributes of data set.
4. User selects attributes for the x and y axis of the plot.
5. User selects other options for plot display, and clicks ‘Update’ button.
6. Plot is updated with user’s settings.

Use Case 11: User incorporates own module into system

1. User places their code in the Weka directory
2. User selects ‘Import Module’ option.
3. An ‘Import Module’ window is opened.
4. User enters the name of the class they wish to import in the ‘Class Name’ text field.
5. The user clicks ‘Import’ button.
6. The system calls the imported module and displays its output in ‘Import’ window.

Alternative courses:

6. If the module cannot be found.
   6.1 System displays an error message.
   6.2 Go back to step 3.

6. There is an error in the module and it cannot be run.
   6.1 System displays an error message.
   6.2 Go back to step 3.

6. If the user clicks ‘Cancel’ button.
   6.1 System displays an error message.
   6.2 Go back to step 2.
B.2 Functional Requirements

1 A single interface should offer the functionality currently available in the Explorer and Experimenter interfaces.

2 The interface must allow users to open data sets.
   2.1 The system must allow the user to open data sets stored as a file or database or from a URL.
   2.2 The system must give a visual indication that the data set has been successfully opened.
   2.3 The user should be able to easily see the name of the data set they are currently working on.
   2.4 The system should allow the user to switch between multiple open data sets.

3 The interface must allow users to apply preprocessing filters to a data set.
   3.1 A description of the filters should be available while the user is selecting the filter to use.
   3.2 After selecting a filter the user should be offered the opportunity to set the parameters of the filter.
   3.3 Once applied, a filter should update the name of the data set.
   3.4 The interface should display the changes in the details of the data set.

4 The interface must allow the user to apply all algorithms available in Weka to a data set.
   4.1 A description of the algorithm should be available when the user is selecting it.
   4.2 After selecting an algorithm the user should be offered the opportunity to set the parameters of the algorithm.
   4.3 The appropriate test options should be available for each algorithm.
   4.4 Once a user chooses to apply an algorithm, the interface should indicate the progress of the algorithm.
   4.5 Once the application of an algorithm has been completed, the results of the application should be displayed in a clear manner.

5 The interface must allow users to view visual representations of the data.
   5.1 The user must be able to plot any attribute of a data set against any other.
   5.2 The user should be able to choose the colours and size of the plots.
   5.3 The interface must offer all other visualization settings offered by the current interface.
   5.4 The interface must allow visual representations of algorithm results to be displayed.

6 The interface must allow users to perform experiments to compare the performance of different algorithms on data sets.
   6.1 The user must be able to add multiple data sets to an experiment.
   6.2 The user must be able to add multiple algorithms to an experiment.
   6.3 The user should be able to include data sets and algorithms already explored in their experiment.
   6.4 The settings available to the user in the current Experimenter interface must be available in the new interface.
6.5 Users must be able to analyse experiments using the same options offered by the current interface.
6.6 The interface must alert the user once the experiment has completed.

7 The interface should offer feedback to the user.
7.1 The interface should make it obvious when it is currently processing a task.
7.2 The system must offer feedback to show that user actions have been successful.

8 Error messages should be displayed whenever appropriate.
8.1 Error messages should be clear and explain why an error has occurred.
8.2 Error messages should offer the user a potential solution to the problem.

9 Users should be able to recover easily from any problems they encounter.
9.1 ‘Undo’ and ‘Redo’ buttons should be available for the user to recover from any mistakes they make.
9.2 A wizard structure should allow users to move back and forth through the steps of all major functionality.

10 Users should be able to easily incorporate their own modules.
10.1 Users should be able to call their own code from a function within the interface.
10.2 Correctly implemented code should be incorporated neatly into the interface.
10.3 Help information regarding the incorporation of modules should be available.

11 Shortcut key combinations should be used for standard tasks.
10.1 ‘Ctrl+C’ must copy the current text.
10.2 ‘Ctrl+P’ must open a print dialog window.
10.3 ‘Ctrl+Z’ must undo the last action.
10.4 ‘Ctrl+Y’ must redo the last action undone.
10.5 ‘Ctrl+S’ must open a save dialog box to save current results buffer.
10.6 ‘F1’ must open the help file.
10.7 ‘Ctrl+O’ must display an ‘Open File’ dialog.
10.8 ‘Ctrl+W’ must begin a wizard dialog.
10.9 ‘Ctrl+F’ must call the function to apply a filter to the data.
10.10 ‘Ctrl+G’ must call the function to run an attribute selection algorithm.
10.11 ‘Ctrl+H’ must call the function to run a classification algorithm.
10.12 ‘Ctrl+J’ must call the function to run a clustering algorithm.
10.13 ‘Ctrl+K’ must call the function to run an association rules algorithm.
10.14 ‘Ctrl+E’ must call the function to begin an experiment.
B.3 Non-Functional Requirements

1 The new interface must be responsive.
   1.1 The interface must react within a few seconds to a user action.
   1.2 The user must be aware when the system is busy and cannot perform any other actions.

2 The new interface should be as user friendly as possible.
   2.1 It should be clear what actions a user should take to complete a task.
   2.2 Error messages should be helpful and not patronise the user.
   2.3 The interface should use elements of other interfaces users may already be familiar with.
   2.4 Tab ordering should be set to sensibly follow the order in which users are likely to use elements of the interface.
   2.5 Initial cursor placement on any screen should be sensible.
   2.6 Changes in system status should be made obvious to the user.
   2.7 Accelerator keys should be used to increase speed of use.
   2.8 The system must support screen readers as far as possible.

3 The interface should use minimalist design.
   3.1 No more than five colours should be used in the system.
   3.2 Colour should be used intuitively, for example red to highlight an error, but must not be used to convey information.
   3.3 The interface should not display too much information at any one time.

4 The system should offer the user adequate help.
   4.1 Tool Tips should offer a description when the mouse hovers over a function.
   4.2 A help file should be available to explain how to use the interface.

5 The interface should use common user interface components for appropriate functions.
   5.1 The interface should use standard menus such as ‘File’, ‘Edit’ and ‘Help’.
   5.2 Drop down menus or radio buttons should be used to select an option from a list.
   5.3 Check boxes should be used to turn an option ‘On’ or ‘Off’.

6 The interface should use domain related terminology.
   6.1 The interface should use common data mining terms to describe functionality.
   6.2 The new interface should use similar terms to the current interface where possible without diminishing the usability of the interface.

7 Integrating the new interface for use should be as simple as possible.
   7.1 Users should be able to begin using the new interface in a few simple steps.
   7.2 Documentation should be provided to assist the user in installing the new interface.
Appendix C: Code Listings

Due to the length of the source code, a listing will not be included here. All the source code can be found on the accompanying CD and will be available at the following URL until June 2006:

http://students.bath.ac.uk/cs2aej/wekacode/index.html
Appendix D: Interface Evaluation

The following section gives the results of the interface evaluation techniques used to analyse both the current Weka interface and the new one developed.
D.1 Usability Questionnaires

Detailed below are the results of the questionnaires completed by current Weka users. The questionnaire used was the Computer System Usability Questionnaire developed by Lewis (1995) and implemented online by Perlman (No Year).

D.1.1 Current Interface

Table D.1: Usability Questionnaire Results for Current Interface

<table>
<thead>
<tr>
<th>Question</th>
<th>User</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall, I am satisfied with how easy it is to use this system</td>
<td>6 4 6 1 6 5 5 5</td>
<td>4.75</td>
</tr>
<tr>
<td>It was simple to use this system</td>
<td>7 5 4 1 5 6 4 4</td>
<td>4.50</td>
</tr>
<tr>
<td>I can effectively complete my work using this system</td>
<td>5 5 3 2 7 4 6 5</td>
<td>4.63</td>
</tr>
<tr>
<td>I am able to complete my work quickly using this system</td>
<td>6 6 2 3 6 7 5 5</td>
<td>5.00</td>
</tr>
<tr>
<td>I am able to efficiently complete my work using this system</td>
<td>5 6 2 3 7 4 5 5</td>
<td>4.63</td>
</tr>
<tr>
<td>I feel comfortable using this system</td>
<td>5 5 4 1 7 7 6 6</td>
<td>5.13</td>
</tr>
<tr>
<td>It was easy to learn to use this system</td>
<td>7 5 2 1 5 6 3 5</td>
<td>4.25</td>
</tr>
<tr>
<td>I believe I became productive quickly using this system</td>
<td>7 6 3 1 6 5 4 6</td>
<td>4.75</td>
</tr>
<tr>
<td>The system gives error messages that clearly tell me how to fix problems</td>
<td>4 3 1 1 2 4 2 3</td>
<td>2.50</td>
</tr>
<tr>
<td>Whenever I make a mistake using the system, I recover easily and quickly</td>
<td>3 4 2 1 3 3 2 5</td>
<td>2.88</td>
</tr>
<tr>
<td>The information (such as online help, on-screen messages, and other documentation) provided with this system is clear</td>
<td>6 4 1 1 5 4 2 3</td>
<td>3.25</td>
</tr>
<tr>
<td>It is easy to find the information I needed</td>
<td>3 4 4 1 5 3 3 5</td>
<td>3.50</td>
</tr>
<tr>
<td>The information provided for the system is easy to understand</td>
<td>4 4 3 1 5 4 2 6</td>
<td>3.63</td>
</tr>
<tr>
<td>The information is effective in helping me complete the tasks and scenarios</td>
<td>5 4 4 3 6 4 4 6</td>
<td>4.50</td>
</tr>
<tr>
<td>The organization of information on the system screens is clear</td>
<td>6 5 2 1 6 5 4 6</td>
<td>4.38</td>
</tr>
<tr>
<td>The interface of this system is pleasant</td>
<td>6 4 5 1 6 6 4 3</td>
<td>4.38</td>
</tr>
<tr>
<td>I like using the interface of this system</td>
<td>6 4 4 1 6 6 4 1</td>
<td>4.00</td>
</tr>
<tr>
<td>This system has all the functions and capabilities I expect it to have</td>
<td>2 4 3 6 7 4 3 3</td>
<td>4.00</td>
</tr>
<tr>
<td>Overall, I am satisfied with this system</td>
<td>7 6 4 2 7 6 5 4</td>
<td>5.13</td>
</tr>
</tbody>
</table>

Scale: 1 = Strongly Disagree, 7 = Strongly Agree
D.1.2 New Interface

Table D.2: Usability Questionnaire Results for New Interface

<table>
<thead>
<tr>
<th>Question</th>
<th>User</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall, I am satisfied with how easy it is to use this system</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>It was simple to use this system</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>I can effectively complete my work using this system</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>I am able to complete my work quickly using this system</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>I am able to efficiently complete my work using this system</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>I feel comfortable using this system</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>It was easy to learn to use this system</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>I believe I became productive quickly using this system</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>The system gives error messages that clearly tell me how to fix problems</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Whenever I make a mistake using the system, I recover easily and quickly</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>The information (such as online help, on-screen messages, and other documentation) provided with this system is clear</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>It is easy to find the information I needed</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>The information provided for the system is easy to understand</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>The information is effective in helping me complete the tasks and scenarios</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>The organization of information on the system screens is clear</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>The interface of this system is pleasant</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>I like using the interface of this system</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>This system has all the functions and capabilities I expect it to have</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Overall, I am satisfied with this system</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

Scale: 1 = Strongly Disagree, 7 = Strongly Agree
D.2 Usability Tests

Usability tests were performed by observing novice users using both the current Weka interface and the new one. Outlined below are the tasks the users were asked to complete along with the results of the tests. The results are set out in the format suggested by Preece et al. (2002).

D.2.1 Tasks

Task #1 – Explore Weka

1. Start Weka.
2. Open a data set from a file.
3. Explore the interface.

Task #2 – Preprocessing tasks

1. Select an algorithm to select the most appropriate attributes to use.
2. Specify the parameters of the algorithm.
3. Select a search method for the attribute selection algorithm.
4. Specify the parameters of the search method.
5. Specify the run options of the algorithm.
6. Run the algorithm.
7. Remove unwanted attributes.
8. Select a filter to apply to the data set.
9. Specify the parameters of the filter.
10. Apply the filter.

Task #3 – Classification

1. Select a classification algorithm to apply to the data.
2. Specify the parameters of the algorithm.
3. Specify the test options you wish to use.
4. Run the algorithm.
5. View the Cost Curve of the results of the algorithm.

Task #4 – Experiment

1. Select at least one algorithm to use in an experiment.

2. Select at least one data set to use in an experiment.

3. Provide a destination filename to save the experiment results.

4. Specify the experiment options.

5. Run the experiment.

6. Run an analysis of the results of the experiment.
### D.2.2 Results for Current Interface

Table D.3: Usability Test Results for Current Interface

<table>
<thead>
<tr>
<th>Participant Number:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Background Information</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sex</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>F</td>
<td>F</td>
<td>M</td>
<td>4M, 2F</td>
</tr>
<tr>
<td>Age (years)</td>
<td>21</td>
<td>16</td>
<td>51</td>
<td>54</td>
<td>19</td>
<td>22</td>
<td>30.5</td>
</tr>
<tr>
<td>Computer Experience</td>
<td>H</td>
<td>H</td>
<td>M</td>
<td>L</td>
<td>M</td>
<td>H</td>
<td>3H, 2M, 1L</td>
</tr>
<tr>
<td>Data Mining Experience</td>
<td>N</td>
<td>N</td>
<td>L</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>5N, 1L</td>
</tr>
<tr>
<td>First Interface Tested</td>
<td>Cur</td>
<td>Cur</td>
<td>Cur</td>
<td>New</td>
<td>New</td>
<td>New</td>
<td>3 Cur, 3 New</td>
</tr>
<tr>
<td><strong>Structured Tasks</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Open Data Set</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1.7</td>
</tr>
<tr>
<td>Choose Select Attributes Algorithm</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2.3</td>
</tr>
<tr>
<td>Specify Parameters</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3.0</td>
</tr>
<tr>
<td>Choose Search Method</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2.0</td>
</tr>
<tr>
<td>Specify Parameters</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2.2</td>
</tr>
<tr>
<td>Run Options</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1.2</td>
</tr>
<tr>
<td>Run Algorithm</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1.3</td>
</tr>
<tr>
<td>Remove Attributes</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2.8</td>
</tr>
<tr>
<td>Select Filter</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1.8</td>
</tr>
<tr>
<td>Specify Parameters</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1.7</td>
</tr>
<tr>
<td>Apply Filter</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1.5</td>
</tr>
<tr>
<td>Select Classification Algorithm</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1.8</td>
</tr>
<tr>
<td>Specify Parameters</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1.7</td>
</tr>
<tr>
<td>Specify Test Options</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>Run Algorithm</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>View Cost Curve</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3.0</td>
</tr>
<tr>
<td>Algorithm for Experiment</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3.0</td>
</tr>
<tr>
<td>Data Set for Experiment</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1.5</td>
</tr>
<tr>
<td>Destination Filename</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1.2</td>
</tr>
<tr>
<td>Experiment Options</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1.2</td>
</tr>
<tr>
<td>Run Experiment</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2.5</td>
</tr>
<tr>
<td>Analyse Results</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2.7</td>
</tr>
<tr>
<td><strong>Total Time (minutes)</strong></td>
<td>14</td>
<td>23</td>
<td>25</td>
<td>26</td>
<td>22</td>
<td>17</td>
<td>21.2</td>
</tr>
</tbody>
</table>

**Key:**
- Sex: M = Male, F = Female
- Computer Experience: H = High, M = Medium, L = Low
- Data Mining Experience: L = Low, N = None
- First Interface Tested: Cur = Current Interface, New Interface

**Scale:** 1 = Easy, 2 = OK, 3 = Difficult, Bold type = Needed Help
### D.2.3 Results for New Interface

Table D.4: Usability Test Results for New Interface

<table>
<thead>
<tr>
<th>Participant Number:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Background Information</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sex</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>F</td>
<td>F</td>
<td>M</td>
<td>4M, 2F</td>
</tr>
<tr>
<td>Age</td>
<td>21</td>
<td>16</td>
<td>51</td>
<td>54</td>
<td>19</td>
<td>22</td>
<td>30.5</td>
</tr>
<tr>
<td>Computer Experience</td>
<td>H</td>
<td>H</td>
<td>M</td>
<td>L</td>
<td>M</td>
<td>H</td>
<td>3H, 2M, 1L</td>
</tr>
<tr>
<td>Data Mining Experience</td>
<td>N</td>
<td>N</td>
<td>L</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>5N, 1L</td>
</tr>
<tr>
<td>First Interface</td>
<td>Cur</td>
<td>Cur</td>
<td>Cur</td>
<td>New</td>
<td>New</td>
<td>New</td>
<td>3 Cur, 3 New</td>
</tr>
<tr>
<td><strong>Structured Tasks</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Open Data Set</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1.3</td>
</tr>
<tr>
<td>Choose Select Attributes Algorithm</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1.7</td>
</tr>
<tr>
<td>Specify Parameters</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>Choose Search Method</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>Specify Parameters</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>Run Options</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>Run Algorithm</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1.3</td>
</tr>
<tr>
<td>Remove Attributes</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2.0</td>
</tr>
<tr>
<td>Select Filter</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1.8</td>
</tr>
<tr>
<td>Specify Parameters</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>Apply Filter</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1.2</td>
</tr>
<tr>
<td>Select Classification Algorithm</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1.3</td>
</tr>
<tr>
<td>Specify Parameters</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>Specify Test Options</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>Run Algorithm</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>View Cost Curve</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2.0</td>
</tr>
<tr>
<td>Algorithm for Experiment</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1.8</td>
</tr>
<tr>
<td>Data Set for Experiment</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1.2</td>
</tr>
<tr>
<td>Destination Filename</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>Experiment Options</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.3</td>
</tr>
<tr>
<td>Run Experiment</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1.3</td>
</tr>
<tr>
<td>Analyse Results</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1.2</td>
</tr>
<tr>
<td><strong>Total Time (minutes)</strong></td>
<td>10</td>
<td>17</td>
<td>19</td>
<td>25</td>
<td>22</td>
<td>14</td>
<td>17.8</td>
</tr>
</tbody>
</table>

**Key:**
- Sex: M = Male, F = Female
- Computer Experience: H = High, M = Medium, L = Low
- Data Mining Experience: L = Low, N = None
- First Interface Tested: Cur = Current Interface, New Interface

**Scale:** 1 = Easy, 2 = OK, 3 = Difficult, Bold type = Needed Help

---

90
Appendix E: User Documentation

As the interface has been designed to be as intuitive and self-explanatory as possible, only a small user guide has been developed. This covers the basic functions that do not appear in the current Weka system. If the interface were to be released for general use, this documentation would need to be developed to include the wider Weka functionality.
1. Getting Starting

When you begin using Weka the main screen will be displayed. From here you can access all the Weka functions, and can control the information that you wish to view.

From the main screen you have access to your ‘Project List’. This allows you to keep track of all the information that you generate during your current Weka session. The following items will be automatically added to your project list when you perform the relevant action:

- Data sets – any data sets you open will be added to the list, as well as any new data sets created by filtering the data.
- Algorithm results – when you run an algorithm, the results will be added to the project list.

Selecting a project from the list will update the information that is displayed on the right hand of the screen. By selecting the tabs, you can view data set details, textual output and visualization of the current project. The current visualization mode can be controlled from the ‘Visualize’ menu.

The first thing you will need to do is to open a data set. This can be done by selecting one of the following options from the ‘File’ menu, or the corresponding icon from the tool bar:

- Open File
- Open URL
- Open Database
Alternatively, data sets can be opened using the Wizard (see Section 3. Wizard).

2. Tools

To apply filters or algorithms to the data set, select the appropriate item from the ‘Tools’ menu. This will open a wizard that will guide you through the process. Follow the on-screen instructions to apply the chosen filter or algorithm to the data set currently selected in your project list.

Once the algorithm or filter has been applied, click the ‘Finish’ button to add the results to your project list.

3. Wizard

The interface offers a wizard to take you through the main functions of Weka. To begin the wizard, select ‘Wizard’ from the ‘Tools’ menu, or click the wizard icon on the tool bar.

Follow the instructions on-screen to move through the steps of the wizard. When you complete a task, click the ‘Finish’ button to add the result to your project list.

4. Experiment

To run an experiment to compare the performance of algorithms, select the ‘Experiment’ option from the ‘Tools’ menu. Follow the on-screen instructions to set up and run the experiment. Once the experiment has been run, click ‘Next’ to analyse the results, or ‘Finish’ to return to the main screen.

If you have any data sets or classifiers in your project list, these can be imported to an experiment using the ‘Import from project list’ buttons on the experiment set up screen.
5. Importing Code Modules

The interface allows you to automatically import your own Java classes that implement algorithms and filters.

*Note: The classes must contain the relevant package information, for example, “weka.classifiers.rules” to be successfully imported.*

To perform this task:

1. Select ‘Import’ from the ‘File’ menu.
2. Click ‘Browse’ and select the class file you wish to import.
3. Specify the relevant package using the drop down menus.
4. Click ‘Import’.

If successful, a message will be displayed informing you that the class has been imported.