DOCTOR OF PHILOSOPHY

Capturing and categorising user interaction

Hunnisett., David

Award date:
2009

Awarding institution:
Bangor University

Link to publication

Capturing and Categorising User Interaction

David Hunnisett

February 27, 2009

Abstract

Capturing meaningful interactions between a user and an application is useful. A meaningful interaction is an interaction that causes a change in state of one of the participants. Meaningful interaction capture is well established for console based applications. No techniques exist that capture only the meaningful interactions with a graphical interface. Data collected by such a system could be used for a variety of applications, such as HCI studies, authorship identification, cognitive modelling and screen recording.

A methodology for capturing the meaningful interactions between a user and a graphical application is described. An implementation of this methodology has been developed, together with a supporting tool-set. A new corpus consisting of captured interactions between users and two applications with contrasting graphical interfaces has been collected and published. This corpus is analysed and used for authorship attribution.

The use of this interaction capture system is evaluated as a high compression screen recorder. By using the interaction capture system as a screen recorder it is shown that the size of captured files are an order of magnitude smaller than the equivalent file created by a video based screen recorder.

Analysis of the captured corpus gives an overall accuracy of 83 percent when predicting the author of a stream. This is significant, showing that the way people interact with an application is unique.

Acknowledgments

I would like to thank my supervisor Bill who didnt let me give up.

The results in chapter 4 were generated in collaboration with Daniel Thomas from Bangor University.

A great deal of thanks go to my editor, who ensured that what I wrote down made sense.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Writing an application interface that is both efficient and user friendly is a challenge. Over the years the way a user interacts with an application has changed dramatically. As computers have become more powerful and cheaper, the methods of interaction have changed. Initially, interaction was entirely console based. However, this has evolved into today's modern rich graphical applications. As a result of these changes, computing has become ubiquitous. The user has moved out of the laboratory and into the home and office. Application interfaces have also evolved as part of this movement.

For a developer, writing an effective interface is a vital part of producing an application. Techniques have been developed to study the effectiveness of an interface in terms of ease of learning, efficiency and ease of use. These studies are expensive and time consuming, putting them out of the reach of many developers, but they are currently the only available method to evaluate the use of a graphical application. The evaluation of interactions with console based interfaces presents less of a challenge than graphical applications. This is because the interactions take place as a stream of characters in a natural language. The automated analysis of these streams can be carried out using techniques developed for analysis of character streams. These techniques have been developed over many years and are still the focus of ongoing research.

This thesis aims to bring together the analysis of interactions between a graphical application and character streams.

1.2 Thesis Statement

This thesis proposes that the interaction between a person and a graphical application is as unique as their writing style. In order to test this hypothesis, the existing and novel algorithms that perform well at the authorship task on natural language documents will be tested on graphical desktop applications. A large body of work exists that demonstrates that an author's use of language is unique. This fact is exploited by algorithms to determine authorship in written documents. In order to perform this task on a graphical application, it will be necessary to develop a method to capture the interactions as a stream of symbols. It will then be possible to perform the authorship task on the captured streams using the same algorithms as are used for natural language documents. The performance can then be evaluated to test the proposal.

1.3 Reasons for Attempting the Authorship Task

The task of authorship attribution has been chosen for the following reasons:-

Performance Excellent results at the task of authorship with natural language documents have already been achieved.

Existing Work There is a large body of research in this area to draw on.

Practicality Authorship does not require specialist equipment to evaluate it.

Matching The tasks of identifying the author of a document and the author of an interaction stream are obviously analogous.

1.4 Aims and Objectives

The main aim of this thesis is to:

> perform the task of authorship attribution on a stream produced by capturing interactions between a human and a computer.

The objectives are:–

1. To examine the history of Human Computer Interaction and Text categorisation. Overlaps between these two areas will be examined.
2. To design a system to capture interactions between a human and a computer.
3. To implement the capture system after first choosing a suitable language and framework.
4. To evaluate other uses of the capture system such as playback and visualisation techniques.
5. To create a corpus consisting of interactions between a human and different applications.
6. To perform the authorship attribution task on the new corpus.
7. Show that the meaningful interactions have been captured by using the capture system as a screen recorder. The performance of a traditional screen recorder will be compared to the capture system.

1.5 Thesis Contributions

The major contributions made by this thesis are outlined below:-

- A rationale for capturing interactions between a human and a graphical application has been devised;

- A technique to capture and record meaningful interactions with an application has been described and implemented. As part of this a tool set has been produced;
- A new corpus has been collected and published — the Bangor User Interaction Stream corpus (BUIS);
- A new character based text categorisation algorithm has been developed and published Hunnisett & Teahan (2004);
- A highly efficient lossy screen recorder has been developed;
- The following publications have been made as a result of this thesis:

  Hunnisett, D. & Teahan, W. (2004), Context-based methods for text categorisation, *in* M. Sanderson, K. Järvelin, J. Allan & P. Bruza, eds, 'SIGIR 2004: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval', ACM.

  Brooks, R., Hunnisett, D. & Teahan, W. J. (2007), 'A practical implementation of automatic text categorisation and correction for the conversion of noisy ocr documents into braille and large print'.

  Teahan, W. J., Thomas, D. & Hunnisett, D. (2009), Protocols for stream-based text categorization. Submitted to ECIR09.
- A collection of stream visualisation tools have been built.

1.6 Thesis Summary

Chapter 2 is a discussion of the background of human computer interaction. Following this, chapter 3 discusses the background of text categorisation research. It also contains an examination of the existing body of work that combines the two fields. From this new areas of further study are identified. A detailed evaluation of text categorisation algorithms is made in chapter 4. In addition a new algorithm for text categorisation is introduced.

A description of the design objectives for an interaction capture system is given in chapter 5. Chapter 6 details the implementation of this capture system in Java.

Methods of visualising streams of symbols are described in chapter 7. The creation of a new corpus (the BUIS corpus) is documented in chapter 8. This corpus contains User Interaction Streams captured by users interacting with two very different applications - WEKA and Asteroids. The BUIS corpus is analysed in chapter 9. This chapter looks at the similarities between a language and the captured user interaction streams. The authorship attribution task is then carried out on the corpus. Chapter 10 shows how the interaction capture system, implemented in chapter 6, can be applied as a compression system similar to a screen recorder.

Finally, chapter 11 discusses the contributions made by this thesis to computer science and suggests possible avenues for further research.

Chapter 2

Human Computer Interaction Background

2.1 Introduction

It is hypothesised that a user's interaction with an application is as unique as an author's writing style. This chapter examines the existing research into Human Computer Interaction (HCI). An understanding of the field of HCI is necessary to build a system that captures meaningful interactions between a human and a computer.

The chapter will provide a reader with a broad understanding of the key features of Human Computer Interaction. It will begin by defining what is meant by Human Computer Interaction. A rationale for studying HCI will be given. The links between HCI and Software Development will be explored. The methodologies for performing HCI experiments will be discussed. Finally, the role of Cognitive Modelling in HCI will be explored.

2.2 Defining Human Computer Interaction

Human Computer Interaction (hereafter referred to as HCI) has no formal definition. In coming to a common understanding of what is meant by the term HCI in the context of this research, the definition given by the Association for Computing Machinery (ACM) *Association for Computing Machinery* (1947) Special Interest Group on Computer-Human Interaction (SIGCHI *ACM's Special Interest Group on Computer-Human Interaction* (2008)) provides a useful starting point for discussion. They define HCI as follows:

> *Human-computer interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them.*

This definition will be examined in more detail and the implications of the meaning of the term HCI will be discussed in depth.

At a basic level, a more in depth understanding of "Human Computer Interaction" can be made by examining the OED definitions of the individual words:

Human of or belonging to the genus Homo.

Computer an electronic device for storing and processing data, typically in binary form, according to instructions given to it in a variable program.

Interaction a point where two systems, subjects, organizations, etc., meet and interact.

From these definitions, it can be said that HCI is the study of the meeting point between a human and an electronic device operating on a program. However, as the meeting point and interaction is not specified it is necessary in terms of this thesis to provide further clarification. For an interaction to take place between a human and a computer, the state of one of the meeting participants has to alter in some way. On the human side, the interaction must affect one or more of the senses. On the computing side, an interaction changes the state of the software.

Historically, the interaction between a human and a computer was mainly via a teletype console. The development of more powerful computers and display devices resulted in a revolutionary paradigm switch in Human Computer Interaction. The revolution was started during research at (*Xerox Palo Alto Research Center* 1999).

The result of the research was the (*Xerox Alto* 1972), the first computer that used the Desktop Metaphor as an interface. The desktop metaphor relied on the use of a mouse to manipulate a virtual desktop. The desktop metaphor is also referred to as the *window, icon, menu, pointing device (WIMP)* interface.

As computing power has increased, the size of the devices has shrunk. Computers have now become ubiquitous. Ubiquitous computing has resulted in ubiquitous Human Computer Interaction. The type and kind of these interactions is drastically varied, from operating a modern washing machine, driving a car, making a withdrawal from an ATM to ordering cinema tickets. All these activities are examples of HCI.

In conclusion, the term HCI has a very wide ranging definition. For the purposes of this research a more narrow definition will be used. Throughout this thesis, the term HCI will be used only to refer to interactions between a user and a desktop application.

2.2.1 The Evolution of HCI

As the power of computing increases, the interactions between the increasingly complex hardware and software becomes more and more complex. Instead of interacting with a computer using a phone keypad to navigate a menu, for example, modern systems now enable the computer to *understand* from spoken words what a user has requested. New hardware facilitates different interactions. Haptic devices such as the PHANTOM Omni Massie & Salisbury (1994) allow a user to feel interactions. Another example of the changing face of user interactions are in Heads Up Displays (HUD). These have long been used in aerospace but as the costs to build and implement these systems has been falling HUD's have been finding there way into other devices, used by a wider range of people, for example in cars and bike glasses.

Augmented and mixed reality — i.e. the mixing of the real world and computer generated information — is a very active field of HCI research. Again, the lower cost of hardware has enabled this to propagate to consumers. Sony's *Eye Toy* (2008) allows a user to augment reality in real time. Other types of augmented reality gaming, such as those described by Kim et al. (2008), where a plot for a game is delivered through a large variety of different media, for example the World Wide Web, email, phone and print media, are upcoming areas of interest for HCI research, due to their increasing popularity.

Table 2.1 shows how the user and method of interaction with a computer has evolved over time. It is based on the table that is part of the work on non command interfaces Nielsen (1993).

Table 2.1: The Evolution of Interaction Types

Generation	**User Type**	**Interaction Hardware**
1945-1955	Inventor	Punch Cards
1955-1965	Technocrats, Computer Scientists	Glass Terminals
1965-1980	Domain Specialists	Full Screen Terminals
1980-1995	Home and Office	WIMP
1995-present	Ubiquitous	Desktop Search and the Web

2.3 Rationale for Studying HCI

As the complexity of interactions between users and computers has changed over time and the types of user has changed, the need to develop effective interfaces has become not only more important but also more feasible.

Human Factors Society (1984) identified key areas where improvements to interfaces can be beneficial:-

2.3.1 Performance Enhancement

By improving interactions, users are better able to use applications more efficiently, learn new applications more easily and therefore perform tasks quicker. Where the application interface has been improved, performance can also be enhanced, as less cognitive resources are required to operate the application .

2.3.2 Resource Conservation

Improved interfaces can reduce both the number of people and the power of the equipment needed to perform tasks. An extreme example of this is shown in world wide patent number 010689 SMITH (1995). This patent describes how better interfaces can reduce the flight crew of an aeroplane from three to two. Interfaces that are more intuitive will also need less resources for training.

2.3.3 Acceptance

An application that is easy to learn and quick to use will be more readily accepted by its users than a complex, difficult to use application. The user experience level (both domain specific and general computing skill) and type of application also play a large part in acceptance.

2.3.4 Cost Reduction

It is clear that by reducing the amount of time and the number of resources needed to use an application, the cost will be reduced. This is correlated with both performance enhancements and resource conservation. Unfortunately, in most commercial applications the cost savings are made by the user, not the developer. This can (and does) reduce the incentive to the developers to improve the interface. However, in a competitive market, having a good interface can be a major selling point. The cost savings of better HCI are well known and significant. For example, an analysis of the cost benefits of an HCI study, followed up with interface improvements of an application used by an Australian Insurance Company in 1990, showed they had saved \$ (Australian) $536k$ a year Fisher & Sless (1990). The cost for the study and interface improvements together was less than \$ (Australian) $100, k$.

2.3.5 Human welfare

As has already been discussed, more and more time is spent interacting with computers in vastly different places. Computers are now common place in, for example, controlling cars, airplanes and medical systems. In all of these cases it is easy to see how a bad interface could result in harm to the operator or a third party. In short, bad interfaces can (and, in some extreme cases, do) kill. A graphic example of this can be seen in the following case study:-

Hospital Computerized Physician Order Entry (CPOE) have been shown to reduce medical errors Bates et al. (1998). Although the new systems reduced the occurrence of some errors the new interfaces also introduced different errors Koppel et al. (2005). Koppel et al describe how interface problems in CPOE systems have lead to "double dosing" errors where the patient receives both the previous and new dosage. Whilst this is clearly an extreme case, it does highlight the need to evaluate interfaces. A well-designed interface can improve a user's well being in subtle but important ways. For example, reducing mouse movements and key presses has been shown to reduce the occurrence of repetitive strain injury.

In order to improve an interaction, a developer must understand *how* and *why* a user uses an application. By understanding the user's task, a developer is better able to assist the user in accomplishing their goal. For example, where a user is blind or partially sighted, the study of HCI has resulted in improved layouts and hints for screen readers Theofanos & Redish (2003). This has enabled greater accessibility for users, who would otherwise miss some of the visual inputs and outputs provided by an application.

Shackel & Richardson (1991) describes the usability of an interface in ways that can be numerically measured. Page 25 of this book provides an operational definition of usability:

Effectiveness Are the users able to perform the tasks with an acceptable performance / error rate in all environments;

Learnability How difficult is it to train a new user, support and existing user and re-learn after non use;

Flexibility Is the system also usable for additional tasks and in new environments;

Attitude are the users happy to user the system.

From these identified categories it can be seen that there is a clear rationale for continued study of HCI.

2.4 HCI and Software Development

In this section, some of the existing tools for assisting developers with HCI are examined. There already exists a large quantity of quality work describing how a developer should write an application. This documentation is, however, only in the form of guidelines. There is no quick way of checking that a developer has been following the guidelines without running and evaluating the application. When non interactive applications, their are many tools, like PMD (Copeland 2005) and FindBugs^{TM} (Ayewah et al. 2007) that provide a simple way for a developer to check that they have been following the coding guidelines for a platform. Although tools like these do not catch every bug, they do provide another source of information for a developer to consider. The guidelines in HCI are far more lax, though in recent years some tools have started to be developed to assist a developer in this area. Dengo is an example of such an application *Dengo* (2006). It is described by the authors as follows:-

> Dengo is an application that allows a programmer to inspect GObjects live in their program and perform a series of tests to check compliance to the GNOME HIG.

GObjects are the objects that form the interface of a GNOME application. The GNOME Human Human Interface Guidelines (HIG) are a set of guides for a developer. They are discussed in more detail below. This project, however, has yet to release any code. Another similar tool is *GNOME Usability Analysis Tool* (2006).

> GUAT (GNOME Usability Analysis Tool) is an application that takes .glade files as inputs and summarises/evaluates the UI elements using the GNOME HIG.

This tool, like Dengo, has yet to release any code.

In addition to tools to verify the compliance of an application to HCI guidelines there has been some work into producing formal specifications for HCI. Jacob (1983*b*) describe two different techniques to formally describe an interface. This work has been extended (Jacob 1983*a*) this shows how the specifications can be executed to evaluate the interface specification.

2.4.1 Guidelines

One of the main products of HCI is guidelines for application developers. By following these guides, a developer is able to produce applications that look and behave consistently for the platform. An application that behaves consistently with other applications on the same platform will allow a user to complete their tasks quicker. This is due to being able to transfer skills and experiences to the new application. An excellent illustration of this is found in the 'save' function. Many applications have a save icon attached to a button on the toolbar. A user will expect that an application with a save button will save the current document. The guidelines are typically provided by the platform vendors, for example, the Apple human interface guidelines Apple (2004), GNOME Human Interface Guidelines GNOME (2008) and Top Rules for the Windows Vista User Experience Microsoft (2005). The interface design guidelines are also used as a basis for passing laws to enable equal access to web sites and applications to users who are disabled. The guides are produced by performing HCI experiments as described in section 2.5.

2.5 HCI Experimentation

There is a standard methodology for performing HCI experiments. The strengths and weaknesses of the standard methodology will be evaluated and other methods examined. It will be shown that although the standard method is effective and has many advantages, there are some disadvantages to using HCI labs. These disadvantages will also be discussed.

2.5.1 Standard HCI experimental techniques

The key feature of the standard HCI methodology is that it is laboratory based. The SIGCHI maintain a list of HCI laboratories around the world.
The usability lab photo gallery (2008) have a link to photographic tours of thirty two HCI labs in various places around the world.

In an HCI laboratory, typically participants must use an application in a dedicated laboratory, rather than at their usual place of work. The lab will be fitted with one way glass and video recorders. The user may be aware of being observed, but the observation is designed to be as unobtrusive as possible. Any comments they make are recorded by one of the observers behind the one way glass. These recordings are then analysed at a later date. The advantages of these facilities are as follows:

Eye tracker One of the facilities often provided by a HCI lab is eye tracking. This is used to monitor where, on a screen, a users is actually looking.

One Way Glass The user can be observed whilst using the application.

Video Recording Facilities Video recordings of both the screen and the user are taken. In addition to recording the screen, video cameras also record the movements and interactions between the user and the application.

Device Zoo A collection of hardware, both in the form of input/output devices such as haptic devices and also software are available.

Consistent Hardware All users will use the same hardware so the interface performance will be consistent.

Audio Recording In addition to recording videos of the users, comments made by the users are also recorded.

Playback Facilities Facilities to allow simultaneous playback of the separate streams captured as the users operate the application.

Dedicated Staff Staff are trained in the use of the hardware and techniques. Typically two researchers will perform the interaction study.

Screen Recorder Software and Hardware to record what occurs on the screen. Simple 2D applications (a typical desktop application) can be recorded using software. More complex 3D applications often require dedicated capture equipment to record the rendered image.

A description of the facilities available in the Pennsylvania State University HCI lab is shown in *Lab Resources* (2004).

In summary, the main advantages of having access to a HCI lab are:-

Dedicated Lab

Dedicated/Trained Staff

Specialist Hardware

Another benefit of the research undertaken in HCI labs is that results, in the form of User recordings, are sometimes made available so that developers and researchers can review the work. For example, the Better Desktop Project*The Better Desktop Project* (2006), provides a collection of videos that are produced by users performing a series of tasks using GNOME and KDE applications and are available to download from the Better Desktop Project's website*Better Desktop Data* (2008).

2.5.2 Disadvantages of a HCI Lab

The major disadvantage of an HCI lab is the cost. They are expensive because they require a dedicated room and specialist equipment. Users must also travel to the labs to take part in a study. For example, in 1988 the monetary cost of an HCI study was estimated at $\$128,330$ Mantei (1988). This was for a relatively small application, with only $32,000$ lines of code. To put this figure in context, WEKA (Witten & Frank 2005), used for analysis in this thesis, has approximately, $270,000$ lines of code. This does not include the additional functionality from modern graphical frameworks.

There are also very few developers who have access to these facilities. Another disadvantage concerning the use of HCI labs is that users can often find a visit to the HCI lab obtrusive and intimidating. As a result, their performance may well be different to their performance in a more naturalistic environment. As a result of these disadvantages, other techniques have been developed. These are referred to as Discounted or Guerrilla HCI Nielsen (1994*a*) studies, henceforth referred to as Discounted HCI.

2.5.3 Discounted HCI

As the name implies, Discounted HCI methodologies have been developed to alleviate some the costs (both monetary and time) incurred by using a dedicated HCI lab. The development of video-conferencing applications and cheaper bandwidth have led to the development of cheaper and less obtrusive HCI studies. Key features of Discounted HCI studies are that they will generally use the following resources:-

Screen Recorder Basic screen recording software is used to record interactions between the user and the application.

Web Cam A small low resolution camera is used to record the user whilst they operate the application.

Video Conferencing Video conferencing software is used to show the interactions and provide an audio stream to the researcher.

Although some of the inconvenience and costs of performing a HCI study are alleviated by performing Discounted HCI, there are still some significant shortcomings to these methods. A user must still use the software at a pre-determined time for the observations to take place. Although the user is in their natural environment, there is still a significant intrusion, as they will be aware that they are being watched as they operate the computer. There is also a cost associated in configuring the software and hardware for the study.

2.6 Cognitive Modelling and HCI

One area of very active research in the HCI area is in that of Human Information Processing. By producing a model of how someone interacts with an application, a developer can then use this model to design an interface that will assist the user in performing a particular task. This model also allows a developer to predict how their application will be used. This can be used to further improve the design. Given that a model is algorithmic, it is quicker and cheaper to evaluate an interface on a model, rather than build the interface then test it in a HCI lab.

2.6.1 Producing a Cognitive Model

An example of a cognitive model is Fitts law Fitts (1954) and Fitts & Peterson (1964). Fitts law describes how the time taken to select an object is related to both the distance and size of the target. Fitts law has been extended many times, for example in the work of Hornof & Kieras (1997). In this paper, the time taken to select an item from a menu was modelled. The resulting cognitive model can be used to estimate how long it will take a user to select an item from a menu. Soukoreff & MacKenzie (2004) reviews Fitts law, and how it applies to ISO standards.

2.6.2 User Taxonomy and HCI

The human component of HCI is extremely varied. Understanding the taxonomy of the users informs decisions about the interface design. User taxonomy is the process of categorising users. The categories typically used are expert, intermediate and novice. These groups can be further split. For example, a user may well be inexperienced at using a computer in general, yet may have a great deal of domain specific knowledge. Understanding the taxonomy of users allows a developer to target specific groups. For example, a user group may be defined by a variety of people, such as all card holders who access hole-in-the-wall ATMs. At the other end of the spectrum a user group may be a single user, only accessing an application after undergoing several weeks of intensive training.

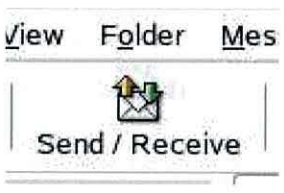
2.6.3 Task Discovery and HCI

An important piece of information for a developer is:-

What task is the user trying to accomplish?

Knowing what task a user is trying to accomplish enables the developer to improve the application by optimizing the interface. The terms "affordance" and "perceived affordance" were first described by (Gibson 1979), though the HCI community was introduced to the concepts by (Norman 1988). The term "affordance" is used to describe the actual function of an object. The term "perceived affordance" refers to what a user *thinks* the function of an object will be. An example of this is shown in figure 2.1. The button tells *evolution* (an email client for gnome platform) to

Figure 2.1: Evolution Button



send any mail in the outgoing mail and check for new mail (its affordance). When new users were set the task of sending a new mail, many of them clicked on the send part of the button marked 'send and receive' thinking that this would let them send a message (the perceived affordance). By making the software aware of the task that a user is trying to perform, either by inferring it from how they use the application or by simply asking them, the developer can either alter the interface to make accomplishing the task easier or guide the user through the steps needed to accomplish the task effectively.

2.6.4 Capturing Interactions

There has been some previous work in capturing meaningful interactions. This work has resulted in th

2.7 Conclusions

This chapter has examined the history and current research into Human computer Interaction. The difficulties for developers designing and implementing high quality interfaces have been examined. The benefit of better interfaces have been clearly shown. The next chapter will examine stream based categorisation in detail and outline how HCI and text categorisation have been linked in the past.

Chapter 3

Stream-Based Categorisation Background

3.1 Introduction

As stated in the thesis statement, the aim of this thesis is to show that:-

> perform the task of authorship attribution on a stream produced by capturing interactions between a human and a computer.

This section will explore a definition of stream based categorisation and how this applies to the problem of attributing authorship. The process of performing categorisation on text documents is more commonly referred to as text categorisation. This is the area of interest relevant to this thesis, which aims to use these techniques on streams produced from the interactions between a human and computer. The aim of stream based text categorisation is to analyse a stream of symbols and then assign it to one or more categories. In stream based categorisation the symbols that make up a stream are analysed sequentially. Natural language text categorisation can also take place by first extracting features (typically words) from the text. Feature based categorisation is less suitable for this research as the features are unclear and the categories depend on the problem domain.

Text categorisation research occurs within the wider area of research called Information Retrieval. Information Retrieval (IR), like HCI, has its own special interest group, SIGIR *Special Interest Group on Information Retrieval* (2008). Initial text categorisation research sought to perform language identification Cavnar & Trenkle (1994). These techniques have been extended to perform authorship attribution.

The rationale and practical applications of text categorisation will be explored in section 3.3.

Section 3.7.1 describes a small subsection of some of the types of data that can be represented as a stream of symbols. This selection is relevant to this thesis as these non natural language streams have been used to research stream-based categorisation using the same techniques and family of algorithms as are used to perform text categorisation of natural language. Section 3.8 will argue that text categorisation has been shown to be extremely effective at the task of authorship attribution.

3.2 Defining Text Categorisation

A simple definition of text categorisation is a follows:-

The process of assigning text to one or more pre-determined categories.

This is, however, a limited definition as it specifies neither how the assignment is made nor what the categories are. How the assignment is made is determined by the content and context of the text. What the categories are is determined by the problem domain. When performing categorisation, all other meta data, such as the publisher, is ignored.

A formal mathematical definition of text categorisation is given by the following binary pairing decision:-

$\langle d_i, c_i \rangle \in D \times C$

where:

d_i is a document from the set of documents $D = d_1 \ldots d_{|d|}$ and

c_i is a category from the set of $C = C_1 \ldots C_{|c|}$ categories.

The boolean assignment of true to $\langle d_i, c_i \rangle$ indicates that the document d_i is a member of the categories c_i. Conversely, an assignment of false shows that the document d_i is not a member of the category. As can seen be from this definition, a document can be a member of multiple categories.

Text categorisation is only concerned with assigning documents to pre-determined categories. The process of discovering the categories in addition to performing the categorisation is referred to as text clustering and is not of relevance to this thesis.

3.2.1 Multi-class and Multi-label

The formal mathematical definition above shows there is no restriction on the number of classes that a document can belong to. This is applicable to some problem domains, for example topic discovery, where a document may cover one or more topics. This is often referred to as Multi-class and Multi-label Luo & Zincir-Heywood (2005) classification.

3.2.2 Multi-class and Binary

In some problem domains, for example authorship where a document can only have been written by a single author, a further restriction can be added. This additional restriction ensures that a document can only be a member of a single category. This is often referred to as binary classification. When referring to the formal mathematical definition above, the following additional constraint is added:-

The boolean assignment of true to $\langle d_i, c_i \rangle$ implies that $\langle d_i, c_{!i} \rangle$ must be false. Where there are multiple different categories, this case is referred to as multi-class. If there is only one class, then the term binary classification is used as the decision is *true* or *false*.

3.2.3 Symbol Streams

The algorithms, rationale and experimentation details described in this thesis are all used to process streams of symbols. If the symbol stream is a document in a natural language then the symbol is more typically referred to as a character. The word 'character' in this context *is not* the primitive data type of a character. It is used to represent a symbol in a language. The word 'stream' refers to the fact that each of these symbols is ordered sequentially and the order is necessary to provide meaning.

3.3 Rationale for Automated Text Categorisation

The practical applications of automating text categorisation have long been recognised. Text categorisation is of obvious benefit when dealing with large quantities of text that would otherwise be impractical or prohibitively expensive to categorise by hand. As more text is digitised, the need for automated systems becomes ever more important.

Historically, one of the biggest drives for improved automated text categorisation has been information retrieval. The practical uses for text categorisation have grown from the increasing use of digitised documents in electronic libraries. Three important areas of research within information retrieval are: electronic libraries; text filtering; and word sense disambiguation. These three fields have provided the main drive to develop high performance text categorisation techniques. This will be explored in more detail below. As well as these motivations, it has become apparent that the techniques developed have other practical applications.

3.3.1 Electronic Libraries

As early as 1961, Maron (1961) described how text categorisation could be used to improve library services. This work has been widely extended (e.g. Kim (2005) and Wang & Desai (2007)) as the power of computer systems and the number of electronically digitised documents has increased.

Library systems rely on meta data to accurately retrieve documents. The meta data is used to describe the topics, author and other attributes of a document. The process of discovering and then adding the meta data by hand to existing documents is time consuming. Text categorisation has been used to automate the process of adding the meta data to existing and new documents.

3.3.2 Text Filtering

Over time, the amount of documents available in a digitised form has grown massively. At the same time the level of detail has increased. There is simply too much information to process by hand. For example, a search in the Bangor University Library for books that have the word "Elephant" in the title returns 24 books available for loan from the library. Performing the same search on Amazon.com returns 5,838 books. Users are mostly interested in documents that are relevant to them. Text filtering is a useful tool for sifting through this information to enable the user to fine-tune a search. So, for example, a personalised news delivery system that monitors breaking headlines that only shows a section of that news according to a pre-selected subject e.g. changes in stock prices, can be used by companies or individuals who own stock.

Text filtering has also been applied to the problem of unsolicited commercial email, better known as spam. These systems aim to categorise email into 'spam' and 'not spam' so that a user is not overwhelmed by having to read too many unwanted emails.

3.3.3 Word Sense Disambiguation

Text categorisation has been used to perform word sense disambiguation (WSD) for computational linguistics when encountering a homonym. When faced with a word with multiple meanings but different spellings, text categorisation enables a given word with multiple meanings to be selected according to context, by categorising the text to detect the context. WSD aims to identify the meaning of the word by using other words to provide context. For example the words 'scale' has the following definitions:-

- scales on a fish;
- an instrument for weighing;
- a measuring system;
- relative size;

- musical scale;
- to climb something;

As can be seen, the word 'scale' on its own is ambiguous but in any given context, for example 'she scales a mountain' its meaning becomes clear. WSD can also be used when performing speech recognition on encountering heterographs, i.e. words that sound the same but have different spellings. An example of a heterograph is *witch* and *which*.

> **Witch** This is also a homonym. The OED gives the definitions as a "a woman thought to have evil magic powers" and "an edible North Atlantic flatfish". These are both nouns.
>
> **Which** The OED defines which as a pronoun and adjective that is used for "specifying one or more people or things from a definite set".

By performing text categorisation, the correct meaning can often be established. This has been shown by Gale et al. (1992) where a category is created for each meaning and then categorisation is used to assign the word to the correct meaning.

3.3.4 Recent Developments

The problem domains described above have spurred the development of a large range of techniques. The techniques have been applied to a large variety of different problems. Some of these are described below. Text categorisation has also been applied to the problem of plagiarism detection. Since the invention of word processing, plagiarism has become easier, with the problem becoming even more pronounced with the proliferation of the Internet. Automated systems to detect plagiarism have therefore become increasingly desirable. Lukashenko et al. (2007) describes some of the techniques and tools currently available to detect plagiarism. This field is still an area of very active research. (Hoad & Zobel (2003) and Leung & Chan (2007)). The task of plagiarism detection is a natural extension of previous work that seeks to identify the author of a document Matthews & Merriam (1964). Text categorisation has already been used to identify the author of anonymous texts Stamatatos (2008) for criminal investigations. By categorising student essay submissions, a system to automate the grading process has also been developed Larkey (1998). This automated grading system has been further improved by Rosé et al. (2003).

3.4 Information Entropy

As described earlier, this thesis is concerned with symbol streams. This is referred to as a "message" in information theory. The "entropy" of a symbol stream describes its predictability or the amount of order. The OED defines entropy as "the degree of disorder or randomness in the system". Information entropy, as referred to in this discussion, is defined by the OED as "a logarithmic measure of the rate of transfer of information in a particular message or language." The information entropy is measured as the number of bits required to encode each character. This was first described by Shannon (1948) and is a measure of the information contained in a message. The more predictable the message, the lower the entropy. As the entropy describes the predictability of a message, it is also a measure of the best possible compression it is possible to achieve for that message. Information entropy is also referred to as Shannon entropy.

The standard measure of entropy is bits per symbol, i.e. the number of bits needed to encode each symbol. In this context, the term 'compression' is concerned with reducing the amount of space needed to store data. For example, to record the status of an event that has only two possible outcomes, each equally likely, such as a fair coin toss, this would take exactly one bit. If the coin was altered so that one side was more likely to be chosen, then less bits are needed. Compression is described in more detail below.

Formal Definition

Shanon entropy is formally defined as follows:-
Let X be a discrete random variable on a finite set $\mathcal{X} = \{x_1, \ldots, x_n\}$, with probability

distribution function $p(x) = \Pr(X = x)$. The *entropy* $H(X)$ of X is defined as

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log_b p(x). \tag{3.1}$$

It is taken that $0 \log 0 = 0$ as part of the definition. If the logarithm is taken in base 2 then it measures the number of "bits".

When looking at symbol streams, for the majority of these streams the optimal compression is not known. This is especially true for natural language. For a natural language such as English, the correct (optimal) Shannon Entropy is not known. As a result it can only be estimated. This is because the statistical model for English is not known. Current estimates of the entropy of English vary greatly. Shannon (1948) has shown empirically the entropy of English text to be between 0.6 and 1.1 bits per character. The best English language compression algorithm to date, paq8hp12 Ratushnyak (2007) (other compressors from this family are evaluated by Skibiński et al. (2005)) has achieved compression as low as 1.6 bits per character *50000 euro Prize for Compressing Human Knowledge* (2007). Compression of some English texts has achieved performance of 1.4 bits per character Teahan (1998). This is still higher than the estimates provided by Shannon, showing that there is still room for further improvements of computer models.

3.4.1 Compression

A definition of compression from the OED is: "To alter the form of (data) to reduce the amount of storage necessary.".

Shorthand is a good example of compression. It has been in use since the early Greeks Manuals (1935). Shorthand is a compressed representation of natural language that is fast enough to be used in dictation. Another example of compression is the telegraph. Here a style of writing was developed to minimise the number of words used. The booklet entitled "HOW TO WRITE TELEGRAMS PROPERLY" Ross (1928) describes how to use this compression system. Recently, similar techniques to optimize language have been used by mobile phone users when sending messages using the Short Message Service (SMS) where a message is limited by the specification to 160, 7 bit characters 3GPP (2007).

Compression of any type of data has always been very important. This has been driven by the desire to store more data and transfer it more quickly. Improved compression results in increased storage capacity and faster data transfer. Compression has been important long before the invention of the computer, but since the development of the electronic computer, compression has undergone a revolution. Within the field of Computer Science the term compression refers to two distinct fields of research:- lossy and lossless compression. Formally:-
A document a is compressed with the compression function C. This produces a compressed document a_c where $a_c = C(a)$. The compressed document a_c can then decompressed by function D to produce the decompressed document a_d where $a_d = D(a_c)$. The compession can be measured by the ratio of $\frac{|a|}{|a_c|}$.

Lossless Compression Lossless compression produces the same output after decompression as was used for the source. This means that a_d in the equation above is identical to the document a i.e. $a_d = a$.

Lossy Compression Lossy compression discards some of the data. In the equation above, $a_d \neq a$; the result of decompression is different to the original source. Examples of lossy compression are JPEG *JPEG File Interchange Format* (1992), and mp3 International Organization for Standardization (2005). When the data is evaluated, it is usually done so by a human who can *interpret* the data despite the missing information. The aim of the process of compression is to produce something that is similar enough to the original but with a very high compression ratio. Lossy compression is more commonly used to compress images and sound but not text.

Data compression can be measured by comparing the number of bits needed to store the uncompressed message with the number of bits needed to store the compressed data. This is called "the compresion ratio" and is typically measured in bits per character (bpc). This evaluation can be made for both lossy and lossless compression.

However, in the case of lossy compression the uncompressed message will not be identical to the compressed message.

3.5 Stream Based Algorithms for Text Categorisation

In this section, algorithms that are used to perform stream-based text categorisation are examined. These algorithms can be split into two distinct families:- those that operate on the character level and those that operate on the word level.

As these algorithms will be used later in this research to perform stream-based categorisation of symbol streams that are not natural language, only the algorithms that deal with categorisation of characters will be examined. These will be examined in three distinct groups:- Hidden Markov Models; Off The Shelf Compression Algorithms; and Count Based Measures.

3.5.1 Hidden Markov Model

A Markov Model is a statistical model for prediction. As the *true* model is unknown, the model and hence the state must be inferred by looking at the available data. Hence the term "Hidden" Markov Model. A Markov Model can be thought of as a state machine. From any state in the state machine there is a set of transitions to other states. Each of these transitions has an associated probability. The probability of a transition is determined not just by the current state but also the previous states. As has been noted earlier the *true* model is not known and the model must be inferred by training on available data, as demonstrated in the following example.

Using a limited alphabet of the letters $A|T|G|C$, the probability of each of these letters occurring in a stream is equally likely, $\frac{1}{4}$. With this knowledge, given a document such as $AAGTTACTAACATATTTA$, the document can be used for training. The table below (Table 3.1) shows contexts of length three that are found in the example document. From the table, it can be seen that if the context is AG then the next character must be a T. If the context is TA then there is a probability of

$\frac{1}{3}$ that the next character will be either C or A or T. The context TT, however, provides a probability of $\frac{1}{3}$ that the next character is T, but the probability of the next character being A is $\frac{2}{3}$. These probabilities form the transition probabilities for the Hidden Markov Model.

Table 3.1: DNA Contexts at Order Two.

Context	Potential States
AA	$G\|C$
AC	$T\|A$
AG	T
AT	$A\|T$
CA	T
CT	A
GT	T
TA	$C\|A\|T$
TT	$A\|T\|A$

3.5.2 Variable Order Markov Model

A Variable Order Markov Model (VOMM) is an extension of the Hidden Markov Model described above. Here 'Order' refers to the number of previous symbols that are considered to make up a context, as described below. Although called *Variable Order*, the word *Variable* can be confusing. Many of the algorithms that use VOMMs define a maximum order. i.e. the maximum number of contexts that should be considered. If a context is not found at this maximum order the algorithm then backs off or escapes to a lower order. Hence the term variable.

The Variable Order size helps to overcome the zero frequency problem. The zero frequency problem is when the encoder encounters a context that has not been encountered before. A Fixed Order Markov Model must default to the base model. So, for example from table above (Table 3.1) if the stream contains the sequence TCT this is a probability of $\frac{1}{4}$. A VOMM can look for smaller sections of the context at lower orders, in this case CT, thus providing a more accurate probability.

The following algorithms are also used for text categorisation. Each use the Variable Order Markov Models.

Prediction by Partial Matching Prediction by Partial Matching (PPM) is a technique used for compression. It was first described by Bell et al. (1989). The PPM compressor uses the previously seen uncompressed symbols in the stream to predict the next symbol. The number of previous symbols used to provide the context n is often added to the name, i.e. PPM_3 is PPM using a maximum context size of 3. The broad title of PPM refers to a large collection of algorithms. These algorithms build upon each other to improve compression performance using techniques such as smoothing and exclusions, described by Cleary et al. (1995). Moffat (1990) describes the implementation of PPMC. This is generally considered the standard PPM implementation.

PPM* PPM* is an extension the PPM family of algorithms as described in Cleary et al. (1995). This further improves the compression and does not have a fixed maximum order size.

Context Tree Weighting Method The Context Tree Weighting Method (CTW) described by Willems et al. (1995) uses a combination of many VOMMs. CTW operates on binary trees. To compress character streams, the characters must first be encoded into binary before the CTW algorithm can be applied. This decomposition process, as described by Volf (2002) can be tricky, as in many cases the position of the bits within a byte is significant. By discarding the positional information, performance is detrimentally affected. A number of approaches to performing the decomposition have been tried, for example, the technique described by Tjalkens et al. (1997). Volf (2002) describes and evaluates a number of other decomposition techniques. From this evaluation it is clear that CTW-DE, as described in Volf's thesis, has the optimal performance.

Probabilistic Suffix Tree Probabilistic Suffix Tree (PST) uses a single VOMM of fixed maximum length. After the initial VOMM is constructed, it is filtered and only *meaningful* contexts are retained in the model. A context is said to meaningful if its probability is greater than a threshold set by the user *and* the probability of context is *significantly* different to the probability of its *parent*. Here, significance is defined as $< \frac{1}{userthreshold}$ or $p > userthreshold$.

3.5.3 Off The Shelf Compression Programs

All of the above algorithms have been used to perform compression and can therefore be used for text categorisation. A description of how compression algorithms can be used to perform text categorisation is given in section 3.6. However "off the shelf" compression programs are very popular as they do not require a researcher to implement the algorithm. The program contains a tried and tested implementation ready to use. Some of the compression programs, such as rar Scheurer (2005) use algorithms already outlined above. Below, some of the other popular "off the shelf" compression programs are described.

Lempel-Ziv-Welch The *compress* and associated *uncompress* commands have been part of the BSD distribution since version 4.3 was released in 1986. This program performs compression using the Lempel-Ziv-Welch (LZW) algorithm. The implementation and algorithm is described by Welch (1984). The output of the compressor is also used as a specification see RFC1950 Deutsch & l. Gailly (1996). The algorithm operates using a sliding window.

gzip `gzip` is a compression algorithm developed by the GNU project because of patent problems with LZW. The format of the output is specified by RFC1952Deutsch (1996) and the algorithm is described by Gailly & Adler (2008). Like LZW, `gzip` also uses a sliding window. The performance of gzip is generally better than LZW.

bzip2 `bzip2` is a block based compression algorithm. It uses a Burrows-Wheeler system to sort the blocks used for compression. The compression performance is better than gzip but is achieved at the cost of memory and CPU. The algorithm is described in detail by Fenwick (1996).

RAR `RAR` is technically a file format and is patented. Internally, RAR has a number of compression algorithms that are selected based on the content of the data being compressed. It is chosen by researchers as it provides excellent compression performance when dealing with character streams. This performance is due to the internal use of PPM.

The algorithms and programs described above are all used to perform data compression. The compression is achieved by either *predicting* what is likely to occur (the VOMM family of algorithms) or by pointing to previous repetitions of the same sequence (algorithms like gzip.) The methods for using both these types of algorithm to perform text categorisation, along with a description of algorithms where the primary purpose is categorisation, will be examined in the section below.

3.5.4 Count Based Measures

The following algorithms have been developed primarily to perform categorisation. Unlike the algorithms discussed so far, they do not aim to compress the data.

R Measure

This is a normalized count based on the occurrences of common sub strings between the testing and training documents. It is described by Khmelev & Teahan (2003). The formula for the R Measure is as follows:-

Given a collection of n documents, each document (D_n) can be considered as a set of strings $S_k = S_k[1 \ldots |S_k|]$ where $|S_n|$ is the length of the document S_n. The R Measure is defined as a measure between document D and the remainder of the collection. It is defined as:-

$R(T|T1, \ldots Tm) = \sqrt{\sum_{k=1}^{l} Q(S[k \ldots l]|D_1 \ldots D_N)}$ where $l = |D|$ is the length of document D , $S[k..l]$ is the kth of document D and $Q(S|T_1, \ldots, T_m)$ is the length of the longest prefix of S, repeated in one of documents $D_1, \ldots, D_n$.

Other Categorisation Techniques

There has been previous work using techniques such as Neural Networks, Bysian classifiers and Support Vector Machines(SVM) to perform text classification. These techniques all require feature extraction to occur before classification. When performing categorisation of natural languages the features typically used are words of these techniques, SVM has the highest performance at categorisation text Yang & Liu (1999). The performance of SVM is however no better than the best stream based categorisation as shown in Teahan & Harper (2003*a*).

3.6 Using Compression Algorithms to Perform Categorisation

In this section, techniques that allow compression algorithms to be used for text categorisation are discussed. All these techniques rely on the assumption that documents from the same category will be similar.

For the purpose of this explanation the following set of assumptions are made:-

- there exists a set of training documents and a set of categories;
- each category will contain a minimum of one training document;
- there is no limit on the number of training documents that can be assigned to a category;
- each training document however can only be assigned to a single category;
- there is a single testing document.

The aim of the process is to use compression algorithms to determine which category the testing document should be assigned to. For the purpose of this explanation, it will be assumed that the compressor is an off the shelf compression programme and as such is treated as a black box.

A technique to use compression algorithms to perform categorisation is described by Benedetto et al. (2002) as follows:-
A document is produced by concatenating the testing document with each training document in turn. This new document is then compressed. When using a prediction based algorithm, when the boundary between the *correct* and *incorrect* document is encountered the prediction mechanism will become less accurate. The algorithms that look for repetitions will be unable to find repetitions, or the repetitions will be shorter. As a result of either of these events, the compression performance will be detrimentally affected. After compressing the testing document with each of the training documents, the compression ratios are again compared and the lowest ratio is selected. This technique measure the relative entropy of the two documents. Using compression algorithms has been shown to have excellent performance at categorisation. (c.f. Teahan & Harper (2003*b*)).

3.6.1 Minimum Descriptive Length

Minimum Descriptive Length (MDL) is documented in detail by Grünwald (2007). MDL describes the principle that:-

> the more we are able to *compress* a set of data, the more regularities we have found in it and therefore, the more we have *learned* from the data.

An alternative way to consider this is that MDL represents a formalised description of Occam's Razor. This represents an information theoretic approach to machine learning.

3.7 Experimentation

In this section the variety of experiments that stream categorisation have been applied to are described. First, the variety of streams types are examined. This is followed by a discussion on the different categories evaluated.

3.7.1 Stream Types

The list below is a non-exhaustive list of the types of streams that text categorisation has been applied to. As can be seen from this list, there is a wide variety in the types of stream that have been categorised.

Natural Language Documents written in any natural language such as English, French, Chinese etc.

DNA DNA sequences can be categorised using the same set of algorithms. DNA sequences are limited to the alphabet *ACTG*. The commonly used software GeneMark^{TM} (Borodovsky & McIninch 1993) and *GENSCAN* (1997) both use Hidden Malkov models to perform categorisation of gnome sequences. The algorithm used by GENESCAN is described in Burge & Karlin (1997).

Music Hidden Marklov Models have been used to categorise music. For example, Chen et al. (Oct. 2006) use text categorisation to identify musical genre.

3.7.2 Categories

Below is a description of some of the different types of categories used for text categorisation. The list is not exclusive and obviously has a great deal of overlap with the rationale for text categorisation, described in section 3.3.

Authorship Attribution Authorship attribution is the task of determining *who* is the author of a given document, from a choice of many potential authors. This has many practical uses, one of the major ones being plagiarism detection. The same techniques that are used for language detection are also used to determine authorship. As this is the subject of this thesis a more detailed discussion will take place in section 3.8.

Language Identification The goal of language identification, given a text of unknown language and a set of training documents, is to identify which of the training documents is most closely related to the test document. To do this, one or more of the testing documents will have had to have been written in the same language as the test document. In the case of languages that use different alphabets, this task is trivial. However, many languages share alphabets. When two or more languages share the same alphabet, such as English, French and German, then the task of identifying the language is no longer trivial.

The task of identifying the language of a document was shown to be feasible and highly accurate by Teahan & Harper (2003*b*). It has been shown to be successful even when the text is noisy by Cavnar & Trenkle (1994). Further developments in language categorisation have targeted the problem of identifying the language of individual blocks of text within a document, rather than the entire document.

Dialect Detection Dialect detection is a natural extension of the task of identifying the language of a document. When performing dialect detection, not only is the language identified but also the dialects within the language Huang & Hansen (15-20 April 2007).

Topic Identification Given a document, text categorisation is used to determine what the subject(s) of the document are. There may be more than one subject per document. For example, this technique has been used to detect student essays that are off-topic Higgins et al. (2006).

Genre Identification This was demonstrated by Kessler et al. (1997) and more recently by Lee & Myaeng (2002) and has been further developed by Ferizis & Bailey (2006). McCallum & Nigam (1998) show how Bayesian Classifiers can be used to classify the genre of news groups.

Age A similar task to identifying authorship, the aim here is to categorise the text based on the age of the author.

Sentiment Text categorisation has been used to determine the sentiment of an author on a given subject. For example, Pang & Lee (2004) describe a system to identify positive or negative opinions of a movie. This has since been extended by the same group Pang & Lee (2005) to try and assess a movie review in terms of the number of stars that the reviewer would have assigned the film.

Bias Text categorisation has been used to determine the bias of an author towards a subject.

SPAM Email spam has become a massive problem in recent years. There is an ongoing battle between the spammers and end users and developers trying to keep their inboxes free of junk. Text categorisation is ideally suited to the task of sorting mail into *spam* and *not spam.* There is a large body of work in this area and there has been a spam track as part of TREC from 2002 till 2007 *SPAM Track Guidelines* (2005 - 2007).

Initially, basic regular expressions were used to identify spam. This solution was quickly broken by spammers, who introduced random misspellings for example "v1agra" rather than "viagra". Developers fought back by developing Baysian classifiers Graham (2004) but again these were defeated by spammers. Recent developments in the Trek 2005 spam track Bratko & Filipič (2005) and more recently by Bratko et al. (2006) have shown that PPM is excellent at performing the task of spam identification.

Gender The task of identifying the gender of the author of a document is another area where text categorisation is useful. For example, Corney et al. (2002) show how the gender of the author of an email can be determined by performing text categorisation, as have Koppel et al. (2003).

Opinion It has been shown by Pang & Lee (2008) that opinion can be categorised.

Computer Intrusions Warrender et al. (1999) describe a system that uses Hidden Markov Models to detect computer intrusions.

3.7.3 Text Categorisation Corpora

There are a number of standard corpora that are available for use in evaluating text categorisation. Each corpus consists of a collection of text documents. Each of the documents within the corpus has been analysed by a human and tagged. Part of the tagging process involves assigning the document to one or more categories, such as the author or major topics. The corpora are publicly available, so the performance of a new technique can be evaluated against known data. This thesis will make use of two published corpora, the Stig Johansson (1978) and the *REUTERS CORPUS* (2000). These are both freely available corpora that have been used extensively in the past and so have been well tested.

3.8 Authorship and Text Categorisation

As described in the thesis statement, the aim of this work is to show that the streams produced by the use of a graphical application behave in a similar way to streams of characters in a natural language.

Given that the task chosen to show this is that of authorship discovery, it is important to examine in more detail the current state of research in this area. (n.b. this thesis is concerned only with the task of determining the author of a document. It is not concerned with authorship verification, which is a related area of research.)

When performing authorship discovery, the following assumptions are usually made:-

The Author is Present in the Training Set The set of training documents contain at least one document written by the author to be tested.

Unique Authorship Each document has only one author. As such, each document will only appear once in the training set and the testing document will only be assigned to a single category.

From these assumptions it can be seen that the task of determining authorship will be a multi-class classification.

The corpora that are used to test algorithms when performing this task contain many authors and as a result many more documents. Although each testing document is processed individually, the results are compared over the entire set. From this, the performance can then be evaluated, by comparing the number of correct assignments with the number of incorrect assignments. This gives a percentage accuracy. This percentage accuracy can be split further to show the accuracy by author.

As well as the percentage accuracy, the other statistics used to evaluate the performance of the classification are *Recall* and *Precision*. These are calculated as follows:-

Recall Recall is the number of documents correctly assigned to the author, divided by the actual number of documents written by that author.

Precision Precision is the number of documents marked as being written by the author which actually were written by the author, divided by the number of documents marked as being written by that author.

Historically, the problem of determining authorship has been of interest since at least the turn of the last century. Holmes (1998) provides an excellent summary of the history and development of algorithms up to 1998. Initially, categorisation was undertaken by hand and it was not until the 1960's and the analysis of the Federalist papers that these techniques achieved widespread recognition as documented by Mosteller & Wallace (1966).

The field underwent steady improvements until the early 1990's, when computing power had increased so much that new algorithms had to be developed to perform authorship attribution. These new algorithms came from the field of Artificial Intelligence, where Neural Networks in the form of Multi-Layer Perceptions were used. These built on the previous work in the area and used words to perform the categorisation. A problem with this technique was that the documents needed to be segmented and significant features identified by hand.

By performing the categorisation on a character level, a number of problems with the traditional approach can be avoided. These are the problem of segmenting the data and the problem of identifying which features are significant.

As this thesis is concerned with performing authorship on document streams that are not a natural language, the techniques used to perform authorship on a character level will be of more relevance than those that perform on a word level. As shown above, the performance of character level algorithms has now reached, and even surpassed, the performance of word based algorithms. Therefore, the use of character based algorithms to categorise streams of symbols of non natural language has a sound theoretical basis.

3.9 Combining HCI and Text Categorisation

The move away from console applications has resulted in an almost complete split between HCI and Information Retrieval. When the data is available, for example, web site traversal and text based searching, there has been some crossover between the two communities. Section 3.9.2 will examine how text categorisation has been applied to console applications.

3.9.1 Predictive Text

Many mobile phones have, for some years now, supported and encouraged users to use a facility marketed as predictive text. As a phone keypad is limited to twelve keys, each key has a minimum of three and a maximum of four letters assigned to it. A user picks the correct character by pressing the relevant button repeatedly. This is a very slow way of entering text. Repeated letters were particularly difficult, as the user had to wait for the current selection to timeout before starting on the same button. To speed up the inputting of text, manufactures have implemented a system known as T9® predictive text James & Longé (2000).

T9 uses a Bag of Words model. The user only has to press the button that contains the desired character. The computer then looks up from a pre-defined dictionary [1] all the possible words that could match the potential characters entered so far. The user then selects the correct word, by cycling through the possibilities. More recent developments have introduced new devices, such as the iPhone. These have relatively large, but not full sized, keyboards. The computer predicts what word a user is trying to type and also corrects minor spelling mistakes.

[1]The dictionary is often expandable by the user, who can add new words

3.9.2 Console Based Applications

Console based interfaces, for example `tcsh` and `bash` FSF (2006) are by their very nature text based. They provide and store a history of previously entered commands, allowing a user to edit these commands before having the new command interpreted by the computer. The commands inputted by the user, and the respective output to the user provided by the computer, are character streams. These character streams are an obvious target for text categorisation. The analysis of these character streams has been used to improve interfaces and also to perform verification of the current user.

3.9.3 User Interface Improvements

As early as 1982 developers were examining methods to improve interfaces. For example, Witten (1982) describes how an interface could *guess* what the user was trying to type and allow a user to select the guess, rather than typing the entire command. This has been extended to build a *Reactive Keyboard* Darragh et al. (1990). This provides a faster input mechanism for typing, by predicting what word is going to occur next. These experimental input enhancements have now become a standard part of modern software. Modern integrated development environments (IDEs) such as Eclipes Eclipse Foundation (2008*a*) and Netbeans Strobl (2008) provide context sensitive auto completion. Emacs has further expanded this concept to develop a feature called Hippie Expand *HippieExpand* (2004) which provides suggested completions based on locality. The completion feature has also been added to word processors such as OpenOffice *OpenOffice.org 2.x Writer Guide, Using word completion* (2008).

It has been known for a long time that users will repeat many of their actions. Further work has examined repetition and the effectiveness of the history feature of command lines by Greenberg & Witten (1988). This knowledge has resulted in the inclusion of interactive command line histories.

3.9.4 Authorship

Character analysis techniques have been used to identify users impersonating other users. This technique was started as long ago as 1988 by Leggett & Williams (1988) by using statistical techniques. It has also been used to provide an authentication system by monitoring key strokes (Monrose & Rubin (1997) and Coull et al. (2003)). Recent developments have shown increased accuracy by using support vector machines Seo & Cha (2007). The previous work in this area has concentrated on identifying authorship to show *who* was typing commands. The knowledge of who is using a computer is important. It can provide information for prosecutions in cases of intrusions. It can also be used to detect a user masquerading as a different user. Another valuable function of identifying authorship is showing whether commands are being executed by a computer program or a human. To do this a combination of timing and command analysis has been shown to be effective see Alata et al. (2006).

3.10 Graphical Applications

There has been much less work that combines categorisation and graphical applications, although in recent years this has started to change. The main cause for the lack of work is the difficulties in capturing the interactions. Pusara & Brodley (2004) describe how mouse movements can be used to authenticate users. The work shown by Garg et al. (2006) shows a capture system designed to capture user interactions and then categorise them for the purpose of performing authorship. This has since been extended by Imsand & Hamilton (2007). The capture system described in this work is inserted at the system level and captures all interactions with the computer.

3.11 The Missing Link

As has been described in this chapter, there is little work that combines text categorisation and HCI with graphical applications. At first this may seem to be because interactions are not text based. However, as shown earlier in this chapter text categorisation techniques have been used for a variety of different stream types. All that is needed to recombine these two areas is a method for capturing the interactions as a stream of symbols.

A Graphical Desktop application is, by definition, a program running on a Turing machine. As a result, it will be in one of many fixed set of states. For graphical desktop applications, the main mechanism by which states are changed is by interactions between the user and the application. These interactions are limited to input and output devices connected to the computer. This leads to the conclusion that there will be a language that describes the states of the application, where the transitions between the states are defined by interactions between the application and the user. If this language can be discovered then text categorisation algorithms can be applied. This will result in the categorisation of a graphical desktop application. Although the resultant stream is not a natural language, previous research has shown that text categorisation techniques can successfully be applied to other character streams with excellent results.

Chapter 4

Stream Based Text Categorisation

4.1 Introduction

As stated in the thesis statement, the aim of this thesis is to perform the task of authorship identification on captured interactions between a user and a graphical application. In order to do this, it is necessary to understand the techniques and tools used to undertake the authorship task in text categorisation. This chapter is an extension of the work published in SIGIR'2003 Hunnisett & Teahan (2004). This paper undertook an experimental analysis of several stream based text categorisation algorithms on a standard text corpus. It demonstrated that the optimum compression model size was not the optimum model size for categorisation. This chapter extends this work and also introduces four different protocols for performing text categorisation (section 4.2). These four protocols are applicable to all stream based text categorisation algorithms, not just the new algorithm the C measure. The performance of these protocols are then examined (section 4.4.3).

4.2 Protocols

In this section, four distinct protocols that can be used to perform text categorisation are discussed. Each of these protocols is applicable to any stream based text categorisation algorithms (see section 3.5 in chapter 3). Three of the four protocols have been identified and described by Marton et al. (2005). The three protocols that they identify are named as:-

SMDL Standard Minimum Description Length. This will be refered to as Protocol I.

ADML Approximate Description Length. This will be referred to as Protocol II.

BCN Best Compression Neighbor. This will be referred to as Protocol III.

In addition to these protocols there is a fourth protocol. This protocol has not yet been tested and is described in section 4.2.3. ADML and BCN allow the use of algorithms without modification (i.e. off the shelf algorithms). As they are simpler to test, there already exists a large body of reasearch in this area. The differences between ADML and BCN are described below.

4.2.1 Concatenation verses Non Concatenation

It is usual to have more than one document from each different author for testing. Previous research has looked at the difference between comparing each training file exclusively with each testing file (BCN), and producing a single training file, by concatenating the training documents together (ADML). This single training document is then tested against each testing document.

4.2.2 Static and Dynamic Models

As with the difference between BCN and ADML, there is a simple difference between ADML and SMDL. The difference, in this case, is whether or not the model is *allowed* to change during the testing phase. If the model is allowed to change during the testing phase the model is said to be dynamic. A typical off the shelf compression algorithm, such as gzip or bzip, will be dynamic, as this leads to much better compression. Although the algorithm is the same for both static and dynamic protocols, it requires much more work to produce a categorisation system using a static model. The extra work involved is that of modifying an existing, off the shelf dynamic algorithm to be static. Once the system has been developed, however, it tends to perform better. (See the results in section 4.4.3.) A further benefit is that the algorithm is either much faster and more memory efficient.

4.2.3 The Fourth Protocol

The previous work, described above, clearly describes three protocols. As has been alluded to earlier, there is a forth protocol. By drawing the protocols diagrammatically, as shown in table 4.1 this fourth protocol can be clearly seen. This uses non concatenated training models together with a static model.

Table 4.1: The Four Protocols for Stream-based Text Categorisation.

	Static	*Dynamic*
Contcatonated	Protocol *I* (SMDL)	Protocol *II* (AMDL)
Non Concatonated	Protocol *IV*	Protocol *III* (BCN)

None of the previous research has clearly identified which of these protocols is optimal. To date, protocol *IV* has not been tested.

4.2.4 Document Concatenation

The concatenation of documents can be carried out in three distinct ways. One method requires concatenation to occur as a separate process, preceding the documents being processed by the algorithm. The algorithm then simply processes the document as a single stream of text. There is, however, an inherent problem with doing this. The boundary between document A and document B will introduce new contexts. For example, if document A consists of the text "abra" and document B consists of the text "cad", by concatenating A with B a training document, "abracad" is produced. However, if the documents are concatenated in the other order, i.e.B

then A, the training document becomes "cadabra". The following tables (table 4.2 and table 4.3) show the contexts at order size three for documents A and B.

Table 4.2: Contexts at Order 3 for Document A

abr
bra

Table 4.3: Contexts at Order 3 for Document B

cad

Table 4.4: Contexts at Order 3 for Document A Concatenated with Document B using Method One

abr
bra
rac
aca
cad

The second method of performing concatenation does not depend on the concatenation order. This is achieved by making the categorisation system able to load multiple files as part of the same model.

Table 4.5: Correct Contexts at Order 3 for Document A Concatenated with Document B.

abr
bra
cad

Table 4.5 shows the contexts that should be available from concatenating the documents together. If the command line utility cat is used (method one) then the additional contexts, highlighted in red in table 4.4, are added to the model. It is obvious that the number of additional contexts introduced by this concatenation process is related to the number of documents in the training set. The number of

new contexts will be $n(k-1)$ where k is the context size and n is the number of documents in the training set.

If the modification of the categorisation system is not possible (for example, when using an off the shelf compression algorithm) the additional contexts can be avoided by the use of a special "Sentinel" symbol, as described below. This is the third concatenation technique. By adding a marker to denote the end of the file (henceforth referred to as a Sentinel Symbol) then the *normal* `cat` can be used. This is explored in more detail below.

Again, using the same two documents the Sentinel Symbol $\emptyset$ can be appended.

Table 4.6: Contexts at order 3 for document A with a Sentinel Symbol

abr
bra
$ra\emptyset$

Table 4.7: Contexts at Order 3 for Document B with a Sentinel Symbol

cad
$ad\emptyset$

The introduction of the Sentinel Symbol has increased the size of the individual model by one. However, as the size of the models is unimportant for text categorisation, this is not a problem. Examining the table of contexts for the concatenated documents (table 4.8) it can be seen that there are still two new contexts, $r\emptyset c$ and $\emptyset ca$. However, both these contexts contain the Sentinel Symbol. When analysing a testing document, an analysis will cease when a Sentinel Symbol is encountered, so these extra contexts will never be used for prediction; i.e. although they are present in the model, they will never be encountered and therefore are not a problem for categorisation.

Table 4.8: Contexts at Order 3 for Document A Concatenated with Document B using cat with a Sentinel Symbol.

abr
bra
$ra\emptyset$
$r\emptyset c$
$\emptyset ca$
cad

4.3 C Measure

This section describes a new algorithm for performing text categorisation, called the C Measure. This has been developed as part of this thesis, first published by Hunnisett & Teahan (2004).

The C Measure is a simple context based measure, based on the presence of contexts within the training text. Only the maximum fixed order length is considered. Like PPM, this is a character based technique. However, due to the simplicity of the algorithm it is possible to produce a highly optimised implementation.

4.3.1 C Measure Algorithm

This section will describe the C Measure algorithm in detail. As has already been discussed, the C Measure is a context based algorithm with a fixed maximum order size. The development of the C Measure algorithm took place using two thousand Reuters News articles. The algorithm has two distinct stages, as outlined below. These are:- *a*) accumulating the count; *b*) the normalisation of the count.

Accumulating the Count

```
for context in testing document do
    if context found in training document(s) then
      count++
    end if
end for
```

Normalising the Count

After all the contexts in the training document have been analysed, the count is normalised. The normalisation process produces a score between zero and one. A score of one indicates that every context in the testing document was found in the training document(s). A score of 0 indicates that non of the contexts in the testing document were found in the training document(s). The normalisation processes takes place as follows:-

The count of matching contexts is divided by the length of the testing document minus the context size (as shown by the divisor in the formula below).

4.3.2 C Measure Formula

The formal definition of the C Measure is as follows:- let A be an alphabet consisting of N symbols, and text string $x^n = x_1, x_2, \dots, x_n$ where $x_i \in A$.
For a training document d_{train} and testing document d_{test} the C Measure at order k of the testing document d_{test} is defined as:-

$$C = \frac{\sum_{i=k}^{i=|d_{test}|} F_i()}{|d_{test}| - (k+1)} \text{ where}$$

$$\begin{aligned} F_i &= 1 \text{ if content } x_{i-k+1}, x_{i-k+2}, \dots, x_i \text{ is present in the training text, } d_{train} \\ &= 0 \text{ otherwise.} \end{aligned}$$

4.3.3 Variations

Many variations of this algorithm have been tested and shown to be none optimal. Some of the variations tested are:-

Minimum Occurrence This variation, rather than incrementing the count by one if the context is found, only increments the count if the context occurs at least n times in the training text. The normalisation is applied the same way as for the standard C Measure.

Count Incrementation In this variation, the count is incremented by the number of occurrences of the context in the training set. The normalisation of this algorithm is more difficult than the other variation. The same normalisation algorithm is used to test a document. The normalisation algorithm, however, no longer produces a number between zero and one. This normalisation algorithm does not take into account the size of the training set. However, when testing the variation, the training sets were of similar size so this was not thought to be a problem. Had the performance been equal to or greater than the C Measure, a normalisation algorithm that takes into account the size of the training set would have been developed.

4.4 Experimental Setting

Three different copora were used to evaluate the performance of the four protocols and the C Measure. The C Measure algorithm was evaluated against the Reuters corpus RCV1. The four protocols (discussed earlier, in section 4.2.3) were evaluated using two different corpora:- the twenty Newsgroups collection; and Gutenberg texts. These are the same corpora that were used by Marton et al. (2005). The same paper also discusses some of the protocols described in this chapter.

4.4.1 Reuters Corpus

The two thousand Reuters News articles *REUTERS CORPUS* (2000) were split into training and testing sets. A data structure was constructed for each of the fifty authors, based only on the training data. The data structure is suitable for use by each of the algorithms, allowing all the algorithms to be tested simultaneously. After the training is completed the data structure is frozen. Each of the testing data files is then tested against each of the authors, using each algorithm. The best prediction is picked for each author and checked against the correct answer. The following algorithms were tested:-

C Count New algorithm explained in section 4.3

$PPMD$ **without Exclusions** PPMD without exclusions.

$PPMD_e$ **with Exclusions** PPMD with exclusions

Exclusions

Exclusions in PPMD provide a way of improving the compression by not encoding probabilities for contexts that have already been seen. This leads to improvement in compression in the region of 1-2 percent as shown in the original paper describing PPM Cleary & Witten (1984).

Results

Table 4.9: Model size against Algorithm Reuters

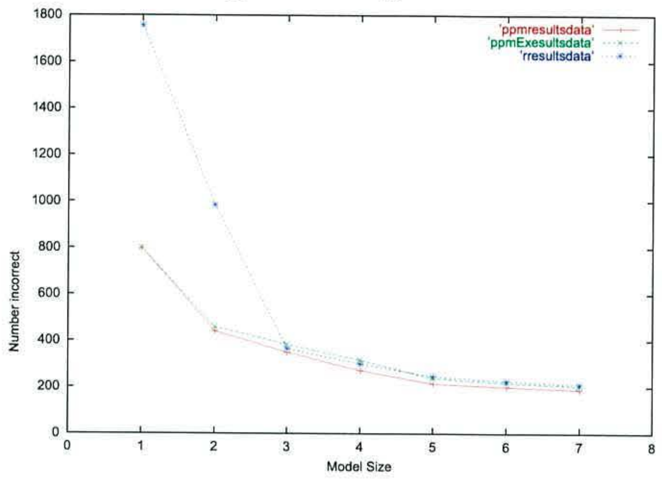
Order	$PPMD$	$PPMD_e$	C Count
1	0.602	0.602	0.122
2	0.772	0.772	0.509
3	0.809	0.826	0.818
4	0.843	0.865	0.851
5	0.882	0.893	0.877
6	0.892	0.900	0.887
7	**0.899**	**0.906**	0.895
8	0.896	**0.906**	0.898
9	0.891	0.904	0.904
10	0.891	0.902	0.908
12	0.888	0.901	0.907
12	0.886	0.896	0.910
13	0.884	0.891	**0.911**

Table 4.9 shows the performance of C-Measure and PPM both with and without exclusions. The protocol used is *I*. The highest performance for each protocol has been highlighted in bold. The overall optimum performance is 91 percent. This is produced by C-Measure at order 13. As can be seen in figure 4.1, all three algorithms start off with improved performance as the model size increases. Both of the PPM models have a maximum at order 7.

Performance

The C count performs very well. At lower orders the categorisation is quite poor. However, at higher orders the categorisation improves to surpass PPM without exclusions. The difference in performance between PPM with and without exclusions is very interesting, as it contradicts the assumption that better compression will lead to better categorisation.

Figure 4.1: Context Size against Categorisation Performance Reuters



4.4.2 NewsGroups and Gutenberg

This section will evaluate the performance of the C Measure against PPM and R Measure.

The two corpora will now be discussed in more detail:-

Twenty Newsgroups This corpus consists of 8998 postings to Usenet discussion groups. They vary in size between $71K$ and hundreds of bytes. The categories for this corpus are not the author but are the NewsGroup to which the message was published. The categories are:-

- alt.atheism
- comp.graphics
- comp.os.ms-windows.misc
- comp.sys.ibm.pc.hardware
- comp.sys.mac.hardware
- comp.windows.x
- misc.forsale
- rec.autos

- rec.motorcycles
- rec.sport.baseball
- rec.sport.hockey
- sci.crypt
- sci.electronics
- sci.med
- sci.space
- soc.religion.christian
- talk.politics.guns
- talk.politics.mideast
- talk.politics.misc
- talk.religion.misc

Gutenberg This corpus is a small subset of the works available from the Gutenberg project Gutenberg (1992). Ten well known authors, each with four books, were selected. This provides a corpus of forty documents. The documents are much larger than the NewsGroup corpus. The smallest is $19K$ and the largest $1.05M$.

The NewsGroup corpus was chosen because it allows the algorithms and protocols to be evaluated with categories other than author.

R measure

The R measure was developed by Khmelev & Teahan (2003). Like the C measure, it produces a normalised comparison between two documents and performs very well at the task of authorship. The R measure and C measure are very closely related. The R measure is the sum of the C measure at all orders Teahan et al. (2009).

4.4.3 Experimental Results

This section will examine the results of performing text categorisation using the four protocols and four algorithms previously described.

News Group Corpus

Table 4.10 shows the categorisation performance of the frequency based categorisers, C Measure and R Measure, with the Ten Newsgroup corpus.

Table 4.10: Categorisation Accuracy for Twenty Newsgroups using Frequency-based Methods.

	Protocol			
Method	*I*	*II*	*IV*	*III*
R-Measure	**0.953**	**0.951**	0.866	0.874
C_1	0.089	0.089	0.122	0.121
C_2	0.378	0.371	0.310	0.252
C_3	0.872	0.874	0.387	0.350
C_4	0.920	0.920	0.517	0.490
C_5	0.933	0.933	0.642	0.616
C_6	0.939	0.937	0.734	0.717
C_7	0.941	0.938	0.801	0.789
C_8	0.942	0.940	0.842	0.838
C_9	0.944	0.943	0.868	0.869
C_{10}	0.946	0.945	0.889	0.893
C_{11}	0.947	0.946	0.904	0.907
C_{12}	0.948	0.947	0.909	0.915
C_{13}	0.947	0.947	0.913	0.919
C_{14}	0.946	0.946	0.914	0.922
C_{15}	0.945	0.945	**0.915**	**0.923**
C_{16}	0.943	0.943	0.914	0.921
C_{17}	0.941	0.943	0.913	0.920
C_{18}	0.939	0.939	0.911	0.919
C_{19}	0.937	0.938	0.910	0.918
C_{20}	0.935	0.936	0.909	0.917

The column labelled "method" shows the method used where C_n is used to indicate the C measure with an order of n. The highest performing method for each protocol has been highlighted in bold. R Measure under protocol I is 95.3 percent accurate. This is the highest performace for the frequency-based measures. The highest C measure accuracy, 94.8, is again using protocol I at order 12.

Table 4.11 shows the categorisation performance using PPM both with and without exclusions.

Table 4.11: Categorisation Accuracy for Twenty Newsgroups using PPM Based Methods

	Protocol							
	With Exclusions				Without Exclusions			
Maximum Order Size	*I*	*II*	*IV*	*III*	*I*	*II*	*IV*	*III*
2	0.927	0.931	0.833	**0.866**	0.928	0.933	0.821	**0.848**
3	0.947	0.952	0.829	0.515	0.946	0.951	0.836	0.509
4	0.951	**0.954**	0.828	0.386	0.950	0.954	0.851	0.383
5	0.951	0.951	0.830	0.438	0.951	0.951	0.868	0.417
6	**0.953**	0.937	0.834	0.502	0.951	0.939	0.878	0.477
7	**0.953**	0.906	**0.836**	0.557	**0.952**	0.906	**0.885**	0.531

As can be seen from this table the performance of PPM is better than the frequency-based methods for the Newsgroups corpus. PPM achieves overall accuracy of 95.4 percent using protocol II.

Gutenberg Corpus

Table 4.12 shows the performance of the frequency based measures when performing text categorisation on the Gutenberg Corpus. As with Table 4.10, the method column indicate the order size n in the form C_n and the highest performance for each protocol has been highlighted in bold.

Table 4.12: Categorisation Accuracy for Gutenberg using Frequency-Based Methods

	Protocol			
Method	*I*	*II*	*III*	*IV*
R-Measure	0.525	0.450	0.475	0.475
C_{11}	0.575	0.475	0.475	0.425
C_{12}	0.575	0.525	0.500	0.500
C_{13}	0.600	0.550	0.550	0.525
C_{14}	0.600	0.575	0.600	0.550
C_{15}	0.625	0.575	0.600	0.600
C_{16}	0.625	0.625	0.625	0.600
C_{17}	0.625	0.600	0.650	0.650
C_{18}	0.700	0.625	0.700	0.725
C_{19}	0.725	0.725	0.700	0.700
C_{20}	0.750	0.725	0.700	0.700
C_{21}	**0.775**	0.750	0.750	**0.775**
C_{22}	0.750	0.750	0.750	**0.775**
C_{23}	0.750	0.750	**0.775**	**0.775**
C_{24}	0.750	**0.775**	0.750	**0.775**
C_{25}	0.750	**0.775**	0.750	**0.775**
C_{26}	0.750	0.750	0.750	**0.775**
C_{27}	0.750	0.750	0.700	**0.775**
C_{28}	0.725	0.750	0.650	0.725
C_{29}	0.725	0.700	0.600	0.700
C_{30}	0.625	0.650	0.575	0.650

In this experiment, the same maximum accuracy, 77.5 percent was achieved with each protocol by using the C Measure. Protocol *IV* had the same maximum performance for order $21 - 27$. Of the other protocols, two (*I* and *III*) each had a single maximum at different order sizes. Protocol *II* had two maximum performances at orders 24 and 25.

Table 4.13 shows the performance of the PPM based algorithms on the Gutenberg Corpus. The highest overall performance is 95.0 percent. This is achieved using protocol *II* and order three. Exclusions had no effect on the performance.

Table 4.13: Categorisation accuracy for Gutenberg using PPM-based methods

	Protocol							
	With Exclusions				Without Exclusions			
Maximum Order Size	*I*	*II*	*III*	*IV*	*I*	*II*	*III*	*IV*
2	**0.750**	0.750	0.575	0.600	**0.725**	0.750	**0.550**	0.575
3	**0.750**	**0.950**	0.550	0.875	0.675	**0.950**	**0.550**	**0.875**
4	**0.750**	0.900	0.575	0.900	0.650	0.900	0.500	0.825
5	0.700	0.900	0.575	**0.925**	0.575	0.875	0.525	**0.875**
6	0.700	0.875	**0.625**	0.575	0.600	0.900	0.525	0.450
7	0.700	0.425	**0.625**	0.275	0.625	0.350	0.525	0.225

4.5 Conclusions

In this chapter a new algorithm called the C measure has been introduced. This algorithm has been shown to have the best performance at the authorship task for the Reuters News corpus. The C measure, however, did not beat the traditional approach of PPM for either the Gutenberg or News Groups corpora.

In addition to introducing a new algorithm, this chapter also described a new protocol for performing text categorisation. This new protocol was evaluated on the Gutenberg and News Group corpora.

The evaluation has shown that there is not a single *best* algorithm for performing text categorisation. It is clear, however, that the performance of the PPM algorithm is excellent across all of the corpora tested.

Chapter 5

Capturing User Interaction Streams

5.1 Introduction

As described in the thesis statement 1.2, the use of a graphical application such as a calculator can be modelled as a stream of symbols. In this chapter a novel method of capturing Human Computer Interaction as a stream of symbols will be discussed. In order to do this, an understanding of the anatomy of a modern desktop application is required. This will be described in section 5.2.

Following this, the design objectives for capturing a stream of symbols will be explored (section 5.3). Technical objectives will also be outlined in section 5.3.2. As part of the process of developing the methodology, it is necessary to define the new terminology that will be used in the remainder this thesis (section 5.4). Finally, section 5.5 explores the definition of a User Interaction Stream as a language and considers the implications of this for a simple application.

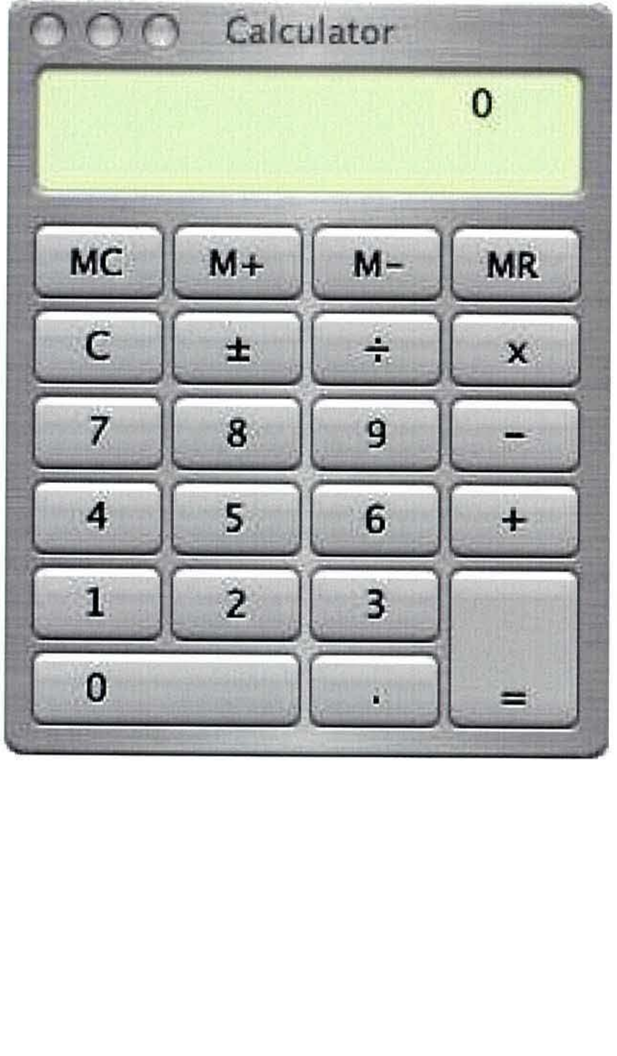
5.2 The Anatomy of a Desktop Application

This research has developed a method for capturing meaningful user interactions. Towards an outline of this development, it will be useful to discuss the way a user interacts with a graphical interface.

Modern operating systems provide programmers with a collection of graphical widgets, such as buttons, labels and other graphical components, for example, radio buttons, text fields, labels and frames. These allow a programmer to build applications that abstract the user from the technical operations. These widgets also allows a programmer to build applications that resemble interfaces found in the real world. This reduces the amount of training and knowledge required to operate the application as discussed in chapter 2.

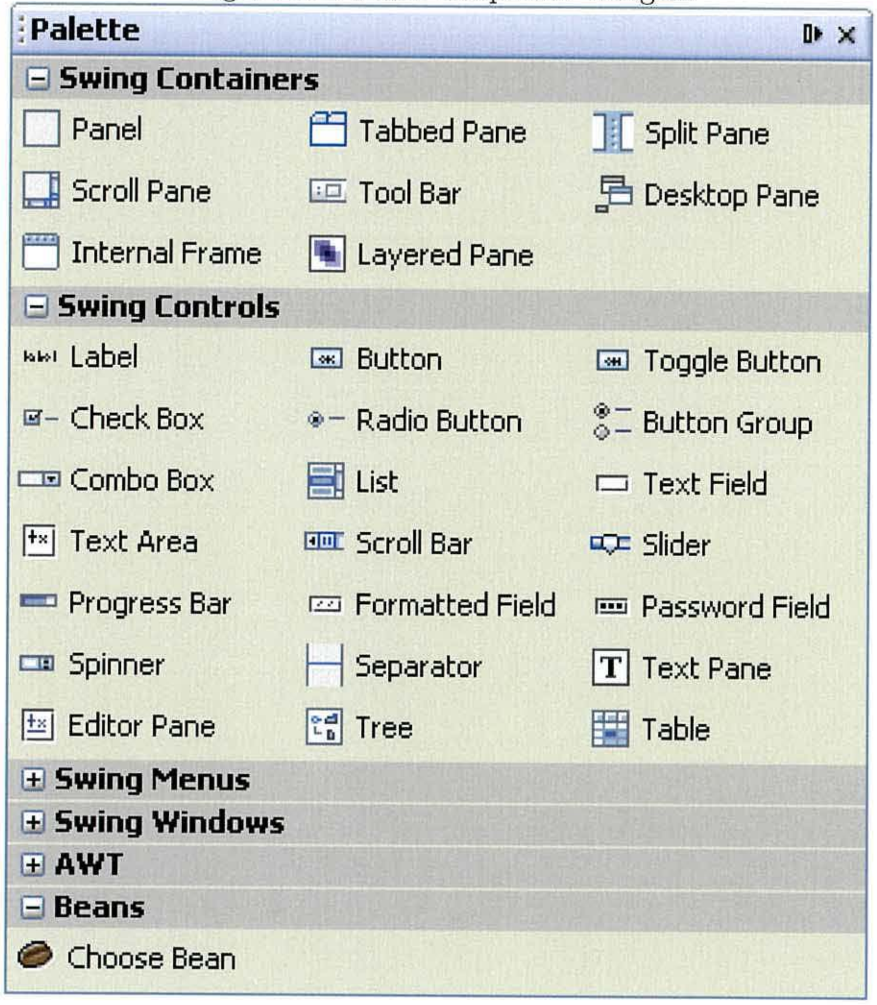
A typical example of a modern graphical application is the virtual calculator, such as the one shown in figure 5.1. This is provided by most operating systems. This interface closely matches the appearance of a *real* desktop calculator. Someone who has used a desktop calculator will be immediately familiar with the virtual calculator and therefore be able to use it confidently to perform tasks. The user interacts with the application by moving the mouse over buttons, clicking and releasing the mouse and pressings keys on the keyboard. These actions represent all the interactions between a user and an application. When a user interacts with an application such as the calculator, the interaction takes place with a widget. For example, when clicking the =, the interaction is a button click, and the widget is the = button. At first glance it might appear that a user can also *drag* objects to interact with them, but a drag consists of a mouse move, after a mouse button down but before a mouse button up, and so is already encapsulated by the actions already listed.

Figure 5.1: A Typical Graphical User Interface.



The pallet shown in figure 5.2 is an example of a small collection of the basic graphical widgets available to a Java developer.

Figure 5.2: Basic Graphical Widgets



5.3 Design Objectives for Developing an Interaction Capturing System

One of the objectives of this thesis is to develop a method of capturing the interactions between a user and a graphical desktop application as a stream of symbols. (1.2). As shown in section 3.11, there is a need to develop this system as it does not currently exist. Ideally, the overall design objective for an interaction capturing system should be that it is both meaningful and unobtrusive. In order to meet these aims, a number of general and technical objectives must be considered. These are outlined below.

5.3.1 General Objectives

The general objectives of producing a quality interaction capturing system are:- consistency, meaningful interaction and minimal user impact. These are each outlined in detail below.

Consistency

The product of an interaction capturing system will be a stream of symbols that represent the captured interactions. This stream will hereafter be referred to as the User Interaction Stream and will be fully defined in section 5.4.5. The User Interaction Streams should be consistent. They will be said to be consistent if the same User Interaction Stream is produced by any user performing the same series of actions regardless of:

Application Screen Size Resizing the application should have no effect.

Screen Location Repositioning the application should have no effect, in a similar way to the way a document written by an author is not effected by where the author was sitting or what the author used in order to write the document.

Physical Location Running the same application on a different computer should produce the same User Interaction Stream.

Restarting Restarting the application should always produce the same User Interaction Stream for the same series of interactions.

Version Different versions of the application with the same widgets should produce the same User Interaction Stream. This can be illustrated as follows:-

Given a basic calculator (figure 5.1) a newer version of the same calculator that has additional features, e.g. the inclusion of scientific functions (sin cos tan etc.) will still produce the same stream for operations when a user only interacts with the basic widgets (i.e. those that were already present in the original version).

Meaningful Interaction

The interaction capture system must capture *meaningful* interactions between the user and the graphical desktop application. In this context, the term meaningful refers to any interaction that changes the state of the application that is being observed. Although it would be possible to capture *all* interactions between a user and an application, this task would be overwhelming. Capturing all interactions would include the position of every mouse movement and every key press. It would also include a video stream of where the user was looking. Even excluding the capture of eye movement, this would produce a large volume of data, most of it unhelpful. A useful analogy would be like trying to capture every stroke of a pen made when writing a single letter of the alphabet.

To address this difficulty, only meaningful interactions will be captured. A useful definition of a meaningful interaction is "one that changes the state of the application." Using this definition, only actions that meet this criteria will be recorded. For example, moving the mouse pointer around within a GUI component, such as a button, does not change the state of the application or indicates intent. However when the mouse leaves that component, the state is changed. It is this action that will be regarded as meaningful and will therefore be recorded.

User Impact

Another design objective is to avoid affecting the user's experience when using the application. The two main considerations for this are:-

Speed The application should not appear to run more slowly when the interaction capture system is running.

Visual Appearance The interaction capture system should not alter the visual appearance of the application in any way.

5.3.2 Technical Objectives

In addition to the above general objectives, there are technical design objectives to consider:

Reusable The same interaction capture system should be applicable to any of the graphical applications.

Adaptable Changing the application should not be made more difficult by the inclusion of the interaction capture system.

Maintenance The application should still be maintainable by the original authors despite the inclusion of an interaction capture system.

Stable The introduction on the interaction capture system should not introduce additional instability into the application.

At present, there are no effective methods for capturing the interactions between a user and a graphical desktop application as a stream of symbols that meet all of the objectives laid out above.

5.4 Definitions of New Terminology

5.4.1 Definition of Target Symbols

A simple definition of a Target Symbol is "a unique symbol assigned to an individual widget in order to identify which widget is the target of an interaction."

By looking at the way Target Symbols are assigned to the calculator interface (shown in figure 5.1 in section 5.2) a more complete understanding of the meaning and generation of a Target Symbol can be reached. As has been discussed previously, the interface of the calculator is made up of a frame containing a label widget and a number of button widgets. A unique symbol, called the Target Symbol, will be assigned to each of these widgets. The symbol is a property of the widget. As the symbol represents the widget and not its relative or absolute position, moving, resizing or even rearranging the layout of widgets will not alter the Target Symbol.

The calculator shown in figure 5.3 shows the Target Symbols attached to a number of widgets. The naming convention for the Target Symbols is explained later, in section 5.4.1.

In this instance, Target Symbol $B3$ refers to the button widget M-. The Target Symbol $L1$ refers to the label and the Target Symbol $F1$ refers to the entire frame.

Figure 5.3: Example of Target Symbols Assigned to the Interface in Figure 5.1

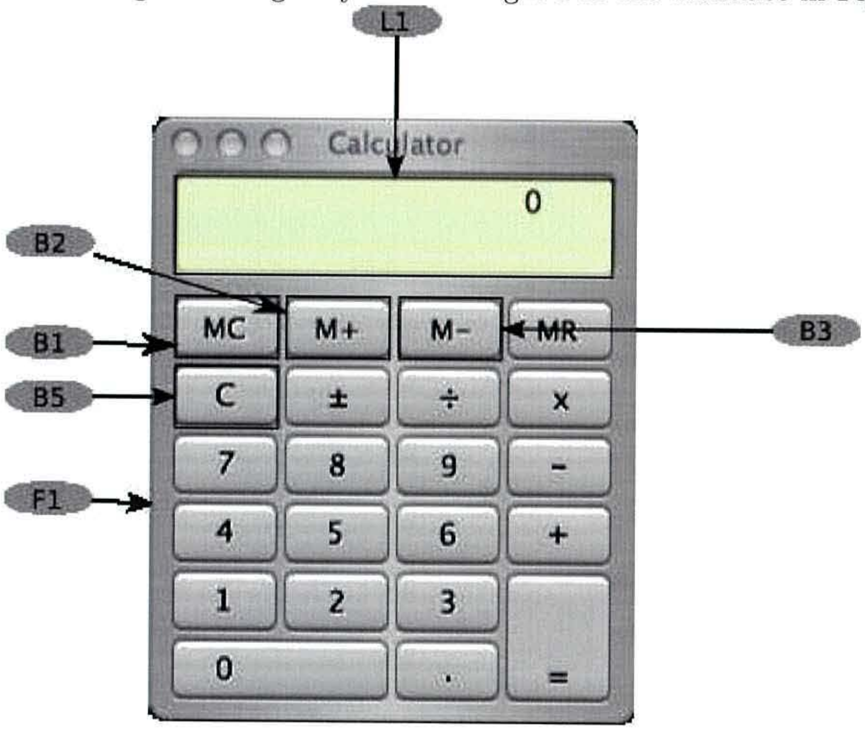


Table 5.1 shows the complete Target Symbol list for all the button widgets for the calculator application shown in figure 5.1.

Table 5.1: Widget Symbols

Button Text	Symbol	Button Text	Symbol
MC	B1	M+	B2
M-	B3	MR	B4
C	B5	±	B6
÷	B7	x	B8
7	B9	8	B10
9	B11	-	B12
4	B13	5	B14
6	B15	+	B16
1	B17	2	B18
3	B19	=	B20
0	B21	.	B22

Although the fundamental property of each symbol is its uniqueness, by encoding an additional piece of information in each symbol — e.g. prefixing all buttons with the symbol *B* — the Target Symbol will thereby identify the class of widget. Again, referring back to the calculator example, all buttons have been given the Target Symbol in the form B*n*, where *n* is a counter from 1 to the total number of buttons in the application and B denotes the widget is a button. There is no additional information encoded in the number. Similarly, the frame has the symbol F1, where the F prefix denotes that it is a frame widget and one indicates that this is the first frame.

5.4.2 Conventions

As a convention, symbols are contained within a box. For example $\boxed{\text{KP}}$ represents a single Action Event symbol. When two symbols are concatenated together to form and Action Event, the box will contain both the Target Symbol and the Action Symbol delimited by a ",". For example $\boxed{\text{KP,F1}}$ contains combined symbols: The Action Symbol $\boxed{\text{KP}}$ and the Target Symbol $\boxed{\text{F1}}$.

5.4.3 Definition of an Action Symbol

An Action Symbol is a symbol, with or without a value, used to denote a unique action performed by a user on an application in order to capture the full *meaning* of the interaction.

In this definition, a symbol is used to identify the class of the action. For example, $\boxed{\text{KP}}$ is the symbol to denote that a user has pressed a key down and MR denotes that the user has released the mouse button. Given that there are any number of keys that a user could press, the symbol alone does not capture the full meaning of the user's interaction. By assigning a value to the symbol e.g. d when the key d on the keyboard is pressed, or h where the user has pressed the h key, the full meaning of the interaction is captured.

As can be seen from table 5.2, Action Symbols provide a simple method to record an action. So $\boxed{\text{KPf}}$ $\boxed{\text{KUf}}$ describes the action of the user pressing and releasing the f key on a keyboard.

Table 5.2: Action Symbols

Symbol	Value	Action Description
MM		Mouse Moved
MC	button number	Mouse Clicked.
MR	button number	Mouse Released.
KD	Key Value	Key Pressed.
KU	Key Value	Key Released.

5.4.4 Definition of an Action Event

An Action Event is the combination of a Target Symbol and an Action Symbol. From this simple definition, it can be seen that an Action Event encodes the action the user performed and the widget that the user interacted with. The Target Symbol provides the context for the action which is encoded in the Action Symbol. Together, this combination captures the *meaning* of the interaction between the user and the application. An example of an Action Event is: $\boxed{\text{MC1,B18}}$. If this is broken down it can be seen that this Action Event is made up of the Action Symbol $\boxed{\text{MC1}}$ (Mouse button one click) and the Target Symbol ($B18$) (

). i.e. the user clicked Mouse Button 1 on the button

.

5.4.5 Definition of a User Interaction Stream

A User Interaction Stream is a stream of Action Events that contains enough information for the all interactions made by the user to be recreated in the same order that they were produced.

Below are explanations of two contrasting examples of User Interactions Streams. Each of these examples show a user performing the same operation on the calculator application (figure 5.1). In each example the user performs the operation $5 + 2 =$ but using a different interaction technique. By examining these two operations, it can be seen how different and distinct User Interaction Streams are produced.

5.4.6 Example One: Using only the Mouse to Perform the Operation $5 + 2 =$

In this example, the User Interaction Stream generated by performing the operation $5 + 2 =$ using the mouse is examined. It can be seen that this User Interaction Stream is made up of twenty nine Action Events. Each Action Event is composed of a Target Symbol and an Action Symbol. The first Action Event to make up the User Interaction Stream is denoted as $\boxed{\text{MM,F1}}$, where $\boxed{\text{MM}}$, the Action Symbol, refers to a mouse movement and $\boxed{\text{F1}}$ refers to the Target Symbol Frame one. The meaning of $\boxed{\text{MM,F1}}$ can be seen to be "mouse enters the frame" (line one in table 5.3). Referring to table 5.3, it is possible to follow each Action Event sequentially.

Table 5.3: A User Performing the Operation $2 + 5 =$ Using the Mouse

Action Event	**Meaning**	**User Interaction Number**
$\boxed{\text{MM,F1}}$	Mouse enters the frame.	1
$\boxed{\text{MM,B21}}$	Mouse leaves frame and moves over button $\boxed{0}$.	2
$\boxed{\text{MM,F1}}$	Mouse leaves button $\boxed{0}$ and moves over frame.	3
$\boxed{\text{MM,B18}}$	Mouse leaves frame and moves over button $\boxed{2}$.	4
$\boxed{\text{MC1,B18}}$	Mouse button 1 is clicked on button $\boxed{2}$.	5
$\boxed{\text{MR1,B18}}$	Mouse button 1 is released on button $\boxed{2}$.	6
$\boxed{\text{MM,F1}}$	Mouse leaves button $\boxed{2}$ and moves over frame.	7

Continued on next page

Table 5.3 – continued from previous page

Action Event	**Meaning**	**User Interaction Number**
MM,B19	Mouse leaves frame and moves over button 3.	8
MM,F1	Mouse leaves button 3 and moves over frame.	9
MM,B15	Mouse leaves frame and moves over button 6.	10
MM,F1	Mouse leaves button 6 and moves over frame.	11
MM,B16	Mouse leaves frame and moves over button +.	12
MC1,B16	Mouse button 1 is clicked on button +.	13
MR1,B16	Mouse button 1 is released on button +.	14
MM,F1	Mouse leaves button + and moves over frame.	15
MM,B15	Mouse leaves frame and moves over button 6.	16
MM,F1	Mouse leaves button 6 and moves over frame.	17
MM,B14	Mouse leaves frame and moves over button 5.	18
MC1,B14	Mouse button 1 is clicked on button 5.	19
MR1,B14	Mouse button 1 is released on button 5.	20
MM,F1	Mouse leaves button 5 and moves over frame.	22

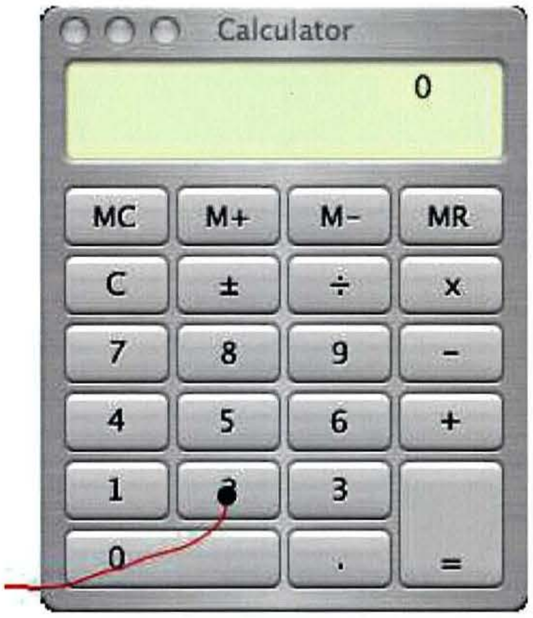
Continued on next page

Table 5.3 – continued from previous page

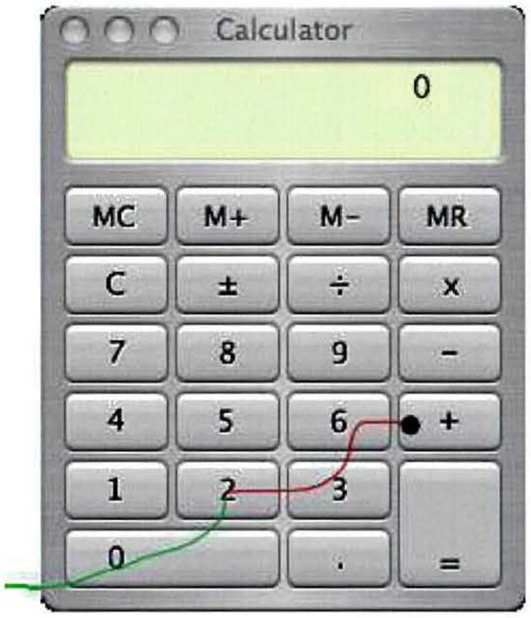
Action Event	Meaning	User Interaction Number
MM,B15	Mouse leaves frame and moves over button $\boxed{2}$.	23
MM,F1	Mouse leaves button $\boxed{2}$ and moves over frame.	24
MM,B19	Mouse leaves frame and moves over button $\boxed{.}$.	25
MM,F1	Mouse leaves button $\boxed{.}$ and moves over frame.	26
MM,B20	Mouse leaves frame and moves over button $\boxed{=}$.	27
MC1,B14	Mouse button 1 is clicked on button $\boxed{=}$.	28
MR1,B14	Mouse button 1 is released on button $\boxed{=}$.	29

Another way to demonstrate the path taken by the mouse to generate the User Interaction Stream whilst performing the operation $5 + 2 =$ using only the mouse can be seen in figure 5.4.

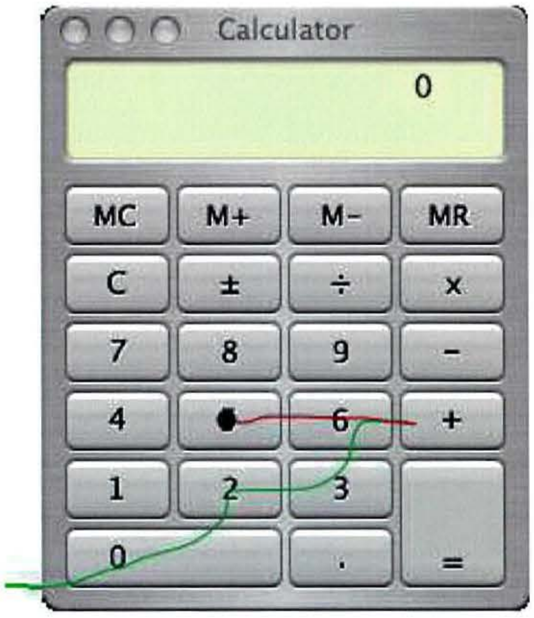
Figure 5.4: The Mouse Path Taken When Performing the Operation $5 + 2 =$ Using Only the Mouse.



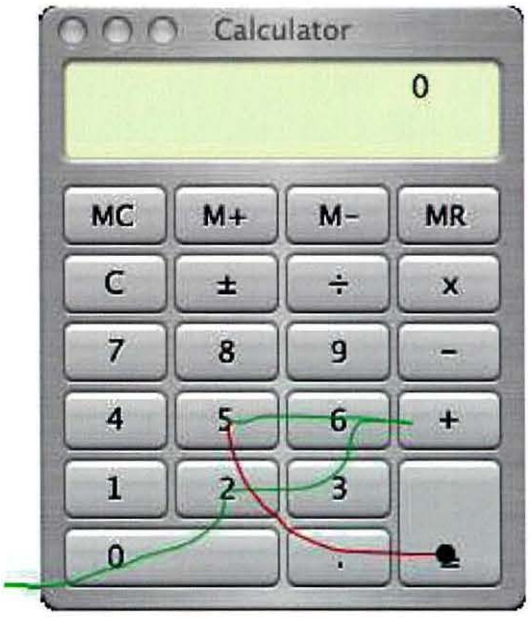
(a) Moving from 0 to 2.



(b) Moving from 2 to +.



(c) Moving from + to 5.



(d) Moving from 5 to =.

Here, the user starts with the mouse off screen. In figure 5.5(a) the user moves to $\boxed{2}$. This sequence generates User Interactions numbered one to four — see table 5.3. The user then presses and releases mouse button 1. This corresponds to User Interactions five and six. Figure 5.5(b) shows the user moving the mouse from $\boxed{2}$ over to $\boxed{+}$. This corresponds to User Interactions numbered from seven to twelve. The user then clicks $\boxed{+}$, User Interactions thirteen and fourteen. Figure 5.5(c) shows the user moving the mouse from $\boxed{+}$ to $\boxed{5}$ and then clicking the button. This corresponds to User Interactions numbered fifteen to twenty. Finally, figure 5.5(d) shows the passage of the mouse from $\boxed{5}$ to the $\boxed{=}$ and the user then clicking $\boxed{=}$. These are the remaining User Interactions.

5.4.7 Example Two: Using only the Keyboard to Perform the Operation $5 + 2 =$

In this example, the User Interaction Stream generated by performing the operation $5 + 2 =$ using only the keyboard is examined. It is shown to be made up of four Action Events. Each Action Event is composed of a Target Symbol and an Action Symbol, so that the first Action Event to make up the User Interaction Stream is denoted as $\boxed{\text{KP2,F1}}$, where $\boxed{\text{KP2}}$, the Action Symbol, refers to a keyboard press of the key 2 and $\boxed{\text{F1}}$ refers to the Target Symbol Frame one. The meaning of $\boxed{\text{KP2,F1}}$ can be seen to be as follows. The user presses the button marked 2 on the keyboard (line one in table 5.4). Using table 5.4, it is possible to follow each Action Event sequentially. In this example, the User Interaction Stream contains four Action Events.

Table 5.4: A User Performing the Operation $2 + 5 =$ Using only the Keyboard.

Action Event	Meaning	User Interaction Number
KP2F1	Press keyboard button 2	1
KP2F1	Press keyboard button +	2
KP2F1	Press keyboard button 5	3
KP2F1	Press keyboard button enter	4

5.5 Language Properties of User Interaction Streams

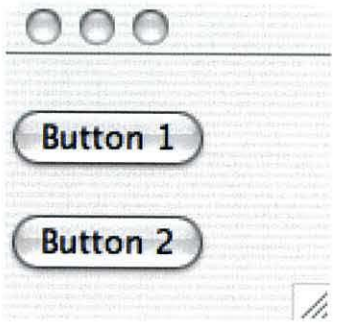
In this section, the formal definition of a User Interaction Stream as a language will be described. As has already been defined, a User Interaction Stream consists of a series of Action Events. Each Action Event is composed of a Target Symbol and an Action Symbol. This can be expressed formally as follows:- A User Interaction Stream is a stream of symbols S in the language L. The stream of symbols S will be composed of letters from the alphabet below:-
$\sigma = \{TargetSymbols\}+$
$\alpha = \{ActionSymbols\}$
The grammar for the language L is defined as:-
$L = \{\alpha\sigma\}*$

This grammar, although complete, does not entirely restrict the possible streams. Certain Action Events can never follow others. For example, it is not possible to have the Action Event *"Mouse Button 1 released"* on button one without first having a ' *"mouse button 1 click"* Action Event on button one. Therefore, there is a more restrictive grammar that takes this into account. This is discussed in more detail below.

5.5.1 The Grammar Of a Limited Application

This section fully defines the grammar of a limited application. The application that will be used to define the grammar consists of a frame and two buttons. The application can be seen in figure 5.5.

Figure 5.5: Simple Application Consisting of Two Buttons and a Frame



Interactions

In order to limit the grammar for illustrative purposes within this section, the interactions have been limited to the following:-

Mouse Movement The user can move the mouse from the frame to button one or button two and from button one or button two to frame one.

Mouse Pressed The user can press a mouse button. (For simplicity, this example will only have one mouse button available.)

Mouse Released A mouse button that was previously pressed is released.

Key Pressed The user presses a key on the keyboard. (Again, for simplicity, this example will only have a single key.)

Key Released A key that was previously pressed is released.

States

By considering the effect of each of the interactions above with the application shown in figure 5.5, a list of all the possible states can be generated. This list of states is shown in table 5.5.

Finite State Automata

Using the application shown in figure 5.5, the limited set of interactions can be used to move between states of a finite state automata. The states are described

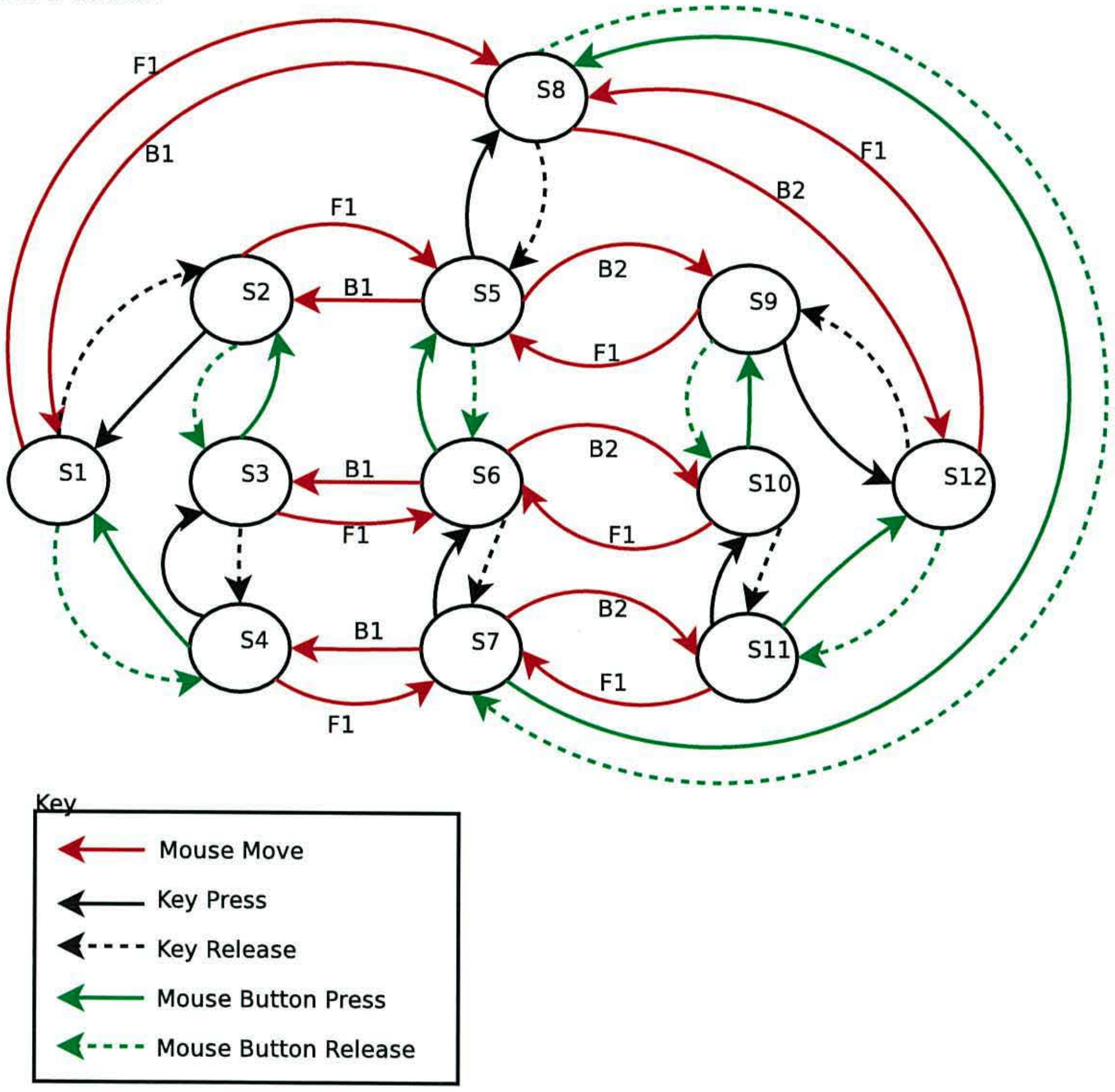
Table 5.5: Possible States of the Simple Application.

State	Description
S_1	Mouse over button one. Key and button pressed.
S_2	Mouse over button one. Mouse pressed, no keys pressed.
S_3	Mouse over button one. No mouse buttons or Keys pressed.
S_4	Mouse over button one. Key pressed, no mouse buttons pressed
S_5	Mouse over frame one. Mouse pressed, no keys pressed.
S_6	Mouse over frame one. No mouse buttons or Keys pressed.
S_7	Mouse over frame one. Key pressed, no mouse buttons pressed
S_8	Mouse over button two. Key and button pressed.
S_9	Mouse over button two. Mouse pressed, no keys pressed.
S_{10}	Mouse over button two. No mouse buttons or Keys pressed.
S_{11}	Mouse over button two. Key pressed, no mouse buttons pressed
S_{12}	Mouse over button two. Key and button pressed.

in the section 5.5.1. Figure 5.6 shows the complete finite state automata. Moving between states occurs only on user input. The label associated with a mouse move shows the end destination of a mouse move This finite state automata can be used to define the language of the simple application.

This finite state automata has been constructed for a very simple application and has been limited to a small number of interactions.

Figure 5.6: Finite State Automata of a Simple Application with Only Two Buttons and a Frame.



Example of User Interaction Streams Language

The interactions with the application shown in figure 5.5 can be captured as a User Interaction Stream. This stream will be restricted by a grammar. The definition of this grammar is given below. As the interactions have been restricted, the Action Symbols are limited to the following (rather than the full set described in section 5.4.3):-

MM Mouse moved.

MP Mouse button pressed.

MR Mouse button released.

The Target Symbols are:

F1 Frame one.

B1 Button one.

B2 Button two.

The state transition table below describes the valid transitions from each of the states. There is no specific starting or end states. The start states would be where the application launches. That is, essentially all states can be considered as potential start states. As there is no way to terminate the program there is no end state. The state table can be converted to a regular grammar using one of the standard algorithms however valid start and end states would have to be defined.

$S_1 \rightarrow S_4|S_2|S_8$

$S_2 \rightarrow S_5|S_3|S_1$

$S_3 \rightarrow S_5|S_2|S_4$

$S_4 \rightarrow S_3|S_6|S_1$

$S_5 \rightarrow S_8|S_6|S_2|S_9$

$S_6 \rightarrow S_5|S_3|S_7|S_{10}$

$S_7 \rightarrow S_6|S_4|S_8|S_{11}$

$S_8 \rightarrow S_1|S_5|S_7|S_{12}$

$S_9 \rightarrow S_5|S_{10}|S_{12}$
$S_{10} \rightarrow S_6|S_9|S_{11}$
$S_{11} \rightarrow S_7|S_{10}|S_{12}$
$S_{12} \rightarrow S_8|S_9|S_{11}$

5.6 Summary

A novel method of capturing Human Computer Interaction as a stream of symbols has been described. A typical desktop application and the components that are assembled to form the interface was used to demonstrate the principles. A detailed exploration of the design objectives for capturing a stream of symbols was outlined. As part of this, the importance of capturing a meaningful interaction was discussed and a definition of meaningful was given. A definition of the technical objectives was also given. Towards a common understanding of terminology, new terminology used throughout this thesis was introduced and defined. Finally, the definition of a User Interaction Stream as a language was explored. The exploration included an example of a full language for a simple application. From this an implementation can be developed and explained.

Chapter 6

Implementation of an Interaction Capture System

This chapter is concerned with the implementation of an interaction capture system that fulfills the design objectives, as laid out in the previous chapter. The design objectives describe the properties of the interaction capture system. They are not concerned with the practical implementation of the system. In order to implement a system to capture interactions, a language and a GUI platform must be chosen. Once this choice has been made, methodologies to capture the interactions using this language and framework can be evaluated. A methodology can then be chosen and implemented.

6.1 Choosing a Language and Framework

There are a vast array of different languages and graphical frameworks in existence. As this thesis is concerned with the capture of interactions between a user and a graphical desktop application, clearly the language chosen should facilitate the production of such an application. One of the key technical objectives (see section 5.3.2) is that the capture system will have no adverse effects on the stability of the application. This requirement encourages limiting the choice of language to strongly typed, safe languages. As the interaction capture system will be generic, picking a commonly used language is desirable. This means there will then be a variety of applications available for use in later evaluations. It will also be advantageous to have access to the source code for the application, as this will simplify the production of an interaction capture system. In addition to having a wide variety of applications, it will be beneficial for the applications to use a common GUI framework. By capturing interactions with a common GUI framework, a large number of different applications can then be evaluated.

Java fulfills all these requirements. It is a safe, strongly typed language. There is an active development community that release many applications with source code. Java desktop applications mostly use the Java Swing framework to provide a desktop interface [1]. By building an interaction capture system that captures interactions with the Swing toolkit, it will then be possible to capture interactions with many different kinds of applications. Therefore, the combination of Java and the Swing framework have been selected as the tools to be used throughout this research.

6.2 Evaluating Methodologies for Capturing Interactions

As discussed in section 5.4.5, the Action Events that make up a User Interaction Stream are composed of an Action Symbol (5.4.3) and a Target Symbol (5.4.1). In order to produce the User Interaction Stream, a method for capturing each of these symbols must be identified. This is discussed in the following sections.

6.2.1 Evaluating Methodologies for Identifying the Target Symbol

The Target Symbol described in section 5.4.1, is used to identify the graphical widget that a user performs an interaction with. This section will evaluate three different potential techniques for identifying the Target Symbol with Java Swing

[1]Some applications are written using SWT Eclipse Foundation (2008*b*).

applications. Each technique will be outlined and the strengths and weaknesses of the technique will be considered.

Technique One — Modifying the Java Virtual Machine

This technique involves modifying the Java Virtual Machine (JVM) in order to identify the Target Symbol of the graphical objects. The Sun Java Virtual Machine (JVM) keeps the address of every object allocated in the heap. This is used by projects such as the Java Heap Analysis Tool SUN (2006) to uniquely identify objects within a heap dump. The unique identifiers are referred to as objectID's. As identified in the design objectives (chapter 5), consistency is a key goal. In order to use the objectID to identify the Target Symbol, and for the resultant stream to be consistent, the JVM would need to be modified. The modification would need to ensure that different invocations of the JVM were assigned to the same objectID each time the same graphical widget object was instantiated. A significant benefit of this technique would be that no changes would be needed to the source code of the application.

There are, however, a number of significant drawbacks to choosing this technique. The work involved in modification would be tantamount to maintaining a separate fork of a JVM. This would be difficult to complete and keep up to date, as new versions of the JVM are released regularly. There is also the potential for the introduction of new, difficult-to-fix bugs within the JVM.

Technique Two — Extending the Swing Framework

A toolkit that extends each Java Swing Class to contain a Target Symbol could be developed. This would require re-writing every Java application in order to use the new toolkit. The advantages of this technique are that the User Interaction Streams would satisfy the requirement for consistency, as outlined in the design requirements (chapter 5). Also, the risk of introducing additional instabilities into the application would be minimal.

However, there are a number of disadvantages to this technique. Access to the source code would be required. Also, the source code would be altered in many places, although the task of altering the code could be automated, using a similar technique to the one described later in section 6.4.1. Each Primitive Swing component would need to be extended.

Technique Three — Using JavaHelp

Java provides a system to enable developers to associate help with a GUI widget. This is called JavaHelp (Lewis 2000). It is used to associate context specific help with individual graphical widgets. The JavaHelp API is designed to provide context sensitive help to end users. This is accessed by a user pressing the $F1$ key on their keyboard, then clicking on a graphical widget. The help system then loads a help file with the specific help topic about the widget that they have clicked on. The JavaHelp API is designed so that the help can be written by a documentation writer with little or no Java application development skills. The application developer simply has to assign a special tag to each widget that has a help description.

The advantages of this technique are many. JavaHelp system tags are persistent across both multiple invocations of the same program on different machines and persistent across different versions of the same application. This satisfies the design objective to produce consistent streams. Using JavaHelp is very safe. For example, if an object does not have an ID assigned, then only the Target Symbol information would be lost. The JavaHelp API is already extensively used and well understood.

There are, however, some disadvantages. If an application already takes advantage of the JavaHelp, it will have tags already assigned to some, though not necessarily all, of the graphical widgets. It is, however, possible to work around this problem relatively easily. As with extending the Swing framework, access to the source code would be required and the code could then be altered. The code alterations could then be automated. The alterations would be relatively minor, as they would consist only of adding a few lines of code. Another potential problem is that there is no requirement for each object to have a unique JavaHelp ID or helpID.

6.2.2 Evaluating Methodologies for Identifying the Action Symbol

The Action Symbol, described in section 5.4.3, is used to identify the type of interaction that a user has performed. This section will evaluate a technique for identifying the Action Symbol with a Java Swing application. The Java Swing Toolkit processes interactions by passing them through the event queue *Java Event Queue API* (2004). To capture events and turn them into an Action Symbol, a component that converts the interaction into an Action Symbol can be pushed onto the top of this queue. All events will then be converted into Action Symbols by this component, before then being handled by the application. The only drawback of this approach is that interactions with modal windows would not be captured. A modal window is one that does not permit the user to enter data into any other window until the window has been dismissed. A modal dialogue is typically used to display warnings.

6.3 Choosing a Methodology

The choice of methodology for capturing the Action Symbol is unambiguous, as the method described above (6.2.2) satisfies all the requirements.

The choice of methodology for capturing the Target Symbol, however, is less clear cut. Altering the JVM can be immediately dismissed, due to the risks involved. This narrows the choice to either using JavaHelp or extending the Swing Framework. As both of these techniques alter the source code of the application, it will be necessary to examine the alterations made to the source code using both methods and then asses the impact of these changes to the original source code.

6.3.1 Assessing Source Code Changes

Listing 6.1 shows the code used to create a Java button.

Listing 6.1: Button Creation Code

```
import javax.swing.JButton;
```

```java
public class ButtonCode
{
  JButton  myButton = new JButton();
}
```

The code shown in listing 6.2 shows how the code from the above listing would have to be changed to use the extension of the Swing Framework.

Listing 6.2: Button Creation Using Extended Toolkit

```java
import javax.swing.JButton;
public class ButtonCode
{
  JButton  myButton = new ExtendedJButton();
}
```

Listing 6.3 shows the modification needed to listing 6.1 to use the JavaHelp system.

Listing 6.3: Button Creation Using JavaHelp

```java
import javax.help.CSH;
import javax.swing.JButton;
public class ButtonExample
{
  publc ButtonExample()
  {
    Jbutton  myButton = new JButton();
    CSH.setHelpID(myButton,"jb01");
  }
}
```

6.3.2 Conclusion

From looking at the code differences between listing 6.1 and the altered versions, it is clear that the alterations needed to use the JavaHelp system (listing 6.3) are less invasive than those needed to use the Swing Framework extension (listing 6.2). The code modification consists only of the addition of a line to the original code rather than the modification of an existing line. The new line that is added is simple to track with source code management systems and so makes spotting the additions simpler. The additional simplicity of the modifications to the code needed to use JavaHelp more than outweighs the work required to avoid other problems. Therefore using JavaHelp was chosen for this research.

6.4 Implementing the Methodology

This section demonstrates the chosen methodology for capturing and recording User Interaction Streams. The method for associating the Target Symbol with the graphical widget will be outlined in section 6.4.1. Section 6.4.2 will describe an automated tool that can be used to perform this association. A method to enforce uniqueness to each association will be discussed in section 6.4.3, followed by a description of capturing Interaction Symbols (section 6.4.4). The final, section 6.4.5, details the logging process.

6.4.1 Associating the Target Symbols

As has already been discussed, associating the Target Symbols with graphical widgets was facilitated by the JavaHelp system. To use JavaHelp for this task, each time a graphical widget is instantiated it must be registered with the JavaHelp system and a unique Target Symbol assigned. The standard JavaHelp has no enforcement that helpID's are unique. The designated Target Symbol is set as the helpID of the graphical widget. As unique Target Symbols are necessary to produce User Interaction Streams, a mechanism for ensuring the helpID's are unique was created. This is described in detail in section 6.4.3.

The assigning of a helpID (i.e. Target Symbol) to a graphical widget takes place at the source code level. The source code for a Java application is scanned and a Target Symbol assigned to each GUI component as it is created. The scanning and assignments can be done manually or in an automated fashion. The scanner, either human or computer, looks for the creation of new Java objects that extend an

AWT component. The example, in Listing 6.4, shows the creation of a new graphical widget, a *JButton*.

Listing 6.4: Button Creation Code

```
import javax.swing.JButton;
public class ButtonCode
{
  JButton  myButton = new JButton();
}
```

When a line like the listing is found, a new line is added that assigns a Target to the graphical component. This is shown in the next code example:

Listing 6.5: Button Creation Using JavaHelp

```
import javax.help.CSH;
import javax.swing.JButton;
public class ButtonExample
{
  publc ButtonExample()
  {
    Jbutton  myButton = new JButton();
    CSH.setHelpID(myButton,"jb01");
  }
}
```

The Code above registers the myButton graphical widget with the JavaHelp system and sets the helpID *jb01*.

For a large graphical application, the process of associating a Target Symbol with each graphical widget would be very time consuming. There is also a huge potential for error, for example, missing a widget creation, as the creation code may well be buried within the application logic. It is non trivial to check that every component has been associated with a Target Symbol. The reason this is difficult is that if a component does not have a helpID, the JavaHelp system will recurse up the component stack to return the helpID of the first component with a helpID assigned. The following is an example of recursion:-

A JButton is added to a panel, which is then added to a JFrame. The JFrame has a helpID assigned to it but no helpID is assigned to any of the other components. When the helpID of the button is requested, JavaHelp will fail to find a helpID for the button. JavaHelp will then request the helpID of the panel (as it is the button's parent). As the panel also has no helpID the call is passed on to the panel's parent, the frame. The frame has a helpID assigned and so this is returned.

The source code example below shows the effect of this recursion. It demonstrates that the assert on line twenty four will always pass. The passing of the assert on line twenty four indicates that both the JButton, created on line 15, called button and the JFrame created on line 13, called frame, have both been returning the same helpID. This is despite *only* the JFrame fame being assigned a helpID. The JButton button was never assigned a helpID.

Listing 6.6: JavaHelp Recursion Example

```
   import javax.swing.JFrame;
 2 import javax.swing.JButton;
   import javax.swing.JPanel;
 4 import javax.help.CSH;

 6
   public class ButtonFrameExample
 8 {

10   public ButtonFrameExample()

12   {
       JFrame frame = new JFrame();
14     JPanel panel = new JPanel();
       JButton button = new JButton();
16     panel.add(button);
       frame.add(panel);
18     CSH.setHelpIDString(frame,"someHelpId");
       String buttonID = CSH.getHelpIDString(button);
```

```
        String panelID = CSH.getHelpIDString(panel);
        String frameID = CSH.getHelpIDString(frame);


        assert (buttonID.equals(frameID) && panelID.equals(frameID));

    }
}
```

Recognising the potential problem of recursion, the extended helpID class described in section 6.4.3 has a check to ensure that the helpID returned is only set on the component that the request was made, i.e. no recursion has taken place.

6.4.2 Automating the Tagging Process

As discussed above (section 6.4.2) the process of manually associating a Target Symbol with each graphical widget would be very time consuming and error prone. For the purpose of this research, this process was automated. This section will outline the method of automation that was used.

Automation Methodology

By targeting an application that has already been written in Java there are a number of properties of the source code that can be exploited. Because the application is known to compile it must, therefore, comply to the Java grammar. The Java grammar is provided as part of the Java Compiler Compiler kit — javacc *JavaCC* (2008).

Using the Java grammar and javacc, an application was built to look for declarations that matched a specific pattern. The parser that is generated by the combination of javacc and the java grammar assigns a *type* to each *token* that has been scanned. Initially the following code sequence was searched for:-

$IDENTIFIER_1, ASSIGN, NEW, IDENTIFIER_2$

\$VARIABLE ($IDENTIFIER_1$) is the variable name of the object that is being instantiated which will be referred to as *\$VARIABLE.*

\$CLASS The final identifier ($IDENTIFIER_2$) is the class that is being instantiated which will now be referred to as *\$CLASS.*

In the code shown in listing 6.1 *\$VARIABLE* is *myButton* and *\$CLASS* is *JButton.* The automated tool is only concerned with altering the code when the *\$CLASS* is part of the Swing toolkit or an extension of a Swing component. The *\$CLASS* is checked to ensure it is an *interesting* class (i.e. a swing component).

To assess if the *\$CLASS* is interesting, it could be instantiated via reflection. However, to instantiate the *\$CLASS* there is a potential for there to be a dependency on other objects, which would in turn also require instantiation. For example, if a developer had extended *JButton* to wrap around his or her own code, then the extended *JButton* constructor would require the instantiation of their wrapped object first. It was therefore decided that rather than try and use reflection, the program would take an optional list of classes to consider as interesting (a member of the Swing classes). It is important to note that methods that return a Swing object will, at some point, instantiate the object to be returned, or the object will already have been instantiated and therefore have had its helpID set.

Encoding the Graphical Widgets Class

As part of the automation process, the class of the graphical widget is encoded as part of the Target Symbol. For example, all buttons are members of the button class. When using the automation outlined above, each graphical widget was assigned a Target Symbol that contained two pieces of information:-

1. The class of the Object e.g. all JButton instances were assigned a helpID with the same prefix e.g. the letter a.

2. A numeric count based on the total number of assigns of that class made in the application so far.

6.4.3 How to Enforce the Unique Assignment of Target Symbols

The code below (6.7) was written to enforce that the Target Symbol for each graphical widget is unique. This meets the design objective 5.4.1. In addition to enforcing a unique assignment, the code also ensures that the returned Target Symbol is assigned to the actual graphical widget and not to the graphical widget's parent(s).

Listing 6.7: Unique Help Enforcement.

```
import java.awt.Component;
import java.util.HashMap;
import java.util.Map;
import javax.help.CSH;
public class UniqueHelpID extends CSH {
    private static Map<Component,String> assigned =
        new HashMap<Component,String>();
    public static void setHelpIdString(Component c, String id)
        throws KeyViolationException {
        if(assigned.values().contains(id))   {
            String currentID = assigned.get(c);
            if(currentID != null) {
                if(!currentID.equals(id)) {
                    throw new KeyViolationException(c,id);
                }}}
        if(assigned.get(c) != null) {
                String currentID  = CSH.getHelpIDString(c);
                if(id.equals(currentID)) {
                        return;
                }
                throw new KeyViolationException(currentID, id);}
        CSH.setHelpIDString(c,id);
        assigned.put(c,id); }
    public static String getHelpIDString(Component c) {
        String result = CSH.getHelpIDString(c);
        String parentID  = CSH.getHelpIDString(c.getParent());
        if(result.equals(parentID)) {
                return null;
            }
        return result;}}
```

6.4.4 Capturing the Action Symbol

This section describes in detail the necessary steps undertaken by the component used to capture and convert User Interactions into Action Symbols.

When processing an interaction, there are a number of steps that take place. First the Target Symbol is identified. This is used to determine whether the interaction is meaningful or not. (In this context, the term meaningful refers to an interaction that changes the state of the application.) After identifying the Target Symbol, the interaction type is determined. If the interaction type is a mouse movement and the Target Symbol is identical to the previous Target Symbol, then no further action is taken because the interaction is not meaningful. In all other cases, the interaction type and, where appropriate the value, are used to construct an Action Symbol. This Action Symbol and associated Target Symbol are then passed to the logging system. The logging system is outlined below (6.4.5.)

6.4.5 Logging the User Interaction Stream

The captured Action Events — which are composed of an Action Symbol and Target Symbol — must be recorded in the order that they are produced to form a User Interaction Stream. These streams can then either be written to disk or written directly into a database. When monitoring users, it was found that a Postrgres database PostgreSQL Global Development Group (2003) running on a SUN ultra 70, was more than capable of logging the events from 25 simultaneous participants in real time, without the users noticing any effects on the application. This satisfies the design objective 5.3, that a capture system should be unobtrusive:-

> The application should not appear to run more slowly when the interaction capture system is running.

6.5 Discussion

A number of different techniques to capture User Interaction Streams from a graphical desktop application were discussed, leading to a decision to target applications written in Java using the Swing Framework.

As a result of this decision, a number of different approaches to capturing Target Symbols were evaluated against the design objectives outlined in chapter 5. Following this evaluation, the JavaHelp approach was chosen and implemented. As part of the implementation process, a tool was created to assist with the task of associating the Target Symbols with graphical widgets.

A methodology for capturing the Action Symbols was also outlined and finally the system for logging the resultant Action Events was detailed.

Chapter 7

Visualisation

This chapter describes two visualisation systems that were developed as part of this research to aid understanding of the data structures and the streams.

7.1 Suffix Structure Visualisation

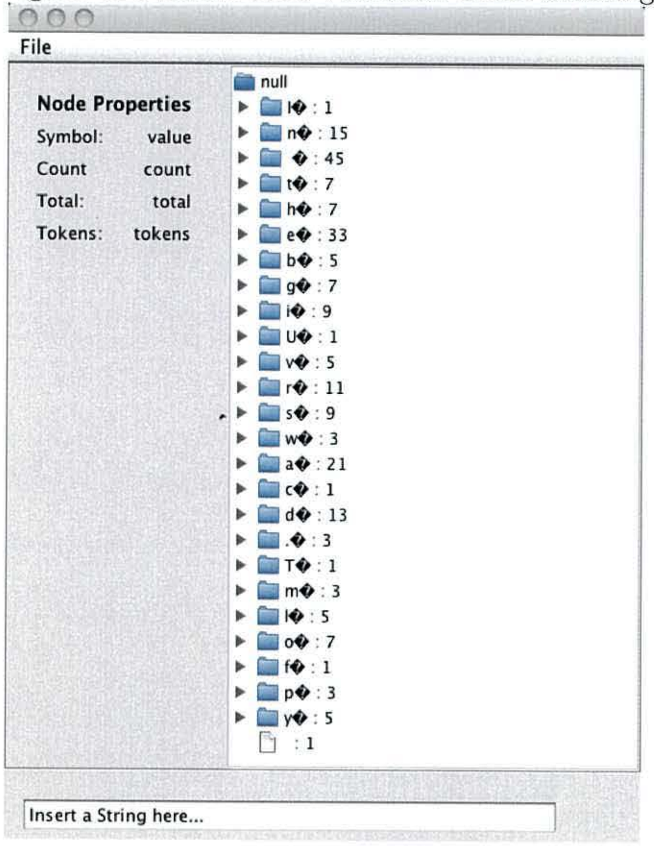
The prediction by partial matching, C measure and R measure algorithm implementations, developed in chapter 4, use a suffix tree structure Knuth (1997) to store the counts needed to calculate probabilities and the counts needed for the other measures. As the suffix tree structure is complex, and therefore hard to ensure its correctness, a tool was developed to allow graphical inspection of the data structure. This tool was then used to manually verify the counts and hence show the correctness of the structure. Figure 7.1 shows the visualiser after starting the application.

When starting the application, the suffix tree is loaded with the phrase:-

> "In the beginning the Universe was created. This has made a lot of people very angry and been widely regarded as a bad move." Adams (1995)

As can be seen in figure 7.1, the tree on the right shows the suffix tree structure. The root node of the tree is labelled *null*. This forms the parent of all the contexts of one character. A symbol has been appended to each character to indicate the end of the character. This was added to distinguish between the various white space

Figure 7.1: Suffix Tree Visualiser After Starting



Node Properties

Symbol:	value
Count	count
Total:	total
Tokens:	tokens

```
null
▶ I� : 1
▶ n� : 15
▶  � : 45
▶ t� : 7
▶ h� : 7
▶ e� : 33
▶ b� : 5
▶ g� : 7
▶ i� : 9
▶ U� : 1
▶ v� : 5
▶ r� : 11
▶ s� : 9
▶ w� : 3
▶ a� : 21
▶ c� : 1
▶ d� : 13
▶ .� : 3
▶ T� : 1
▶ m� : 3
▶ l� : 5
▶ o� : 7
▶ f� : 1
▶ p� : 3
▶ y� : 5
   : 1
```

characters such as line feed, space tab etc. Next to each node there is a count to show the number of occurrences of that node.

When a node has been selected, the node properties are displayed on the left pane. The node properties are:-

Symbol This is the symbol of the current node.

Count The number of occurrences of this symbol in its current context. This count has had count scaling applied, as described by Cleary et al. (1995).

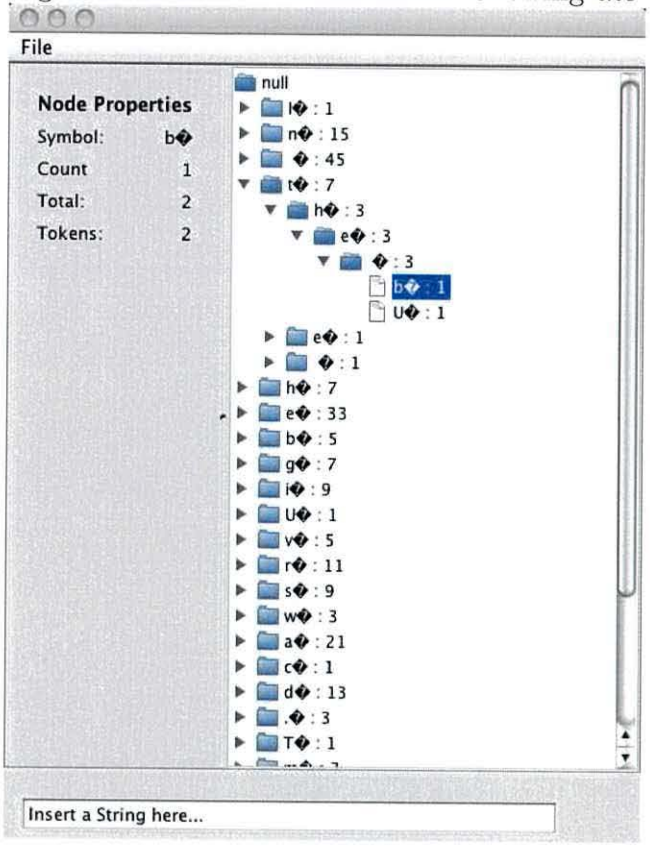
Tokens The number of tokens at this depth.

Total This is the total count of all the nodes at this depth.

Each node of the tree can be expanded to show contexts.

Figure 7.2 shows the full expansion of "the". From this, we can see that the letter *t* is followed by either the letter *h* three times, or by the letter *e* once, or by the space character. From the original phrase, it can be seen that these counts corresponds to the word "the", that occurs twice (count scaling has been applied - hence the count of three) the word "created" and the space after the word "lot".

Figure 7.2: Suffix Tree Visualiser Showing *the*



Node Properties

Symbol:	b�
Count	1
Total:	2
Tokens:	2

```
null
▶ l� : 1
▶ n� : 15
▶  � : 45
▼ t� : 7
    ▼ h� : 3
        ▼ e� : 3
            ▼  � : 3
                b� : 1
                U� : 1
    ▶ e� : 1
    ▶  � : 1
▶ h� : 7
▶ e� : 33
▶ b� : 5
▶ g� : 7
▶ i� : 9
▶ U� : 1
▶ v� : 5
▶ r� : 11
▶ s� : 9
▶ w� : 3
▶ a� : 21
▶ c� : 1
▶ d� : 13
▶ .� : 3
▶ T� : 1
```

As can be seen, the tree has only one branch until after the space character that always occurs following the complete word. At this point, there are two leaf nodes *b* and *u*. These leaf nodes represent "the beginning" and "the universe". The selected symbol is *b*. As can be seen, there are two different tokens at this level. Each token occurs once, hence the total of two. The effect of count scaling can be seen, as the leaf nodes both have counts of one, whereas their common parent has a count of three. At the bottom of the window there is a text field. This field allows a user to enter text and then have this converted into the appropriate suffix tree structure that is then visualised. The tool also allows a user to load a text file and visualise the resulting suffix tree structure.

7.2 Stream Visualisation

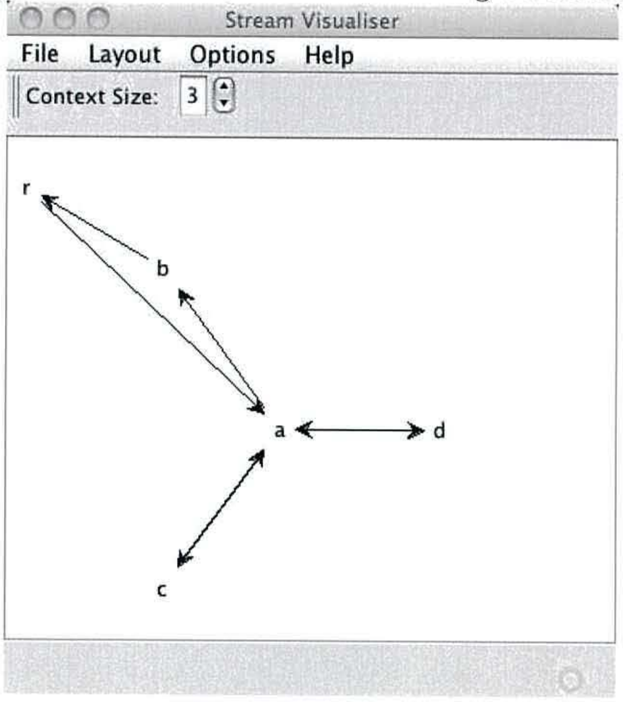
The Stream Visualisation tool has been developed to show the relationship between unique symbols in a variable order Markov Model in a graphical form. The tool shows a graphical representation of the symbols in a stream, with directed arrows between symbols. The tool can be used to visualise streams at a character level and at a word level. This section will show the operation and output of the tool, looking first at the character level and then at the word level. Finally, a demonstration of the tool being used on streams produced by the event capture system will be outlined.

7.2.1 Character Visualisation

This section will introduce the stream visualisation tool and show its operation on character streams of English text. The first view of the stream visualiser is shown in figure 7.3.

This shows the tool in its default state, with the stream *abracadabra* displayed. As can be seen, each letter forms a node of the graph and the edges show relationships to other characters. The letter a is followed by b, c, *or* d. This can be seen by following the edges from a.

Figure 7.3: Stream Visualiser Showing *abracadabra*



Context Size: 3

Figure 7.4: Stream Visualiser Showing "the cat sat on the mat".

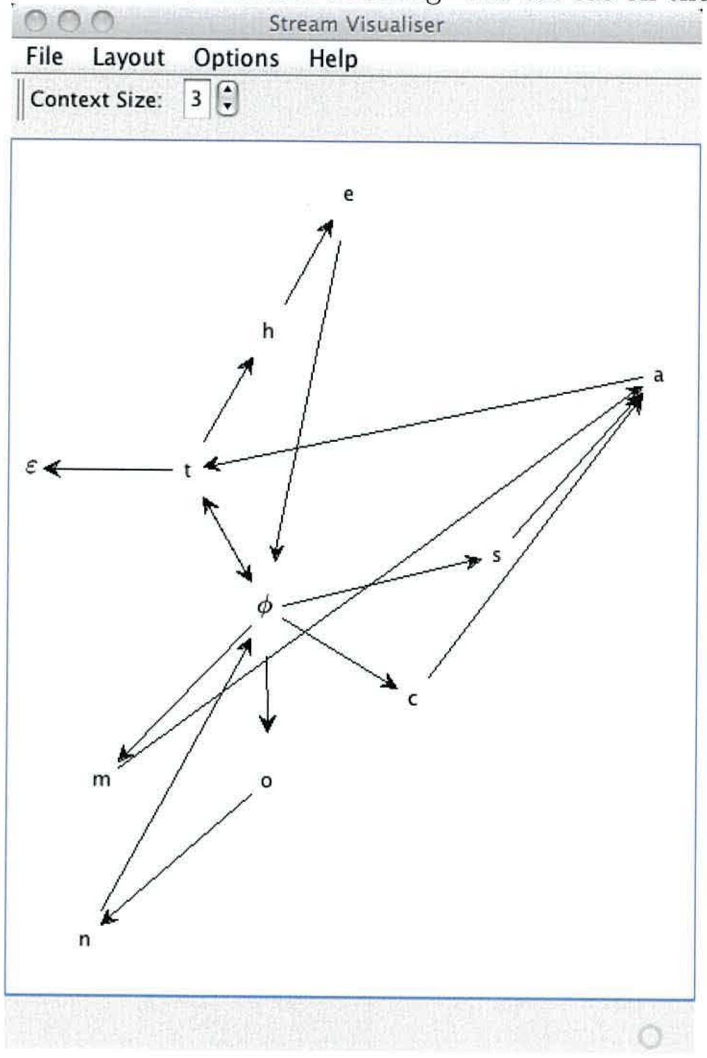


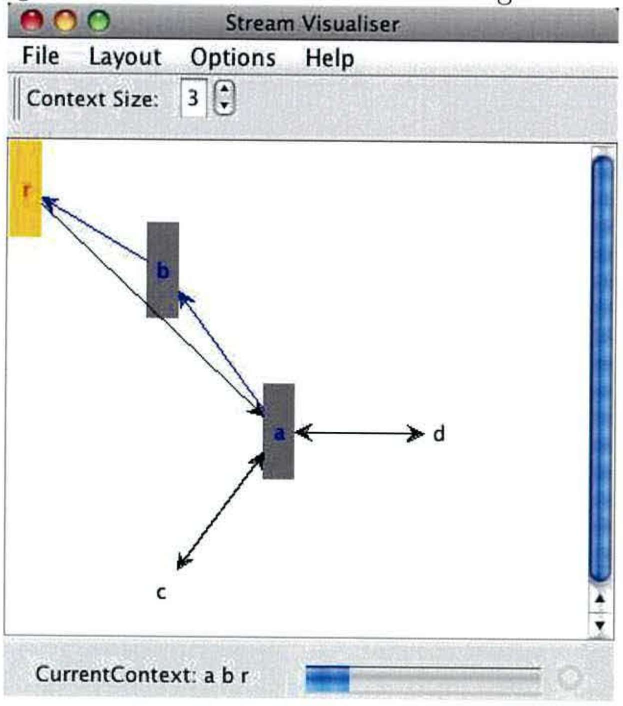
Figure 7.4 shows the phrase "theϕcatϕsatϕonϕtheϕmatε" loaded as characters. The node in the center of the graph is the space character. This has been substituted with the symbol ϕ. There is another white space character, the carriage return. This has been substituted with the symbol ε.

The basic view shows the structure of a file that has been loaded. However, the tool provides a way to visualise contexts, because the stream is stepped through character by character. The control labelled context size allows a user to select the size of the context to be displayed. By default, this is 3 characters. When a user has selected a context size, then they can select *watch* from the file menu. This starts the playback of the current stream, highlighting context along the way.

Figure 7.5 shows the playback at the point where the context consists of the first three characters of *abradabra*, i.e. *abr*.

The current character is *r*. This node is highlighted orange. The gray nodes show the previous context. The edges highlighted in blue, are the edges involved in the current context. The display shows the current context. This is shown on the progress bar at the bottom of the window. To the right of context, a progress bar shows the position of the current context relative to the entire length of the stream.

Figure 7.5: Stream Visualiser Showing Context.



Context Size: 3

CurrentContext: a b r

7.2.2 Word Visualisation

As has already been mentioned, the Stream Visualiser can also display streams based on words rather than characters. This section will demonstrate the use of the tool on streams of English text with words rather than characters.

Figure 7.6 shows the tool with the phrase "the cat sat on the mat" loaded as words.

As the phrase is short there is very little repetition. The word *the* is followed by both *cat* and *mat*.

Figure 7.6: Stream Visualiser Showing "the cat sat on the mat" as Words.

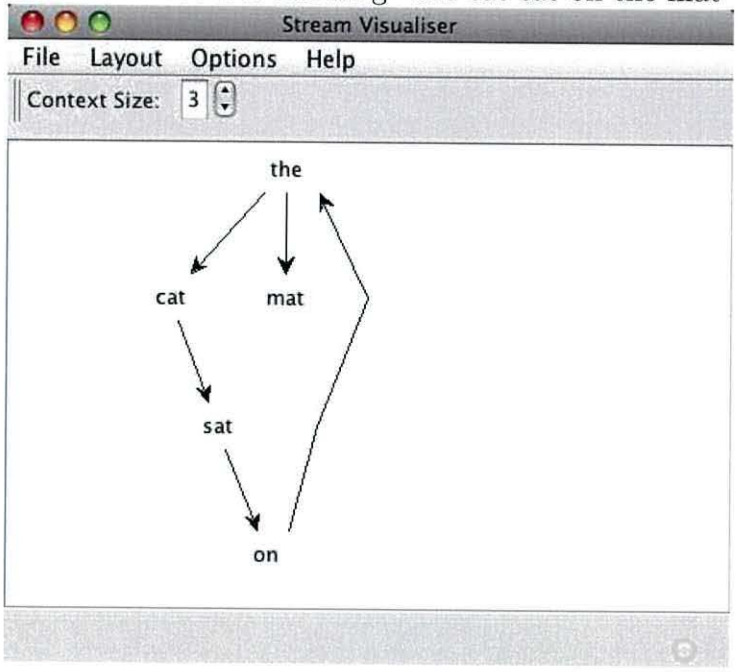
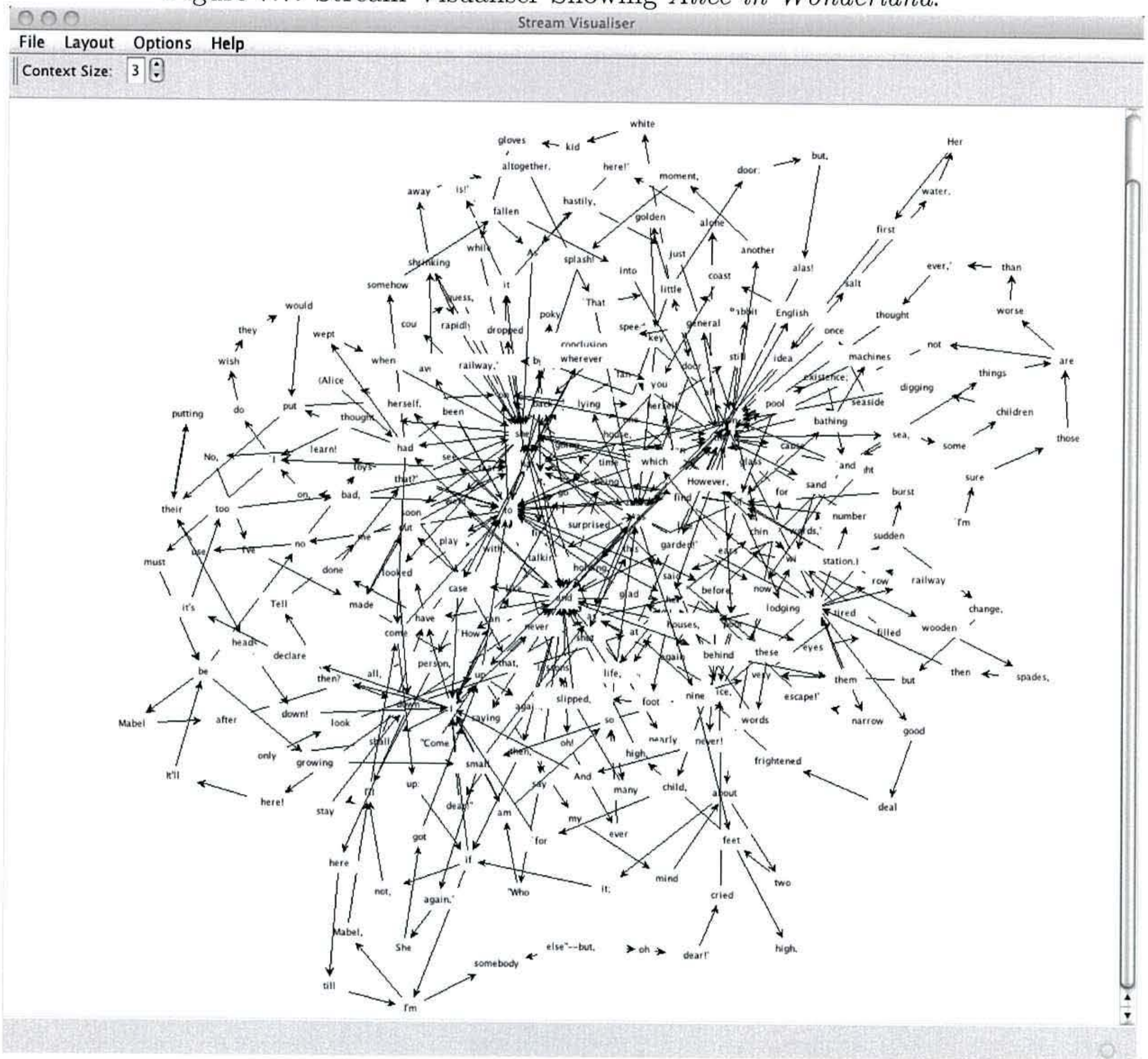


Figure 7.7 shows the first section of Alice in Wonderland Carroll (1865).

As can be seen there are many more words than in the simple example shown above. As a result, the graph has been zoomed out to show all the words.

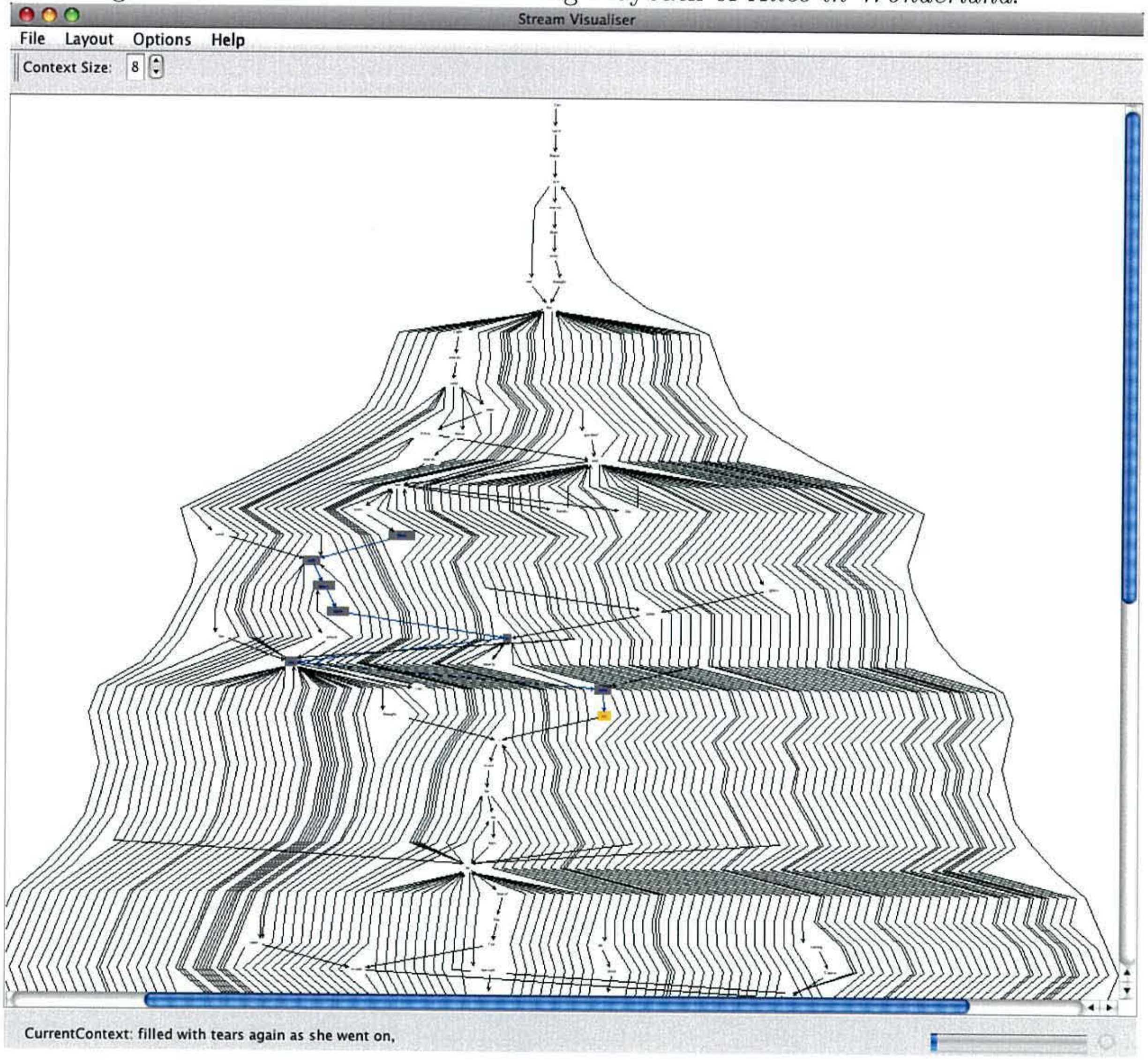
Figure 7.7: Stream Visualiser Showing *Alice in Wonderland*.



As when visualising characters, the tool allows a user to playback the input stream. This is shown in figure 7.8.

Again, the graph has been zoomed out to show more detail. The status label at the bottom of the screen shows the current context: "filled with tears again she went on."

Figure 7.8: Stream Visualiser Showing Playback of *Alice in Wonderland*.

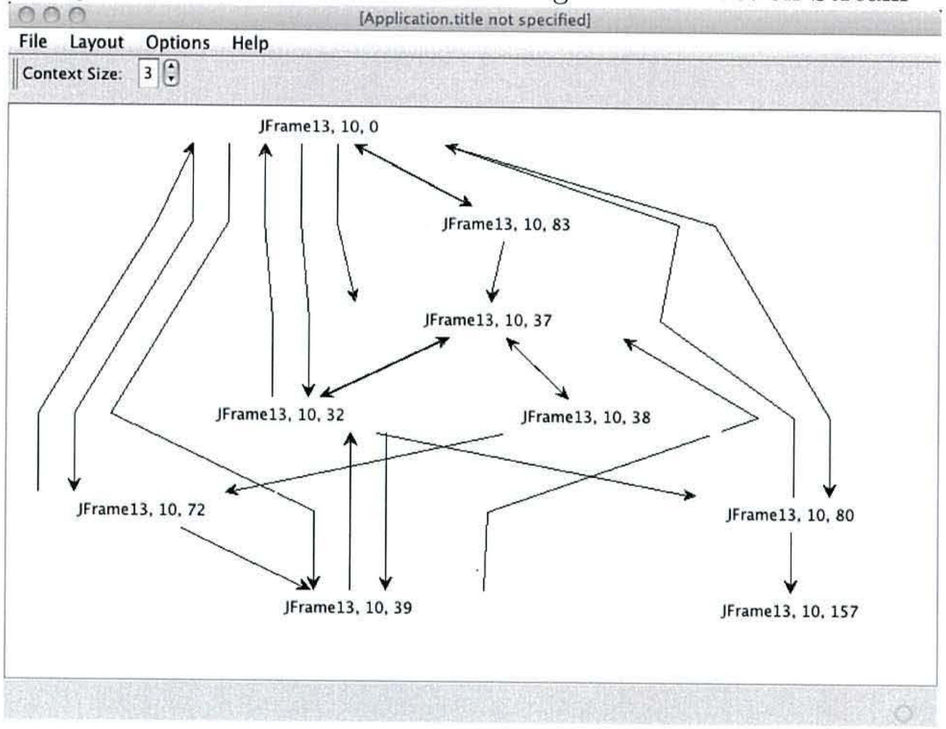


CurrentContext: filled with tears again as she went on,

7.3 User Interaction Stream Visualisation

The Stream Visualiser can also be used to visualise the User Interaction Streams produced by the capture system, as described in chapter 5. A User Interaction Stream is made up of Action Events, where each Action Event consists of a Target Symbol and an Action Symbol. The Target Symbol and Action Symbol are analogous to letters in a natural language (section 5.5). The Target Symbol, Action Symbol and resulting Action Event are not easily representable in a meaningful visual way. The most meaningful way would be to capture images of each component that each taget symbol represents and display these. This would need to be carried out by hand after taking a screen shot of the application. Figure 7.9 shows the visualisation tool with a User Interaction Stream loaded.

Figure 7.9: Stream Visualiser Showing User Interaction Stream



The User Interaction Stream displayed was captured by playing the game Asteroids (described later, in chapter 8.) This diagram is not very useful because it is not very clear what the symbols (the node labels) correspond to. To enhance the meaning of the diagram, the diagram has been combined with the User Interaction Playback mechanism, described in chapter 10. This combination allows a developer to see patterns as a user interacts with the application. For this reason videos of the visualiser have been produced to accompany this thesis.

7.4 Discussion

Two different techniques for visualisation were developed. The first was developed to ease the checking of data structures used in the algorithms for text categorisation. The second visualiser showed the interrelationship between components of a stream. This visualiser can also be used to visualise the User Interaction Streams. The playback mechanism described in chapter 10 was coupled to this visualiser, allowing a user to see the stream as it is being used.

Chapter 8

Creating the BUIS Corpus

8.1 Research Proposal

A corpus was created, called the BUIS corpus, to experimentally evaluate the User Interaction Streams captured during the use of an application. This corpus has been made available to other researchers *Bangor User Interaction Stream Corpus* (2008). As outlined in the thesis statement "it is hypothesised that the interactions between a user and a graphical desktop application will be similarly unique." To examine this hypothesis a new corpus was produced and subsequently analysed. This chapter describes the rationale and selection of two applications. The choice of user groups and the experimental settings are then outlined and this results in the production of the new corpus described.

As described in chapter 6, a mechanism for capturing User Interactions has been developed. The capture mechanism is applicable to any application written in Java using the Swing toolkit. This chapter describes the rationale for choosing two contrasting applications to create a corpus of User Interaction Streams that can then be analysed. The two applications have contrasting user interfaces. The first has a complex graphical interface and the second a much simpler interface.

8.2 Rationale for Selecting an Application

In order to choose an application to perform the experimental analysis in chapter 9, it is important that any chosen application satisfies a number of criteria, as outlined below.

8.2.1 Rationale for Selecting Application One

Source Code Language

To use the capture system developed in chapter 6, the application must: *a*) be written in Java *b*) must use Swing toolkit to provide the user with a graphical interface. *c*) ideally, it should have no existing context sensitive help. In order to capture the User Interaction Stream, an application must satisfy the first two criteria. The third criteria, though not mandatory, is desirable as it allows the use of the automated tool to tag the Target Symbols (discussed in chapter 6 section 6.4.2)

Source Code Availability

In order to modify the application to record User Interaction Streams, the source code must be available. It is also desirable that the source code is available to others so that the experiment can be repeated and extended. Ideally this means that the application should be available under an open source or free software license, such as the GNU GPL Free Software Foundation (2007).

User Base

For the purpose of this research, it is necessary to choose a set of users who are all using the same application to perform the same task. If users were performing different tasks, then it would be difficult to differentiate between the authorship and task identification problems. It is also important that the application has been tried and tested and is therefore stable. Furthermore, it is desirable that the users be available to use the application under conditions where their usage can be recorded. This is discussed in more detail later in section 8.4.1.

Application Interface Complexity

An application with many UI components is desirable as this increases the potential number of interactions. Simple applications such as the calculator, given as an example in chapter 5, have a limited range of User Interaction Streams. Simple applications also limit the number and types of tasks that users can be asked to perform. As this would result in smaller User Interaction Streams, a more complex application is desirable. However, the complexity of the application must not compromise the potential user base. i.e. It must not be so complex that there are very few users of the application. The application should still be user friendly.

8.2.2 Rationale for Selecting Application Two

This section describes the selecting of the second application. This application is much less complex than the first application and is chosen explicitly for the second experiment. This application meets the following criteria:

Source Code Language

The application must be written in Java and use the Swing Toolkit.

Source Code Availability

The source code must be available with a licence that allows redistribution.

User Base

A pool of users must be available to run the application. Finding a suitable simple application with a captive user group is problematic. There are few simple commonly used applications. A desktop calculator application, similar to the one used to explain concepts in chapter 5, could have been used. Although calculators are used frequently, their usage is sporadic and unpredictable. For this reason, it was decided to try and find a computer game with a simple interface instead. Users could be encouraged to form a league and be rewarded for playing the game by competing with each other to get the highest score, thus rewarding constant use over a short period of time.

Application Interface Complexity

The application should have a simple user interface to meet the criteria for the second experiment.

8.3 Selecting the Applications

8.3.1 Selecting Application One — WEKA

WEKA (Witten & Frank 2005) was chosen as it fulfils all the requirements discussed above (8.2.1).

WEKA is described by its authors as follows:

> *WEKA is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from your own Java code. WEKA contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes.*

WEKA provides a graphical interface to a collection of machine learning algorithms.

Source Code Language

WEKA is written in Java and uses the swing graphical framework, satisfying the requirement 8.2.1. In addition it also has no context sensitive help.

Source Code Availability

WEKA is distributed under the GNU GPL licence, satisfying the second requirement.

Application Interface Complexity

The interface for WEKA is complex, satisfying the complexity requirement as stated in section 8.2.1. The complexity of the WEKA interface can be seen in the section below (8.5.1) detailing the tasks that the users perform.

User Base

The last consideration is the user base. WEKA has a large user base across the world. WEKA also forms a key component of the Artificial Intelligence course at the University of Wales Bangor. Students must use WEKA to perform a series of experiments, in order to familiarise themselves with some of the algorithms and techniques that WEKA provides. As a result, they represent a captive test group. In addition, as they are made up of Computer Science students, they have some similar previous experience of using computers, though it must be noted that there is still a fairly wide range of computing experience within this general level.

8.3.2 Selecting Application Two — Asteroids

A version of the classic computer game Asteroids was chosen. The reasons this application meets the objectives outlined above will now be evaluated. Asteroids has been written as an example application for the Xito application manager *The Xito platform* (2005). It is a Java version of the popular classic computer game Asteroids. The Asteroids interface is shown in figure 8.1.

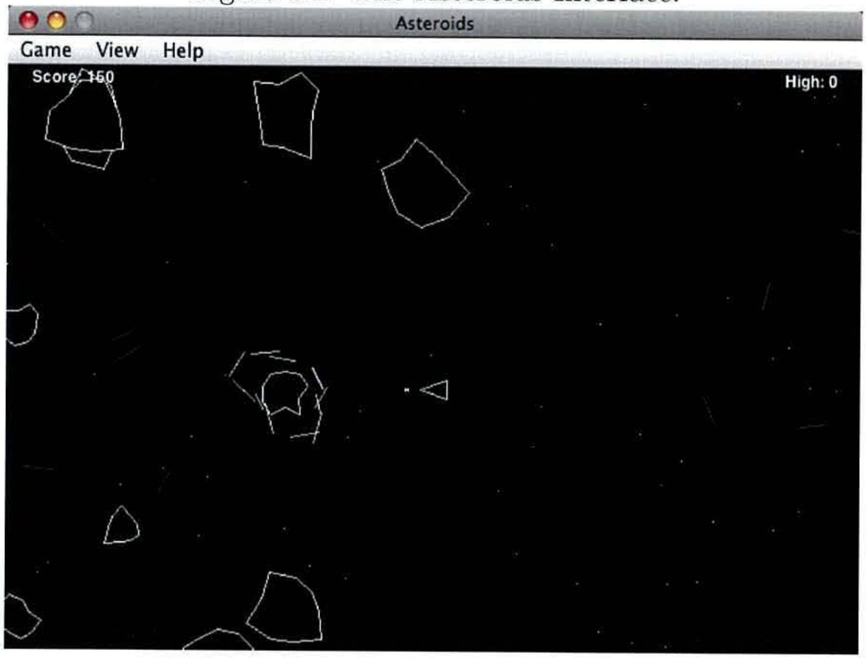
Source Code Language

It is written in Java and uses the Swing Toolkit and has no Java help.

Source Code Availability

The source code is public domain so the application is freely distributable.

Figure 8.1: The Asteroids Interface.



Application Interface Complexity

The game has a few simple user interface components.

User Base

The game, being a classic, is familiar to many people and, as it has a high score system, users are rewarded for playing the game.

8.4 Users

In this section, the rationale for selecting user groups for each application is evaluated. The User groups are then described.

8.4.1 WEKA User Group

Choosing a WEKA User Group

As has already been discussed in the section above, the choice of application was heavily influenced by the availability of a user group within the timescale parameters of this research. It would have been desirable if each user could use the application in a HCI lab (as described in chapter 2 section 2.5.1). The recorded stream could then be augmented with lots of additional meta data, such as user experience, verbal comments, eye tracking etc. But, as has already been discussed, (section 2.5.1) these facilities are expensive and significantly intrude on the user using the application. Additionally at the time of performing this experiment, there was no available access to these facilities. The experiment had to be designed with this fact in mind.

Given the fact that the Interaction capture system was capable of capturing the interactions of many users simultaneously and unobtrusively, by choosing the task of authorship attribution with a user group of similar abilities performing the same task, the requirement of carrying out a traditional HCI study becomes unnecessary. The required meta data (the stream author) is available without the need of specialist tools.

User Group Chosen

As has already been noted, the chosen application WEKA forms a key component of the Artificial Intelligence module, taken as part of the Computer Science course at Bangor University. The application is invariably new to the students, so, although there is some variance in the general computer usage experience of the users, none had prior exposure to WEKA.

This captive group of naive users were all required to complete a lab script in order to finish the course. The majority of the students chose to complete the Lab using the computing facilities provided by the department. As a result, the application that they ran was under the control of the lab supervisors.

The course ran in the same form for two years, with the same lab script. This provided an ideal User Group that fitted the time and money constraints of the research, as well as meeting important criteria related to the choice of application. In total, sixteen unique users completed the lab session over two years.

8.4.2 Asteroids User Group

Choosing an Asteroids User Group

This group was selected from a volunteer group of experienced computer users, competent in playing computer games. The high scoring system of the game was modified, in order to allow the automatic sharing of high scores, thus producing a competitive environment for playing the game. This ensured a large number of interactions for analysis were performed by each player.

User Group Chosen

The location for the game was sent to a group of users who enjoyed playing computer games and also to the staff of a software development company. Of the users invited to play the game, a user group consisting of ten users who were motivated to play the game on multiple occasions, was self selected.

8.5 Experimental Setting for WEKA

In this section, the experimental setting is described. WEKA was chosen as an application and students undertaking an Artificial Intelligence course provided the user group. As part of the course, students had to complete a number of laboratory assignments. Two of these assignments (labs nine and ten) introduced the students to WEKA. The lab was timetabled for 3 hours a week, during which the Course Lecturer and two Postgraduate Students were available to provide assistance. In addition to the timetabled labs, the students could work on the assignments in their own time, either at the laboratory or at home. Only those students who used the computer systems laboratory had their events logged. Students who worked at home did not have their events logged. The experiment ran for two years, with a different batch of students each year. The total number of students over the two year period was 16. The section below describes the script that users followed to complete the assignment.

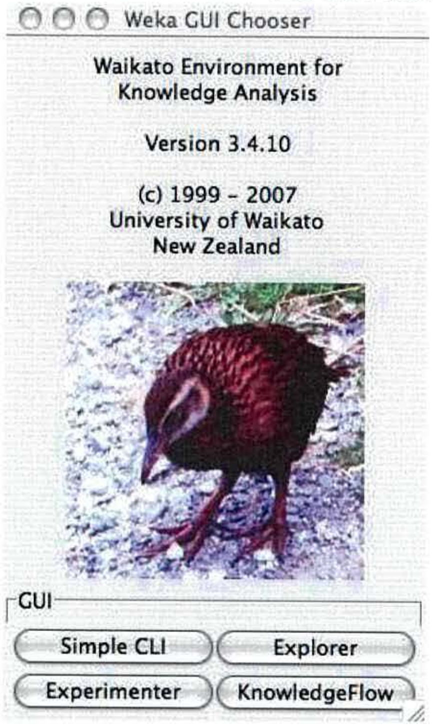
8.5.1 Lab Script

The users followed a lab script. The full script can be found in appendix A. The script is very prescriptive, telling users where to click. Below is a walk-through of that script as followed by the students. The titles of the sections below correspond to the task titles from the lab script available in appendix A. Upon completing a task, the user was asked to note this in a separate window. The completion of the task was recorded as part of the corpus.

Part 2: Starting WEKA

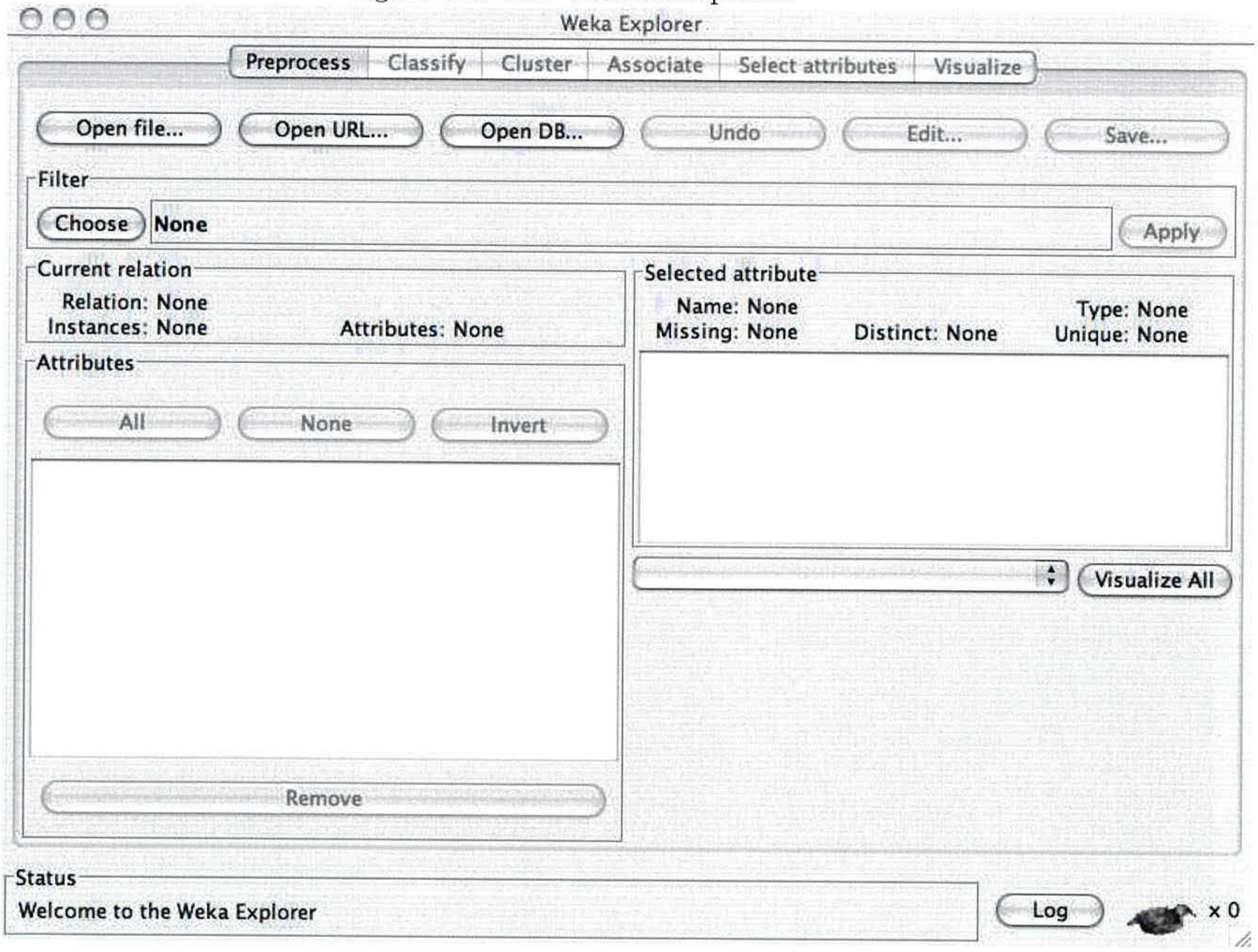
After starting WEKA a user is presented with the screen shown in figure 8.2.

Figure 8.2: The WEKA Chooser Interface.



The script then asks the user to start the explorer by clicking on the explorer button. The explorer window, shown in figure 8.3, is then displayed. The user is then asked to explore the user interface and note any comments about the interface.

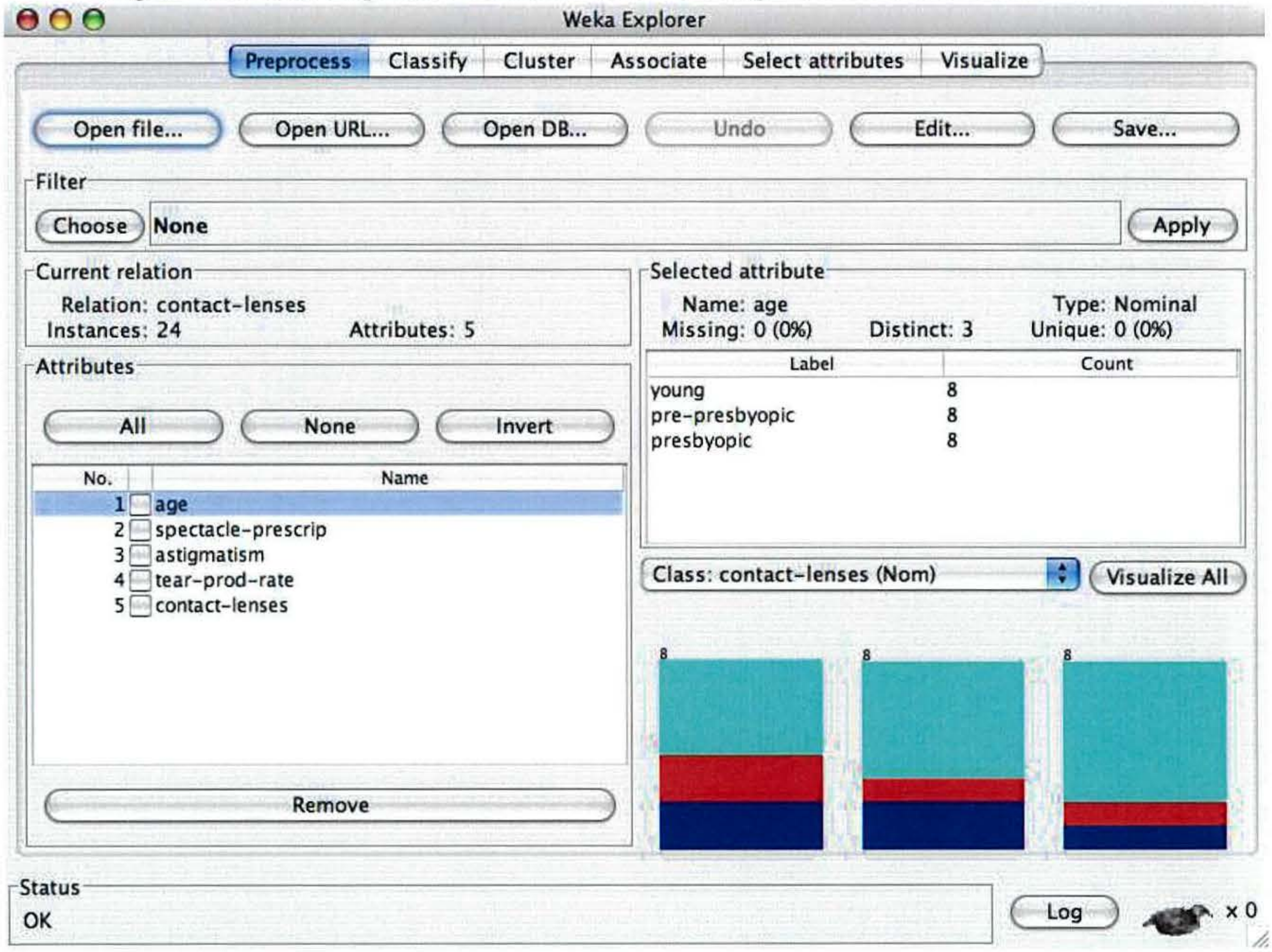
Figure 8.3: The WEKA Explorer.



Part 3: Loading a data file

The user is asked to load a data file into WEKA by clicking the "open file" button and locating the "contact-lenses.arff" file.
The user is then presented with the screen shown in figure 8.4.

Figure 8.4: The Explorer Interface After Loading Contact Lens Data.



Filter: None

Current relation
Relation: contact-lenses
Instances: 24 Attributes: 5

Selected attribute
Name: age Type: Nominal
Missing: 0 (0%) Distinct: 3 Unique: 0 (0%)

Label	Count
young	8
pre-presbyopic	8
presbyopic	8

Attributes

No.	Name
1	age
2	spectacle-prescrip
3	astigmatism
4	tear-prod-rate
5	contact-lenses

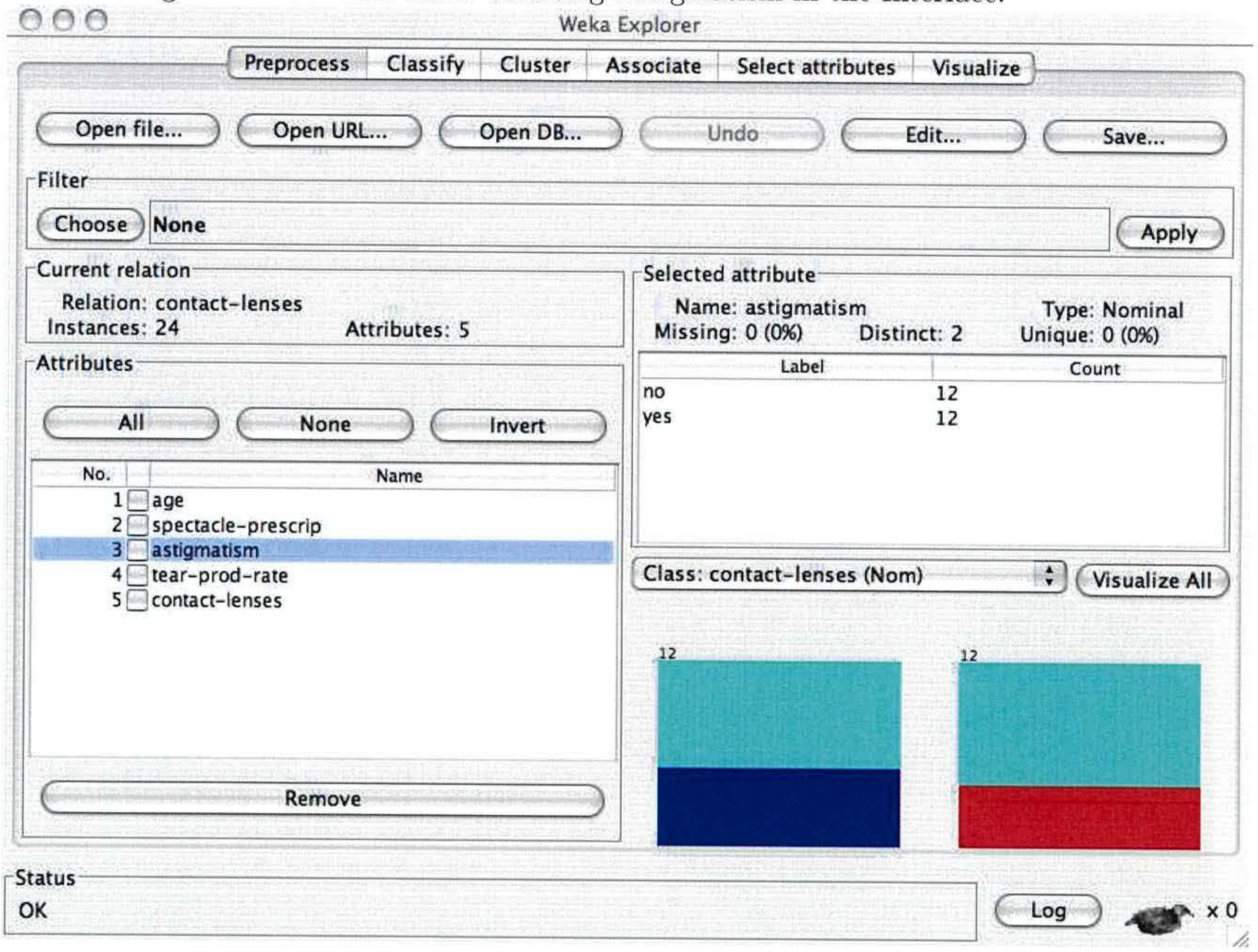
Class: contact-lenses (Nom)

Status
OK

The user is then asked to look at the different attributes. Figure 8.5 shows the astigmatism attribute display.

When a different attribute is selected, the relevant attribute information and associated graphs are displayed.

Figure 8.5: WEKA: After Selecting Astigmatism in the Interface.



Filter: None

Current relation
Relation: contact-lenses
Instances: 24 Attributes: 5

Attributes
All None Invert

No.	Name
1	age
2	spectacle-prescrip
3	astigmatism
4	tear-prod-rate
5	contact-lenses

Selected attribute
Name: astigmatism Type: Nominal
Missing: 0 (0%) Distinct: 2 Unique: 0 (0%)

Label	Count
no	12
yes	12

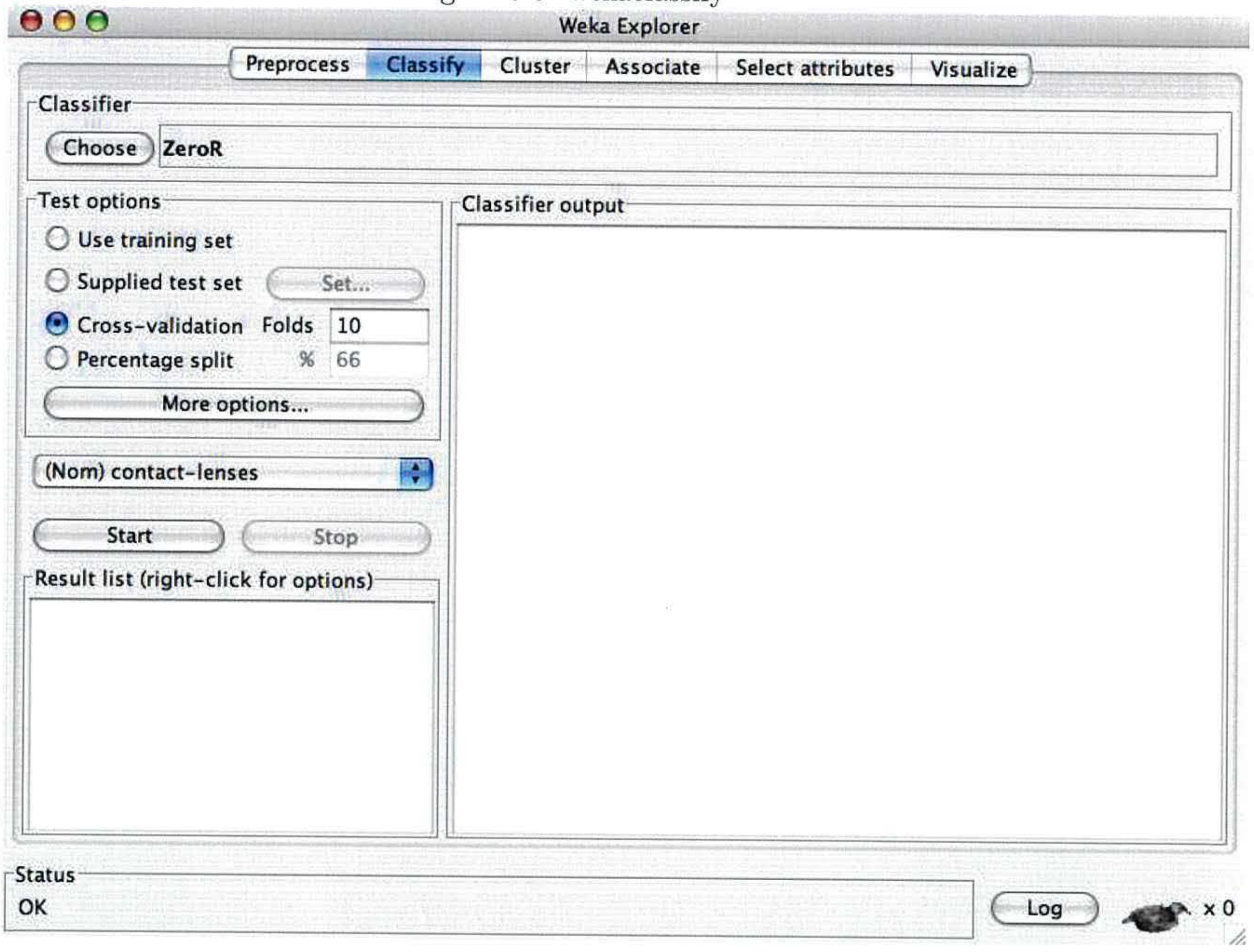
Class: contact-lenses (Nom)

Status
OK

8.5.2 Part 4: Performing Classification

The user is asked to switch to the classifier tab. Figure 8.6 shows what the user was presented with after they had switched to this tab.

Figure 8.6: wekaclassify



Classifier: ZeroR

Test options: Cross-validation, Folds 10
Percentage split % 66

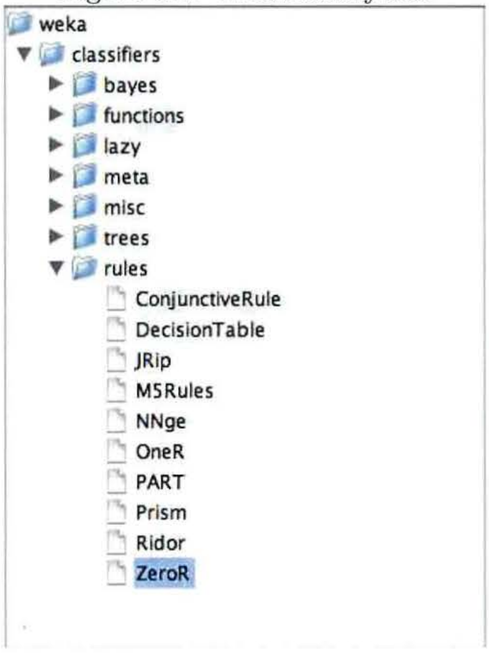
(Nom) contact-lenses

Status: OK

The user is then told to choose a classifier by clicking on the "Choose" button. This brings up the tree view shown in figure 8.7.

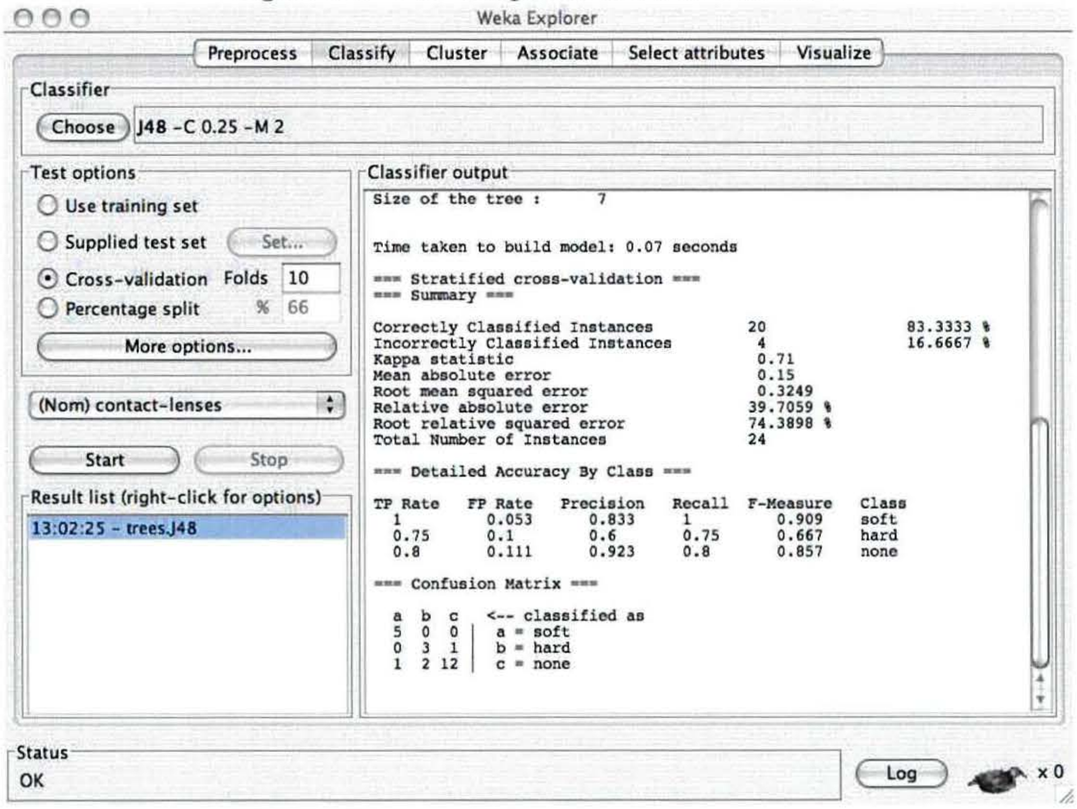
The user is told to pick the *weka.classifiers.trees.J48* classifier. After selecting this option the user is asked to ensure that the default option of "Cross-validation" is set.

Figure 8.7: wekaclassifytree



The user is asked to record the output of the classifier. An example of such output is shown in figure 8.8.

Figure 8.8: The Output of the J48 Classifier.



Classifier: J48 -C 0.25 -M 2

Test options:
- Use training set
- Supplied test set
- [✓ Cross-validation] Folds 10
- Percentage split % 66

(Nom) contact-lenses

Result list (right-click for options): 13:02:25 - trees.J48

Classifier output

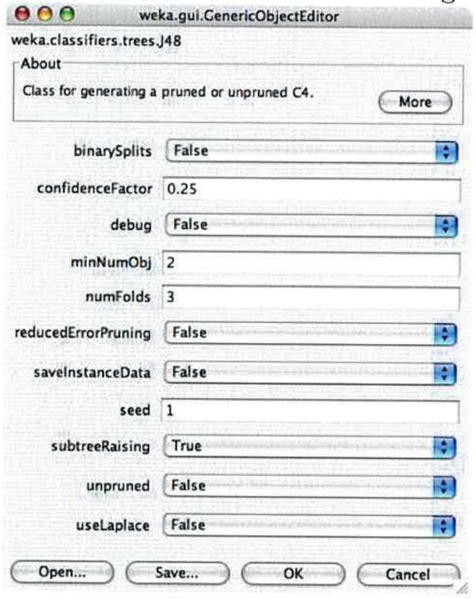
```
Size of the tree :      7


Time taken to build model: 0.07 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          20               83.3333 %
Incorrectly Classified Instances         4               16.6667 %
Kappa statistic                          0.71
Mean absolute error                      0.15
Root mean squared error                  0.3249
Relative absolute error                 39.7059 %
Root relative squared error             74.3898 %
Total Number of Instances               24

=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  1         0.053     0.833     1         0.909     soft
  0.75      0.1       0.6       0.75      0.667     hard
  0.8       0.111     0.923     0.8       0.857     none

=== Confusion Matrix ===

  a  b  c   <-- classified as
  5  0  0 |  a = soft
  0  3  1 |  b = hard
  1  2 12 |  c = none
```

Status: OK

Part4: Task Three

The user then experiments with the effects of changing the parameters of the `J48` algorithm. Figure 8.9 shows the dialogue box they are presented with to enable them to change the parameters.

Figure 8.9: The J48 Parameters Dialogue Box.



weka.gui.GenericObjectEditor

weka.classifiers.trees.J48

About

Class for generating a pruned or unpruned C4.

binarySplits	False
confidenceFactor	0.25
debug	False
minNumObj	2
numFolds	3
reducedErrorPruning	False
saveInstanceData	False
seed	1
subtreeRaising	True
unpruned	False
useLaplace	False

Part 4: Task Four

The user is told to change the classifier to `divide-and-conquer` decision tree algorithm, ID3 *weka.classifiers.trees.Id3* . This is done by selecting ID3 from the tree shown in figure 8.7.

After selecting the alternate classifier, the user is asked to note differences between the classification made by the J48 and ID3 on the same data set.

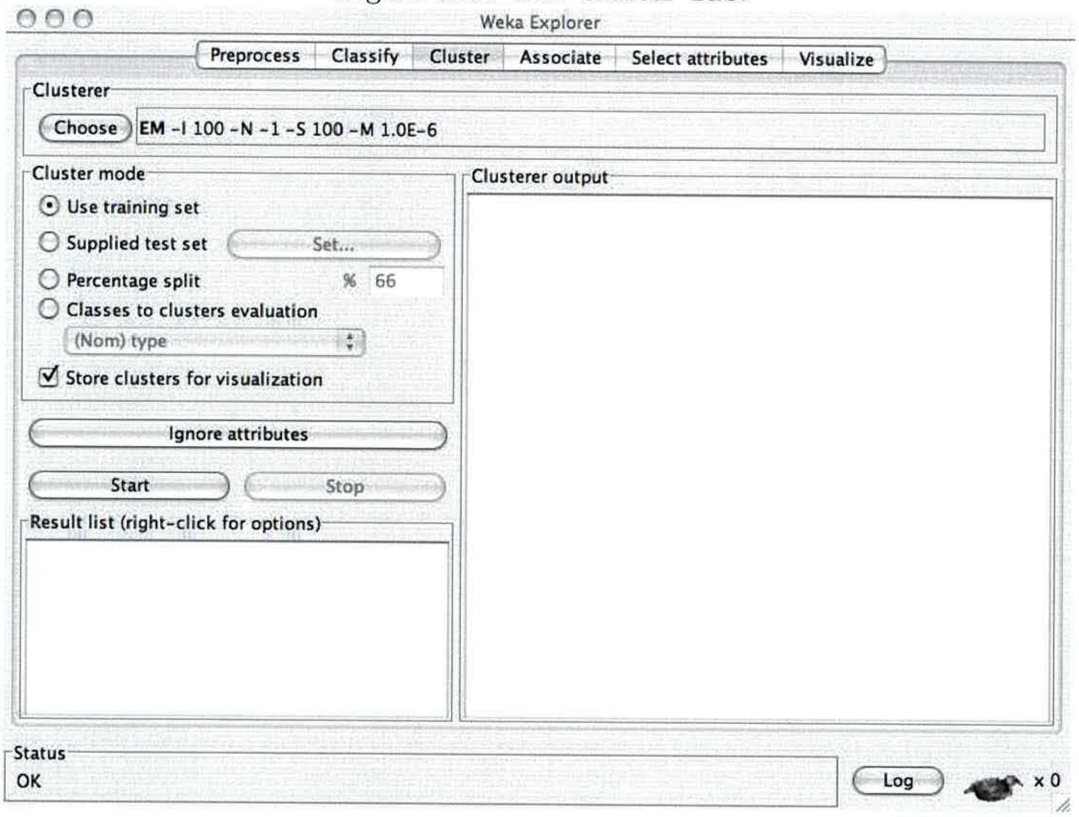
Part 4: Task Five

The user is asked to experiment with different classifiers from the tree in figure 8.7.

8.5.3 Part 5: Clustering

In this part, the user is asked to use the cluster tab in WEKA. The user is told to load the "zoo.arff" file, then switch to the *cluster* tab. The cluster tab is shown in figure 8.10.

Figure 8.10: The Cluster Tab.


Weka Explorer

Preprocess | Classify | Cluster | Associate | Select attributes | Visualize

Clusterer

Choose EM -I 100 -N -1 -S 100 -M 1.0E-6

Cluster mode

- [✓ Use training set]
- Supplied test set — Set...
- Percentage split — % 66
- Classes to clusters evaluation — (Nom) type
- [✓ Store clusters for visualization]

Ignore attributes

Start Stop

Result list (right-click for options)

Clusterer output

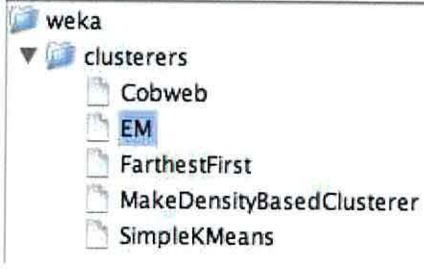
Status

OK

Log x 0

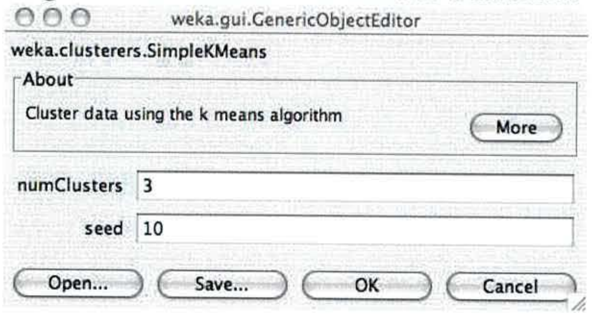
The user is then asked to select the *SimpleKMeans* from the cluster tree shown in figure 8.11. Before running the cluster algorithm, the user is asked to change the

Figure 8.11: Cluster Algorithm Selection Dialogue Box.



cluster parameters. After bringing up the cluster parameters dialogue box, the user is asked to change the number of clusters to three in the dialogue box, as shown in figure 8.12.

Figure 8.12: Cluster Parameter Selection.



weka.clusterers.SimpleKMeans

Cluster data using the k means algorithm

numClusters 3

seed 10

Part 5: Task Six

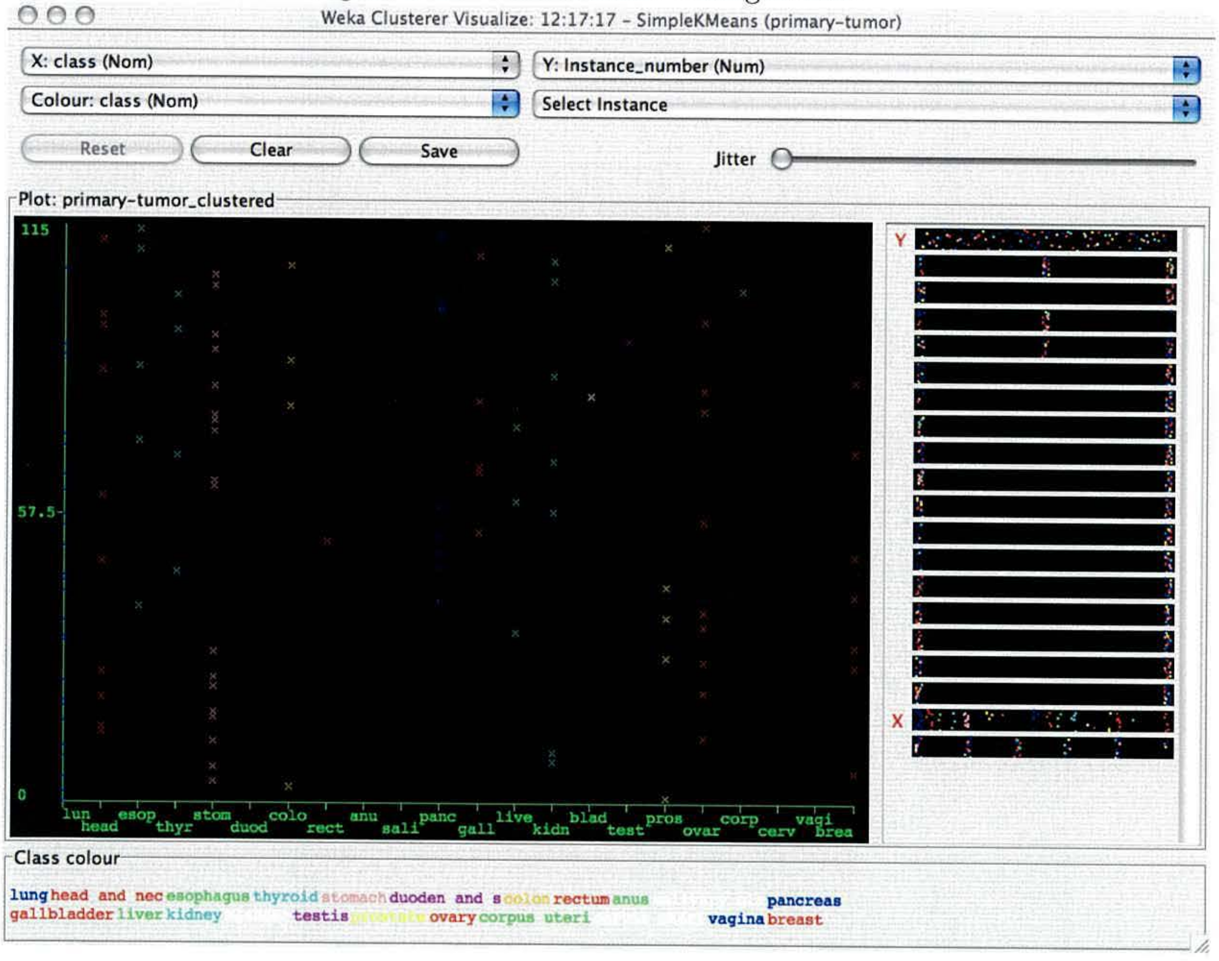
The user is told to use the cluster tab and experiment with a different data file, “primary-tumor.arff”.

Part 5: Task Seven

This task encourages the user to explore the *visualize* tab. The user is asked to do this using the “primary-tumor.arff” that they used in part six.

There was a discrepancy between the instructions and the interface provided by WEKA. This was because the instructions were written for a previous version of WEKA. To get to the visualization, a user had to right click on the results list and select *visualize cluster assignments.* This then presented them with the dialogue box shown in figure 8.13.

Figure 8.13: Visualisation Dialogue Box.



Weka Clusterer Visualize: 12:17:17 - SimpleKMeans (primary-tumor)

X: class (Nom)

Y: Instance_number (Num)

Colour: class (Nom)

Select Instance

Plot: primary-tumor_clustered

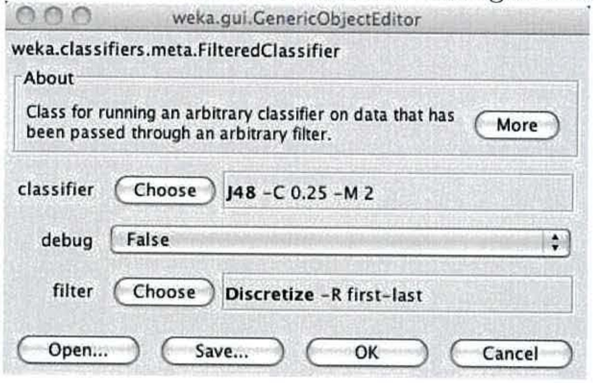
Reset
Clear
Save
Jitter
115
57.5
0
lun
head
esop
thyr
stom
duod
colo
rect
anu
sali
panc
gall
live
kidn
blad
test
pros
ovar
corp
cerv
vagi
brea
Y
X
Class colour
lung head and nec esophagus thyroid stomach duoden and s colon rectum anus pancreas
gallbladder liver kidney testis prostate ovary corpus uteri vagina breast

8.5.4 Lab Ten

The aim of this lab is to introduce the user to text categorisation tasks using WEKA. The user is asked to load *ReutersCorn-train.arff*. As this is text data it requires pre-processing. To do this the user is asked to select the *weka.classifiers.meta.FilteredClassifier* from the dialogue box shown in figure 8.7.

After selecting the Filtered classifier, the user has to select the filter parameters. The filter parameters dialogue box is shown in figure 8.14.

Figure 8.14: Filter Parameters Dialogue Box.


weka.gui.GenericObjectEditor

weka.classifiers.meta.FilteredClassifier

About

Class for running an arbitrary classifier on data that has been passed through an arbitrary filter.

classifier: J48 -C 0.25 -M 2

debug: False

filter: Discretize -R first-last

The filter *filters.unsupervised.attribute.StringWordVector* is used in combination with the *classifiers.bayes.NiaveBayesMultinomial* classifier. After setting the filter and classifier options, the user is told to specify a specific file for testing: *ReutersCorn-train.arff*. When the user has run this analysis, the user is asked to repeat the experiment with different train and test files:-

Corporate Acquisitions.

Crude Oil.

Grain.

The rest of the lab consists of experimenting with the data files above, and with different classifiers, from the following list:-

Support Vector Machines *weka.classifiers.functions.SMO*

Nearest Neighbour *weka.classifiers.lazy.IBk*

decision tree *weka.classifiers.trees.J48*

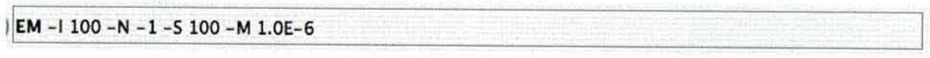
8.5.5 The Experiment

The lab was undertaken by sixteen students over two years. This section describes how the events were recorded and observations made by the support staff whilst the students carried out the lab assignment.

8.5.6 Experimental Observations

As stated in the rationale for choosing an application, a complex application was desirable. The complexity of the WEKA interface has been demonstrated by two of the tasks outlined above. The laboratory assistants all noted that many of the students found two components of the WEKA interface unintuitive. The two problems appear to share the same root cause. The cause appears to be the overloading of a JLable component, to show not just text data, but to also function as a button. Figure 8.15 shows the dual purpose label.

Figure 8.15: This label also functions as a button.


EM -I 100 -N -1 -S 100 -M 1.0E-6

The first of these tasks was described in section 8.5.2. This task involved changing the classifier parameters. The other task that caused difficulties for many users was the task of modifying the cluster parameters (described in section 8.5.3). It was not clear to the user that clicking the label shown in figure 8.15 next to the chosen button would have an effect.

8.5.7 Application Modification

The application chosen in section 8.2 was modified with the automated tool (described in section 6.4.2). The tool assigned a unique Target Symbol to each of the graphical widgets that make up the WEKA interface. After assigning target symbols, the next stage was to modify the application so that Action Symbols were recorded. This was done by modifying the *main* method. The modification to the *main* method ensured that, on starting the application, the logging component, (described in section 6.4.4) moved to the top of the Java Event Queue. The component was configured so that after creating an Action Event, the Action Event was then logged to a database.

8.5.8 Experimental Notes

Task two (section 8.5.1) asked the user to experiment with the WEKA interface. This task was non prescriptive and as such the events recorded when performing this task were not used in the analyses performed in chapter 9.

8.6 Experimental Setting for Asteroids

As with the previous experiment, the modified application records Action Events to a database. As this application was not run in a laboratory environment, it was necessary to provide a convenient way for users to play the game and enable the recording of the User Interaction Streams generated whilst playing the game.

To accomplish this goal, the application was packaged and delivered using Java Webstart*Java Web Start Technology* (2008). The database was configured to allow remote connections. When the application was launched, it connected to the database remotely. This worked well, although it did limit the location that users could play the game in, as some institutions had firewalls that prohibited outgoing connections for anything other than Web based traffic. These users could download and start the game, but could not play as their events could not be recorded.

A Web site was created in order to explain the purpose of playing the game. It was also used to distribute the game. To start, a user simply had to click on a link. In addition, some technical details were available for users interested in the experiment.

In addition to being used to distribute the application, the Web site also recorded and displayed the highest scores. By seeing the score, this encouraged users to compete for the highest score, ensuring full usage for recording purposes.

8.6.1 Lab Script

There was no script for users to follow when playing the game. Users were simply asked to play the game as often as they liked.

8.6.2 Experimental Observations

Initially, some users commented that the application felt very jerky. The section below outlines the changes that were made to remove this problem. No other comments were made by users.

Changes to the Interaction Capture System

The only change that needed to be made to the interaction capture system was to move the logging of captured events to a separate thread. Users were encouraged to play the game on their own computers. As a result of this, there was significant latency when recording generated events to the database.

In the initial implementation of the capture system, the recording was done on the Swing thread as this was simpler. When carrying out the first experiment, events were recorded over a local area network, so there was no perceivable effect of recording the events. To alleviate the effect of the much bigger latencies between the users playing Asteroids and the database logging the recorded Action Events, the recording and logging of events were split into two separate threads. When an event was generated, this happened on the SwingThread. The recorded Action Event was then pushed onto a queue. When the application was started, a second thread was started. This thread removed Action Events from the queue and logged them to the remote database. The use of a queue ensured that the order of action events was preserved.

8.6.3 Application Modification

The automated tool described in section 6.4.2 was used to find and assign Target Symbols to the user interface components that made up the application.

8.7 Summary of the BUIS Corpus

The BUIS corpus consists of the results of the two experiments outlined above. The users are anonymous. The corpus is available to download from *Bangor User Interaction Stream Corpus* (2008). The corpus contains two files:

WEKA.sql The User Interaction Streams captured by the sixteen users using WEKA.

Asteroids.sql The User Interaction Streams captured by the ten users playing the Asteroids game.

Each file is a stand alone database dump for the postgresql database. The dumps consist of three tables:-

sessions This table has three columns that contain First Name (`f_name`), Last Name (`s_name`) and a `sessionID`. The names have been made anonymous.

tasks This table has four columns. The first, `id` references a `sessionID` in the session table. The second and third are the task name (`taskname`) and task details (`taskdetails`). The final column, `taskid` is an identifier.

events The Action Symbol is recorded in two columns, `event` and `value`. The `event` column is used to store the Symbols type and the `value` column is used to store the value, for example the mouse button number. The target symbol is recorded in the column named `object`, The date and time of the event is recorded in the column labeled `eventtime`, The column `taskid` references the `taskid` in the tasks table.

This corpus is analysed in the next chapter.

Chapter 9

Experimental Results and Analysis of the BUIS Corpus

This chapter will analyse the BUIS Corpus that was created in chapter 8. The chapter will be split into two distinct sections. Section One will analyse the results of data collected using WEKA. Section two will analyse the results collected using Asteroids.

Each section will detail the basic statistics of the collected data. The similarities between a natural language, both in relation to letters and words, will be explored. Finally the categorisation performance at the task of authorship will be evaluated.

9.1 BUIS WEKA Analysis

This section will examine the data collected for Part One of the BUIS corpus. This part of the Corpus contains User Interaction Streams for sixteen users using the WEKA machine learning application following a lab script.

9.1.1 Statistical Analysis

Analysis of the raw collected data shows that of the sixteen users who participated in the experiment, all generated a different number of action events. Table 9.1 shows the number of events generated against the person.

Table 9.1: User against Events Generated

Person	Number Of Action Events
N	28794
D	12919
C	11478
L	6660
J	6361
G	6096
E	6004
B	5367
H	4343
K	3625
O	2724
A	2401
M	2201
F	2140
P	1749
I	1425

As can be seen from table 9.1, there is a wide variety in the number of events. The three users with the largest number of events — *N*, *D* and *C* — also had the largest number of tasks. One user, *N*, produced twice as many events as the next highest user, *D*. Overall, the average number of events per user was 6518, with a standard deviation of 6805. The standard deviation is very large. However, it was shown that the figure was skewed by the presence of 3 very high counts. By creating a sub set of the data to ignore the highest three users (*C*,*D* and *N*) the average drops to 3930 and the standard deviation falls to 1952.

Target Symbols

In total, 545 user interface components of WEKA were assigned a Target Symbol. The sixteen users interacted with a total of 236 unique Target Symbols. From this, it can be seen that users interacted with less than half the components present in the

WEKA interface. This was to be expected, as the tasks they were given do not cover all of the functionality available in WEKA.

Table 9.2 shows the number of unique target symbols against each user.

Table 9.2: User against Unique Target Symbols

Person	Number Unique Interactions	Target Symbols Unique to User
N	188	4
E	169	5
L	154	0
B	151	5
D	151	0
M	144	1
C	143	3
G	139	0
K	137	1
I	135	2
O	135	0
F	110	0
J	107	0
A	99	7
P	99	0
H	84	0

As can be seen from the table 9.2, the same user, N, generated both the largest number of events and interacted with the largest number of Target Symbols. Despite interacting with more Target Symbols than any other user, N only interacted with 188 of the 236 Target Symbols that were interacted with by all other user. From this, it is clear that some users have interacted with different parts of the WEKA interface than other users.

The Table 9.2 also shows the number of Target Symbols that were unique to that user. From this, it can be seen that user A, despite not generating many events and only interacting with 99 unique Target Symbols, interacted with the most Target Symbols not used by other users.

Action Symbols

It was not possible to determine the total potential number of unique Action Symbols, as this was dependant on the hardware used. For example, the number of keys on the keyboard may vary, as could the number of buttons on a mouse. The users using the modified version of WEKA generated 34 unique Action Symbols. Table 9.3 shows the number of Action Symbols generated by each user.

Table 9.3: User against Unique Action Symbols

Person	Number Unique Interactions Types	Action Symbols Unique To User
F	30	12
E	18	0
D	18	0
N	17	0
C	13	1
L	13	0
J	12	0
M	10	0
G	10	0
K	10	0
A	9	0
B	9	0
H	9	0
I	8	0
O	8	0
P	7	0

From Table 9.3 it can be seen that as well as there being less unique Action Symbols than Target Symbols in total, apart from user *F*, the distribution of Action Symbols was more homogeneous than the distribution of Target Symbols across all users. Only two of the sixteen users generated Action Symbols that were unique to them.

Action Events

This section examines the statistics relating to the number of unique Action Events captured against the user. (As described in section 5.4.4, an Action Event is the combination of a Target Symbol and an Action Symbol). Table 9.4 shows the number of unique Action Events generated by each user. It also shows the number of Action Events that were unique to that user, i.e. the number of Action Events that the user generated that no other user created.

Table 9.4: User against Unique Action Event

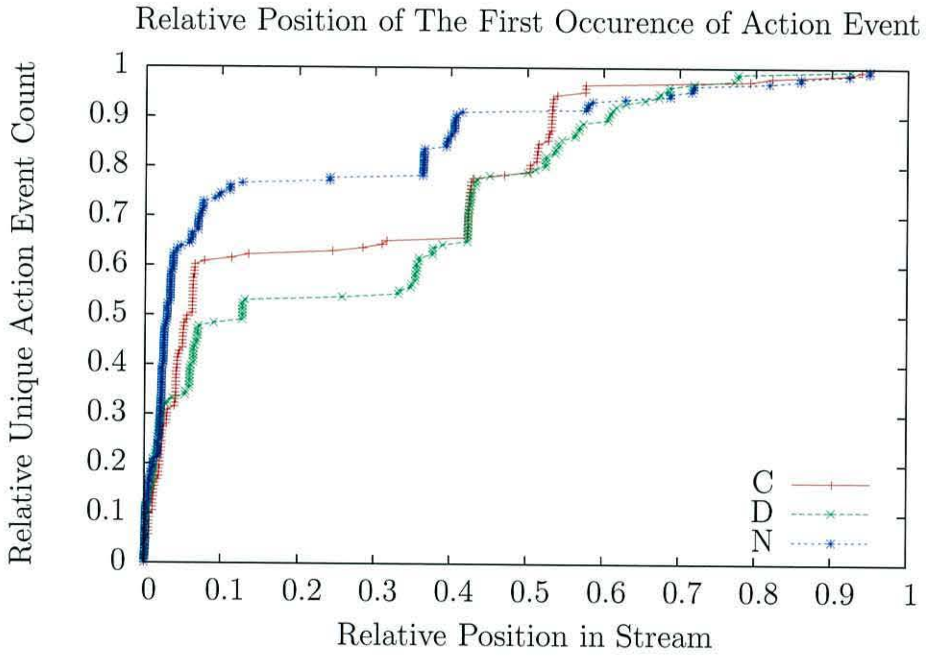
Person	Number Unique Interactions Types	Action Events Unique To User
N	712	88
D	537	34
E	469	23
L	424	8
C	419	21
M	409	23
B	375	14
G	358	9
J	333	7
K	331	6
F	330	25
O	310	0
I	302	10
A	278	29
P	242	3
H	216	3

From the table, it can be seen that the number of users with a unique Action Events (17) is much higher than the number with either unique Target Symbols (8) or with unique Action Symbols (2). Furthermore, it can be seen from the table that even though each of the users was interacting with the same graphical components, the method of interaction was different for different users.

Uniqueness Across a User Interaction Stream

So far in this chapter, the number of unique Target Symbols, Action Symbols and Action Events have been analysed and discussed. It has been shown that there are a large number of both Target Symbols and Action Events that are unique to an individual user. The following graph, figure 9.1, shows the relative position of the first unique Action Event for the top three Users.

Figure 9.1: Relative Position for Top Three Users



As can be seen from this graph, initially there was a large number of unique events. The first ten percent of the file contains between fifty and seventy percent of the unique actions. User *N* had more events than any other user, but has generated a larger percentage of the unique action events in the first ten percent of his or her User Interaction Stream than any of the other users. There is a common pattern of a large number (nearly 50 percent) of unique Action Events generated in the first 10 percent of the stream. After this there is then a steady, mostly flat section with the remaining rises as a series of small jumps.

Tasks

When following the lab scripts, the users were asked to record when they had completed a task. There was no way of enforcing this. As a result, many users simply forgot. Table 9.5 shows user against number of recorded tasks.

As can be seen, there is a large variation in the number of recorded tasks. The top three users N, D, and C were also the users who recorded the most tasks.

Table 9.5: User against Number of Recorded Tasks

User	Number Of Tasks
A	4
B	4
C	14
D	13
E	8
F	1
G	8
H	7
I	2
J	11
K	1
L	8
M	1
N	12
O	2
P	2

9.1.2 Natural Language Similarities

In this section, User Interaction Streams will be compared and contrasted with the components of a Natural Language. The comparison will be discussed in two stages. First, the similarities of Target Symbols and Action Symbols to letters will be evaluated. This will be followed by a comparison of the similarities between Action Events and words. For both of these comparisons, the LOB corpus Stig Johansson (1978) described in chapter 3.7.3, will be used as an example of English.

Letters

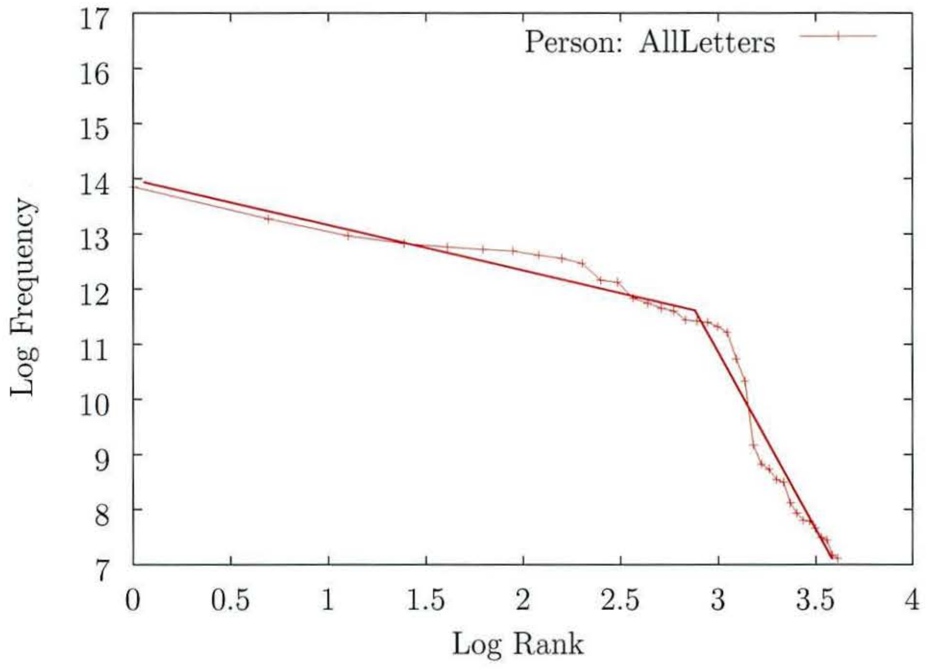
In this section, the similarities between Target Symbols and Action Symbols to letters in a natural language is examined. It will be shown that the 'letters' produced by the interaction with an application show similar characteristics to the letters of a natural language, such as English.

Zipf's law Zipf (1968) describes the letter frequency distribution of English. This states that:-

The log frequency of a letter is inversely proportional to the log of its position in the rank table.

This can be understood more clearly when seen plotted as a graph, showing Log Frequency against Log Letter Rank for the letters of the LOB corpus. See figure 9.2.

Figure 9.2: Log Frequency against Log Letter Rank for LOB corpus.



As can be seen the plot is made up of two straight lines.

Target Symbol

Figure 9.3 shows log frequency plotted against log Target Symbol rank for each user.

The relationship is clearly linear. This fits well Zipfs law.

Figure 9.3: Log Frequency vs Log Target Symbol Rank for each user.

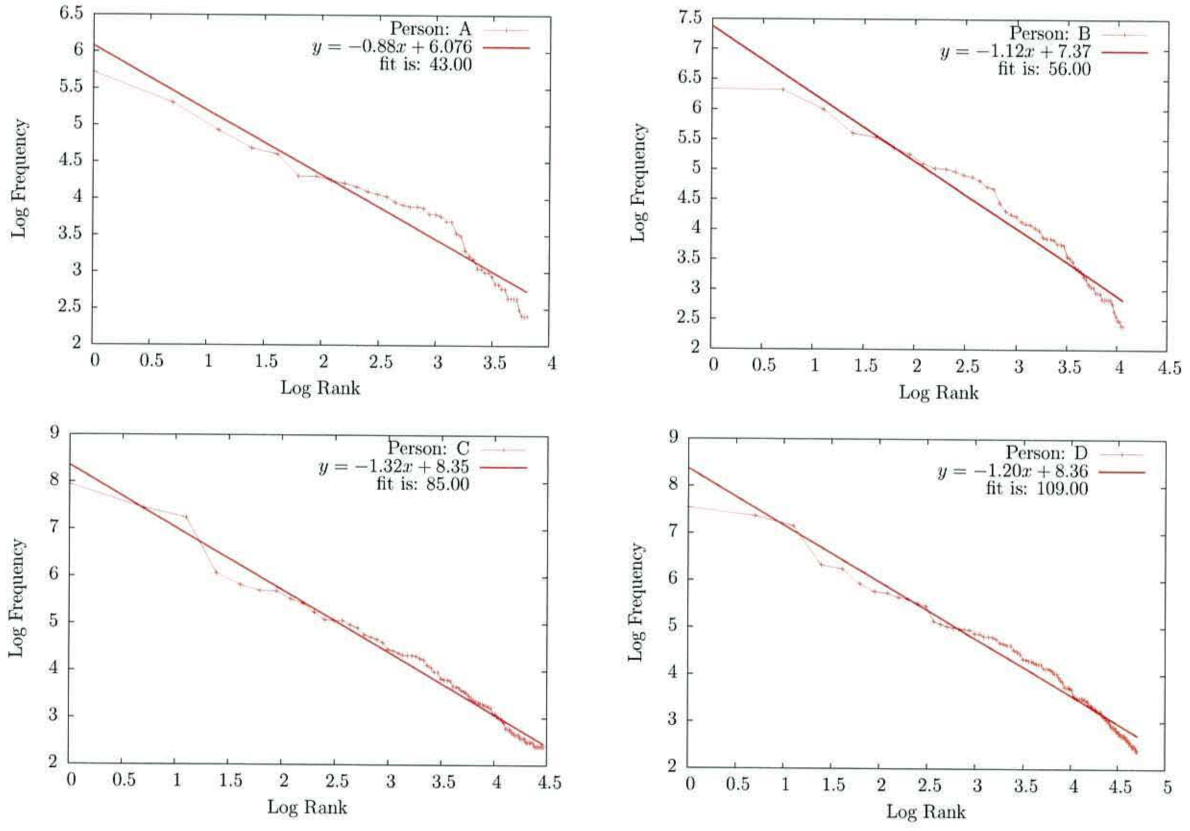


Figure 9.3 (continued)

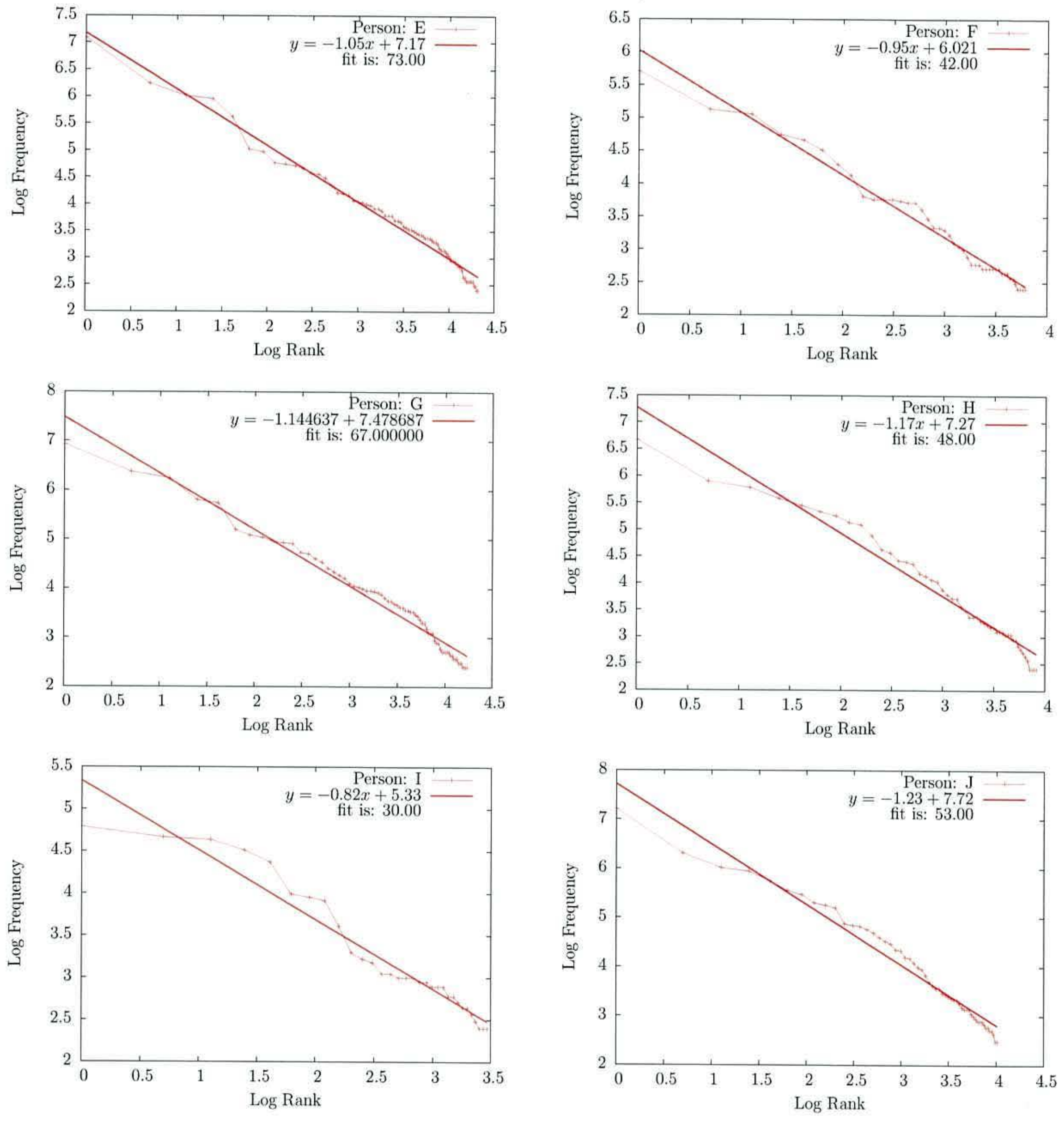
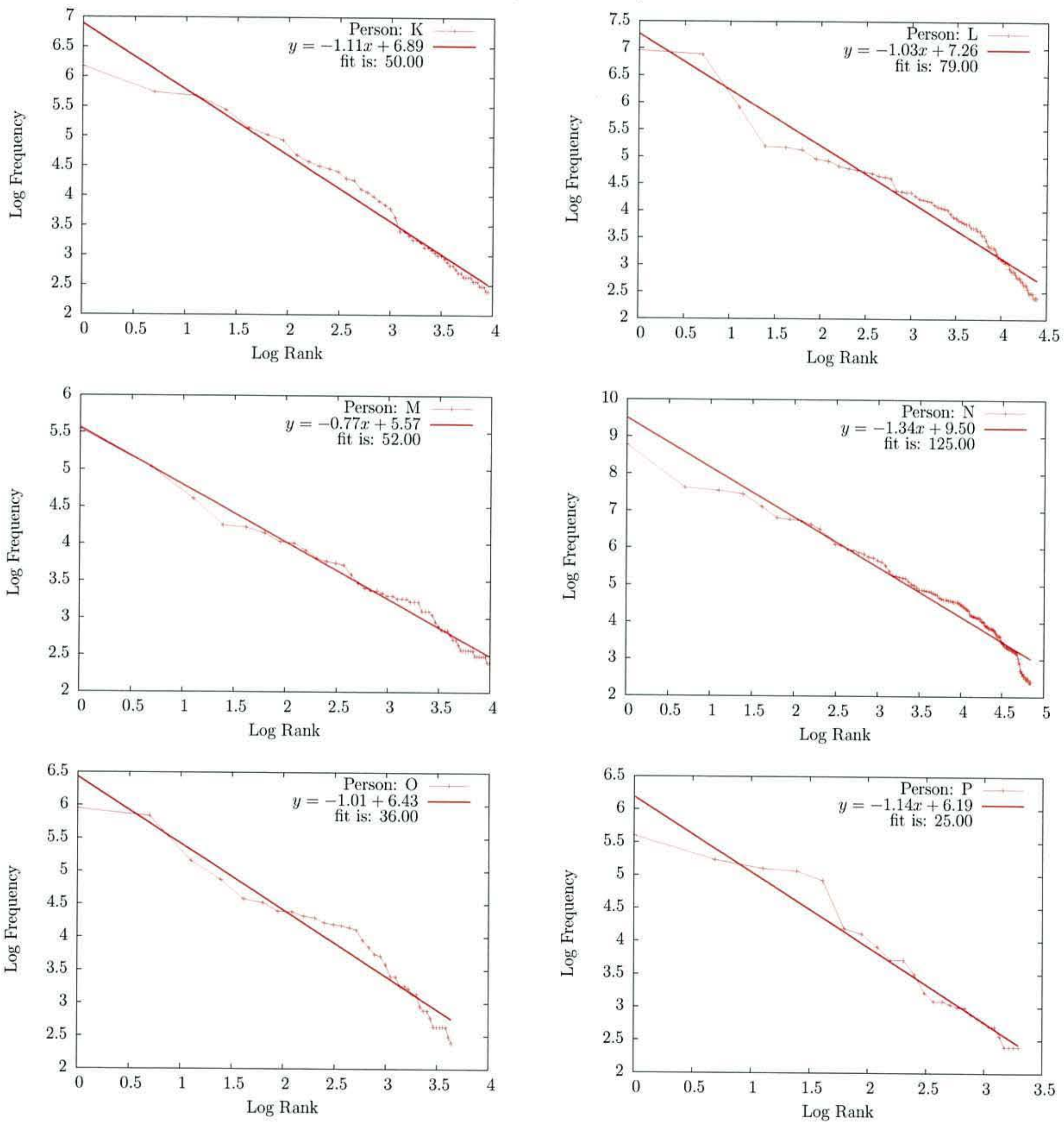


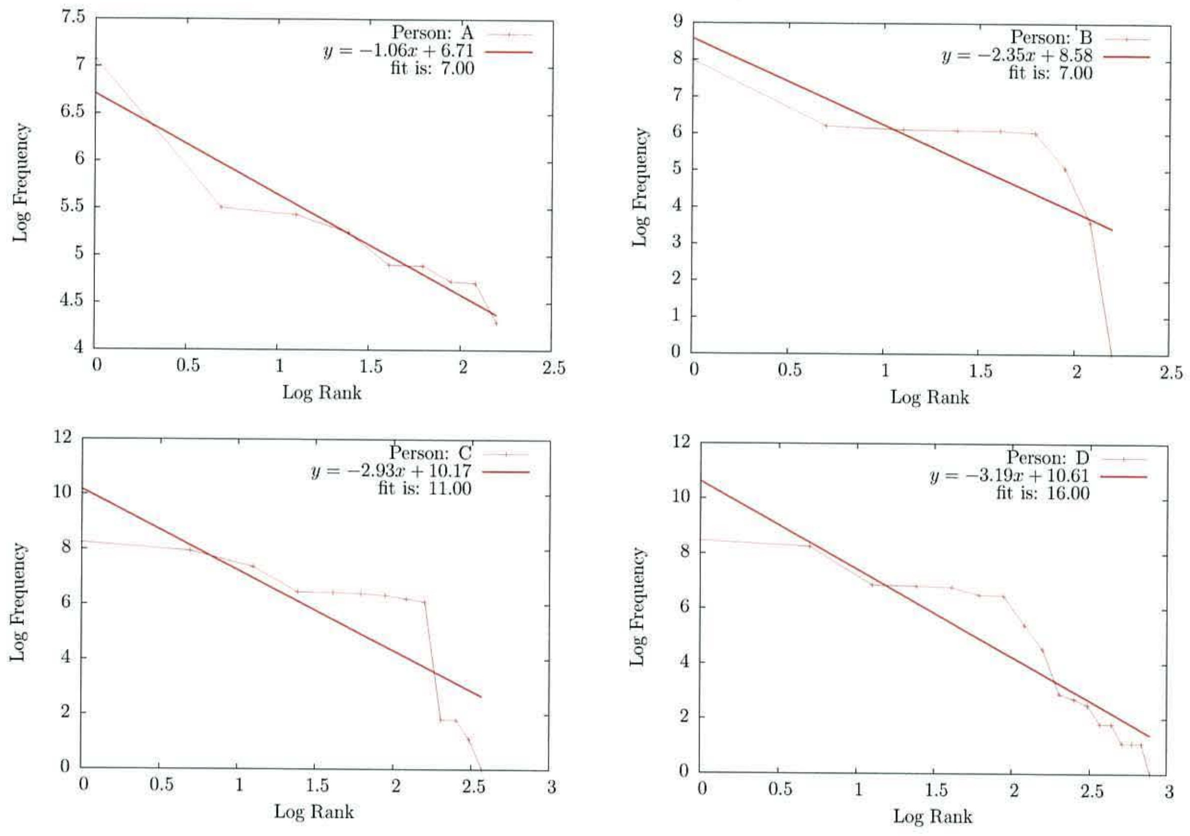
Figure 9.3 (continued)



Action Symbol

Figure 9.4 shows log frequency plotted against log Action Symbol, ranked for each user.

Figure 9.4: Log Frequency vs Log Action Symbol Rank for each user.



There appears to be two distinct patterns. An almost linear relationship, corresponding to Zipfs law for users: A,D,F,H,I,N,O and P, and a very different pattern for users: B,C,E,G,J,K,L and M.

Figure 9.4 (continued)

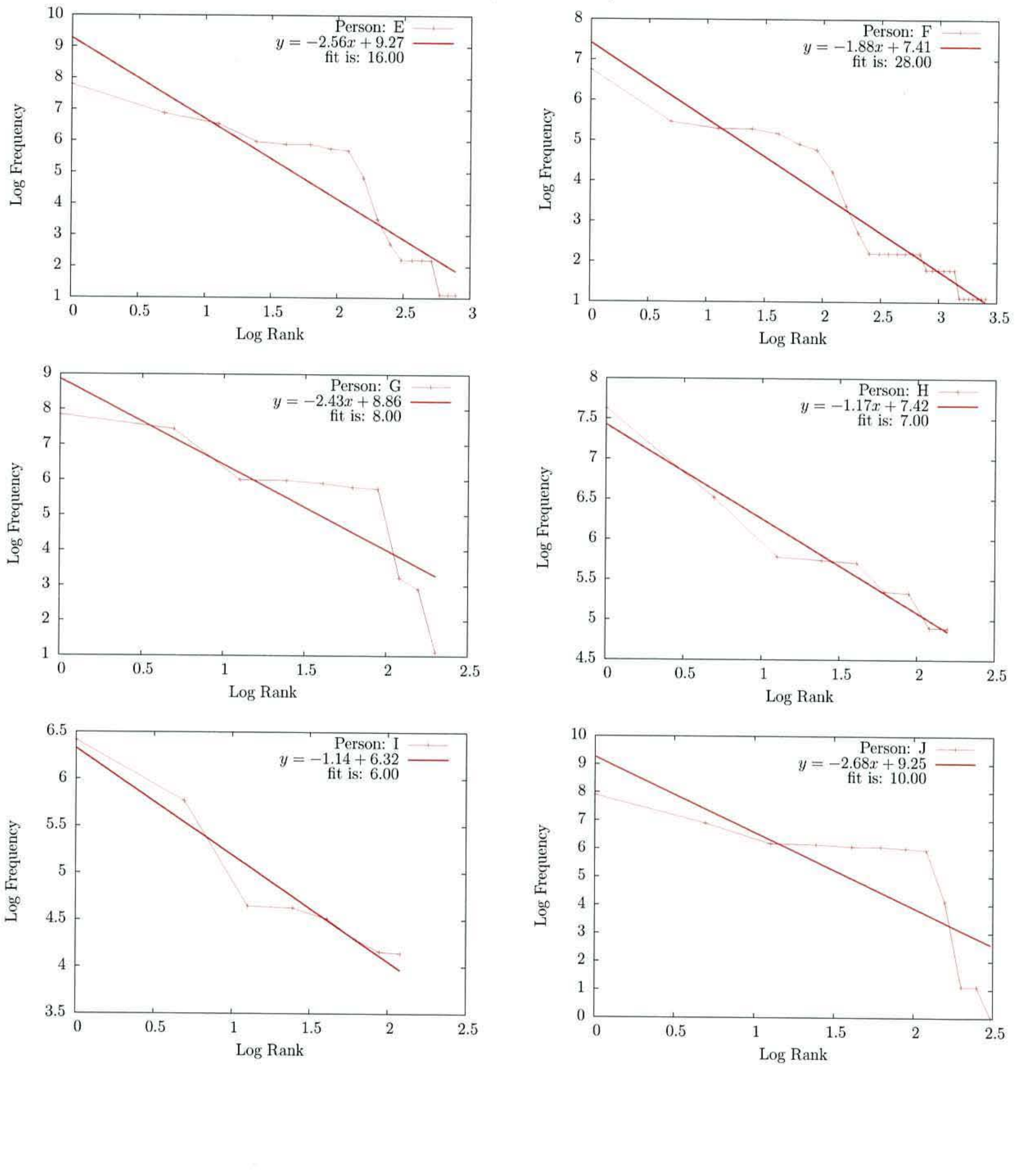
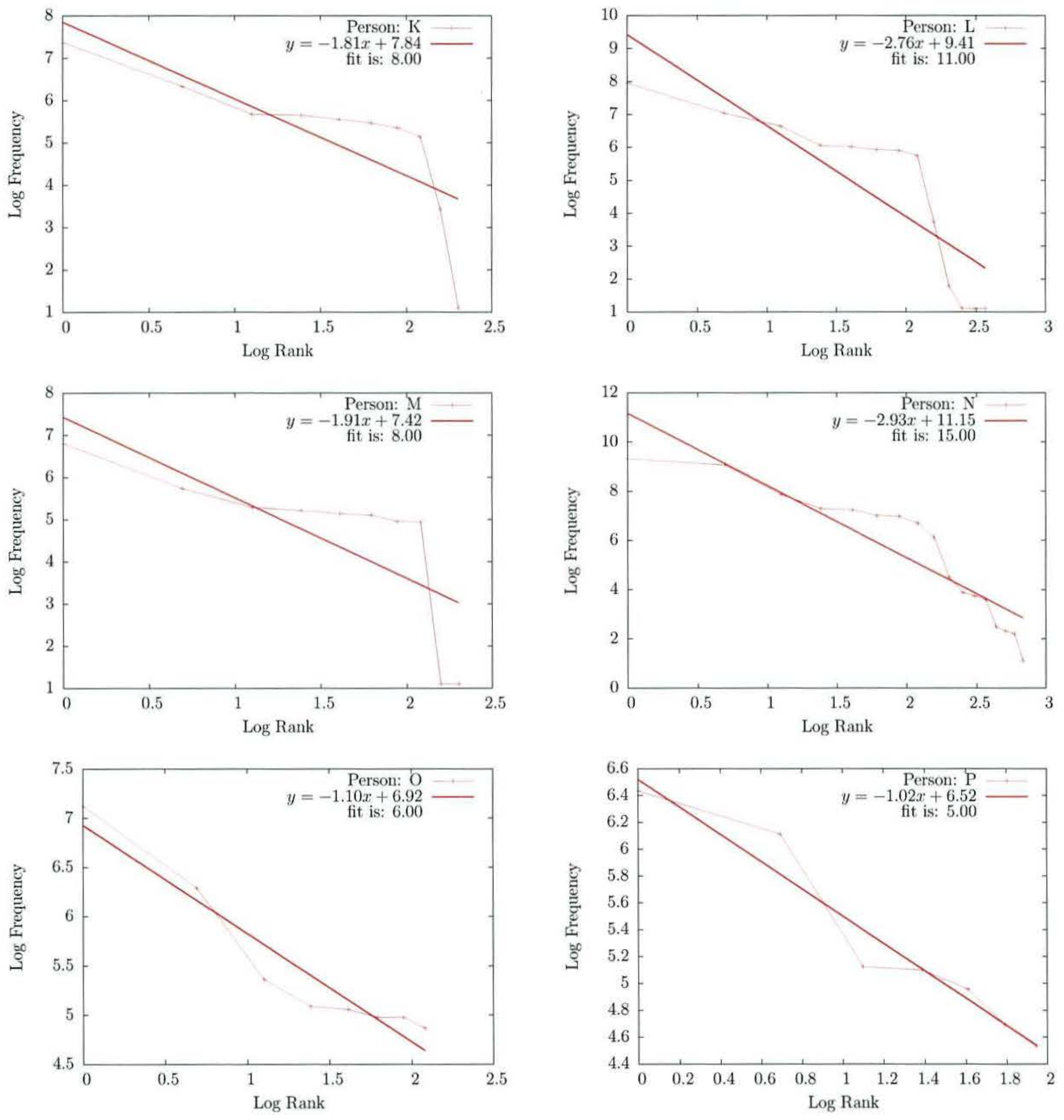


Figure 9.4 (continued)



Words

As with letters, the relationship between log frequency and log rank for words also closely follows Zipfs law (see Li (1992)). Figure 9.5 shows the log frequency against log rank graph for the LOB corpus.

Figure 9.5: Log Frequency against Log Word Rank for LOB corpus.

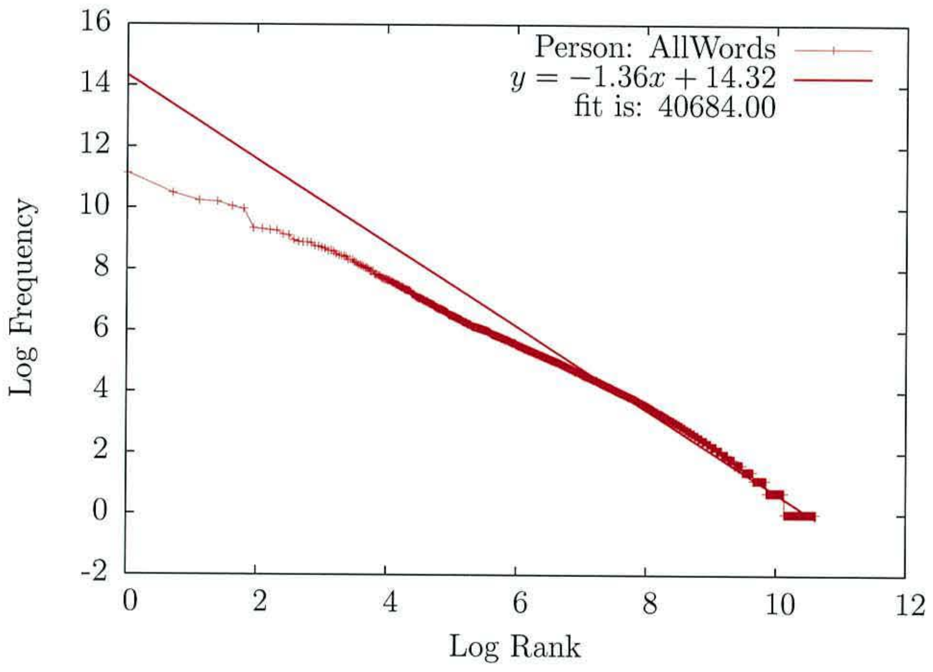


Figure 9.6 shows log frequency plotted against log rank Action Event for each user. This is equivalent to plotting the log frequency against log word rank. Section 5.5 describes the language like properties of a User Interaction Stream.

Figure 9.6: Log Frequency vs Log Action Event Rank for each user.

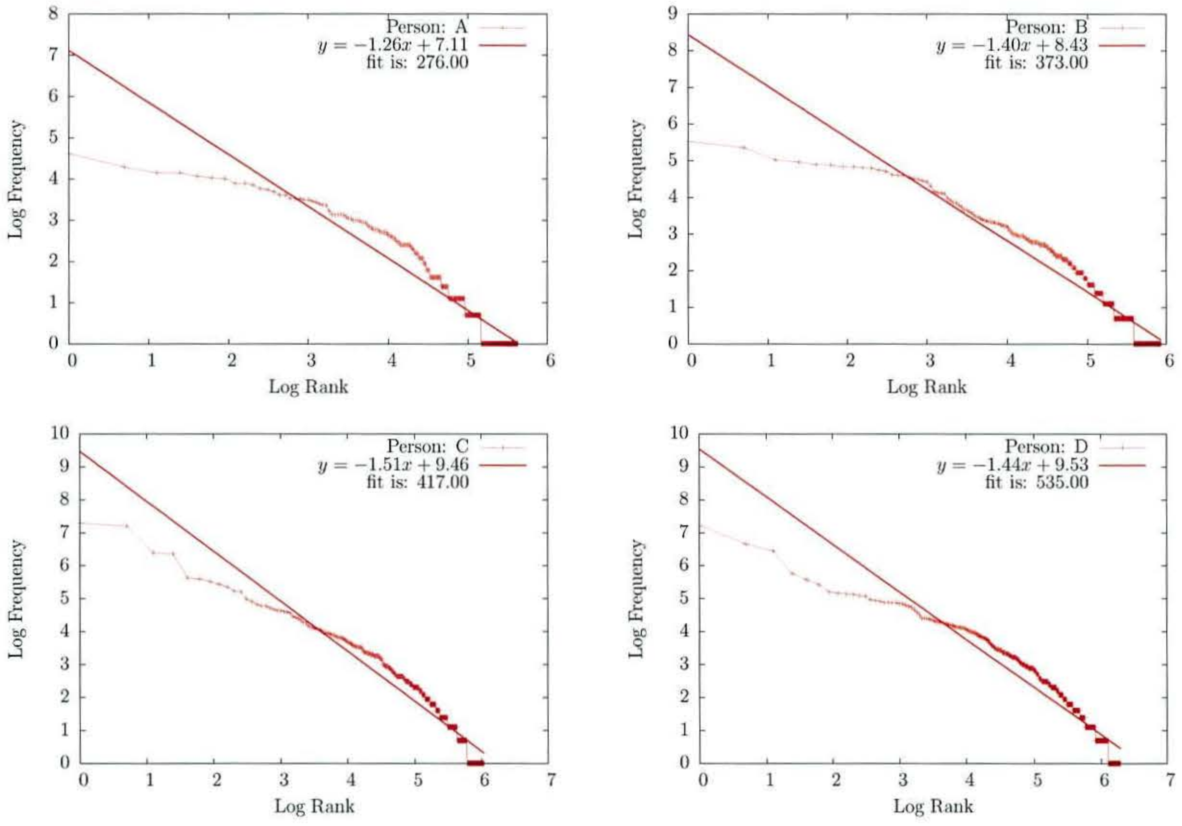


Figure 9.6 (continued)

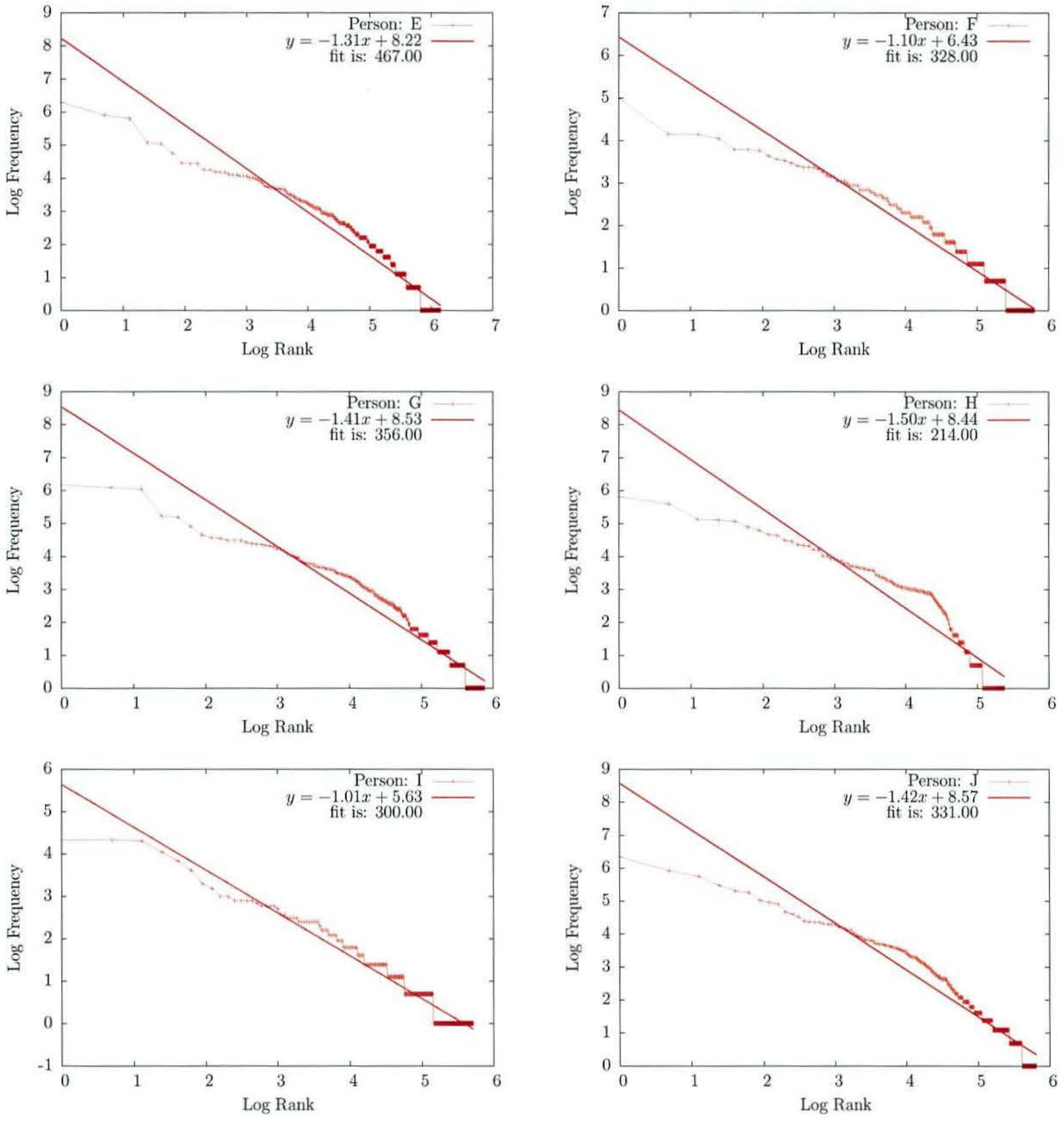
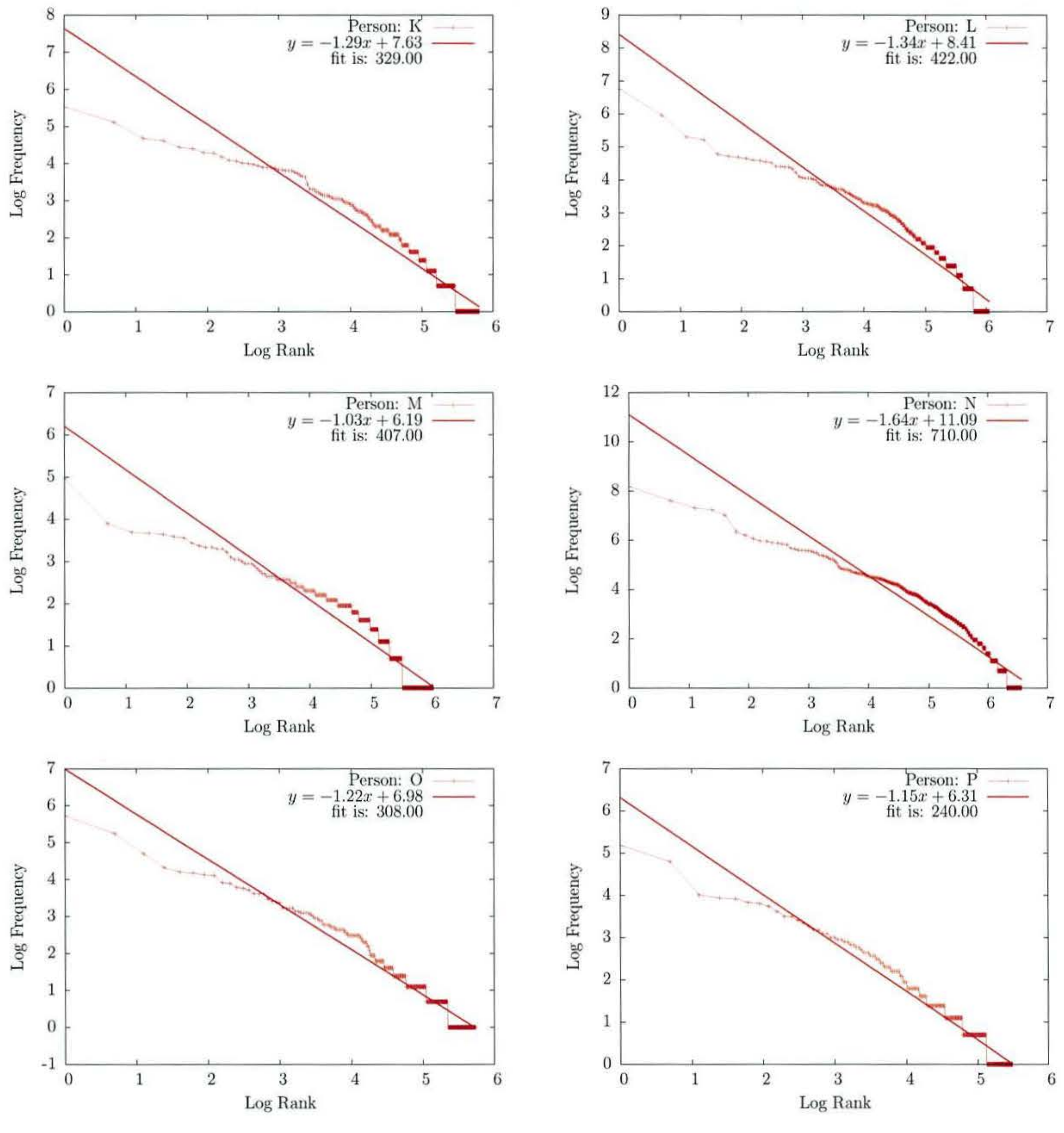


Figure 9.6 (continued)



9.1.3 Categorisation

This section will examine the performance of categorisation at the authorship task.

Categorisation By Task

As described in section 8.5.1, users were asked to note when they had completed a task in the laboratory exercise. The result of this was that each user produced a number of unique documents. This section will look at performing the categorisation of data generated by users, treating each task as a separate document. Although the users were asked to note when they had completed a task, many forgot — an excellent example of the completion problem Karwowski (2006).

When performing this categorisation, a training set was produced. The training set consisted of all but one of the User Interaction Streams. The excluded User Interaction Stream was used as the document. A different User Interaction Stream was then selected and the process repeated until each User Interaction Stream had been used for testing.

Categorisation of All Users

This section will look at the performance of categorisation for every user. As shown in chapter 4, the optimal maximum context length for compression is not always the optimal model size for categorisation performance. Section 10.8.1 will show that the optimum model size for compression is order 4. Although not guaranteed to be optimal, this was a good size to start investigating the categorisation performance. Section 3.8 described what should be examined when considering categorisation performance: overall accuracy, recall and precision.

Table 9.6 shows the recall and precision for all users at order 4. The overall accuracy was 23.58 percent. As can be seen from the table, only six of the sixteen users had documents correctly attributed to them. The relatively high overall accuracy was caused by the fact that the users who had documents attributed correctly to them, for example *D* and *N*, produced more documents. It is interesting to note that user C generated the most tasks and a large number of events but did not have any documents correctly attributed.

Table 9.6: Categorisation Performance for All Users using PPM Order Size 4.

User	Number Of Events	Recall	Precision
D	12919	0.385	0.294
O	2724	0.0	0.0
K	3625	0.0	0.0
H	4343	0.286	0.667
C	11478	0.0	0.0
P	1749	0.0	0
B	5367	0.0	0.0
L	6660	0.286	0.4
G	6096	0.143	1.0
I	1425	0.0	0.0
M	2201	0.0	0.0
A	2401	0.0	0
F	2140	0.0	0.0
J	6361	0.333	0.25
N	28794	0.667	0.286
E	6004	0.0	0.0

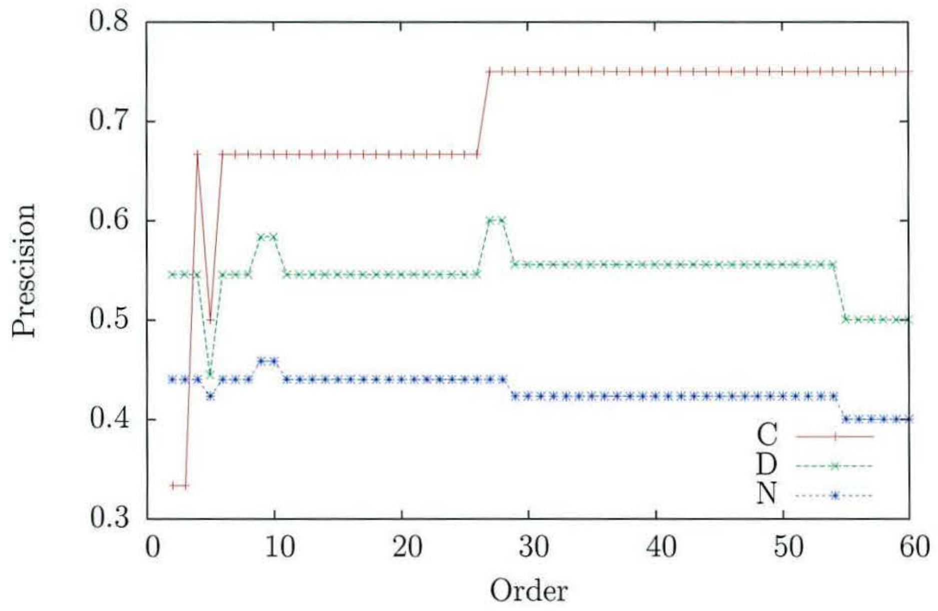
Categorisation of Top Users

As described earlier, three of the users (N, D and C) produced far more events than the other users. It has already been shown, Teahan & Harper (2001), that for successful language identification a minimum of approximately 2,500 characters is needed to achieve 95 percent accuracy at the language identification task. The task of authorship attribution requires more data.

This section will examine the categorisation performance of the 3 users with the largest number of Action Events. As with the section above (section 9.1.3) each *recorded* task will be considered as a unique document. These users also have the largest number of tasks. As there are only three users in this set, it is possible to graph the categorisation performance for each user against model size. Figure 9.7 shows the precision performance against model size.

As can be seen from the table, the optimum model size is twenty seven or twenty

Figure 9.7: Precision against Order for Top Three Users



eight. Both these order sizes have the same precision and recall. This is shown in table 9.7. There is another smaller peak at order nine and ten, although this peak is smaller.

Table 9.7: Recall and Precision for Three Users at Order 4.

User	Recall	Precision
D	0.462	0.6
C	0.214	0.75
N	0.917	0.44

Even though these results are much better than for all users, the performance of the categorisation of user *C* is still disappointing. User *C* categorisation performance is much worse than the other two users. In the previous section, user *C* was identified as one of the users with no streams correctly assigned at order 4. As has already been stated, a minimum of 2,500 characters are needed to perform language identification.

Table 9.8 shows the number of Action Events in each User Interaction Stream used for categorisation against user for the user *N*,*D* and *C*. As can be seen from this table, many of the tasks do not meet the required number of symbols needed to perform language identification. As a result of this there is a lack of training data which is affecting the performance.

Table 9.8: User and Task against Number of Action Events

User	Task ID	Number Of Action Events
C	10	290
C	21	510
C	29	1972
C	43	365
C	61	1618
C	63	30
C	65	312
C	66	775
C	67	405
C	76	944
C	78	466
C	79	329
C	84	2290
C	94	1172
D	25	645
D	32	4068
D	48	11
D	49	1155
D	54	546
D	6	303
D	7	1733
D	74	1777
D	83	386
D	86	619
D	90	614
D	92	429
D	95	633
N	102	1773
N	103	1576
N	60	2211
N	62	1090
N	64	4560
N	68	534
N	69	7037
N	70	1028
N	71	1385
N	72	1505
N	73	744
N	75	5351

Cross Validation Approach

As has been shown in the section above, using the tasks to split the User Interaction Streams into documents had some problems associated with it. In this section, the problem of different document size will be addressed, by concatenating all of the User Interaction Streams for a user to form one document. This single document will then be split into ten subsection to perform tenfold cross validation.

Chapter 4 identified a problem whereby, when concatenating documents, *new* contexts can be introduced that are not present in the original documents. In the case of the User Interaction Streams captured to form the corpus, this was not a problem. The reason for this was that, although the documents were spilt by task, the user did not stop between the tasks. So, provided the concatenation was done in the correct order, no *new* contexts were introduced. Instead, the contexts that were omitted were reintroduced. The omitted contexts are those that occur at the end of one task and the start of the next task.

The top three users were analysed. The results are shown in figure 9.8 below:

These results are much improved, but the performance is still considerably less than when performing the authorship task on natural language. It is possible that user N is still swamping the other users. Another possibility is that the nature of the streams is effecting the results. As shown in section 9.1.1, the first occurrence of unique Action Events did not just occur at the start of the User Interaction Stream. It was distributed all the way across the stream. The standard cross validation approach used above suffered as a result. An alternative method of performing cross validation is described in the appendix B. When this mechanism was used there was a dramatic increase in performance, as shown in the PPM results table 9.9. The performance was the same at all orders. This can be seen in figure 9.9. This gave an overall accuracy rate of 83.3 percent.

The same data set was also analysed using SVM, C Measure and R Measure. The SVM implementation was provided by WEKA. The results of performing categorisation using SVM are shown in the table 9.9. The overall Accuracy is 60 percent.

Figure 9.8: Precision against Order for Top Three Users (PPM).

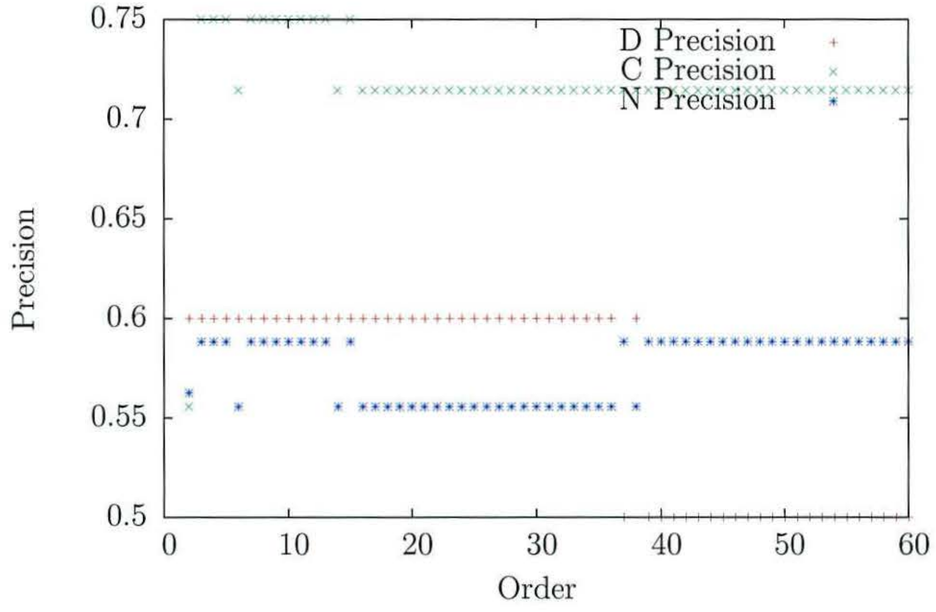


Figure 9.9: Precision against Order for Top Three Users (PPM).

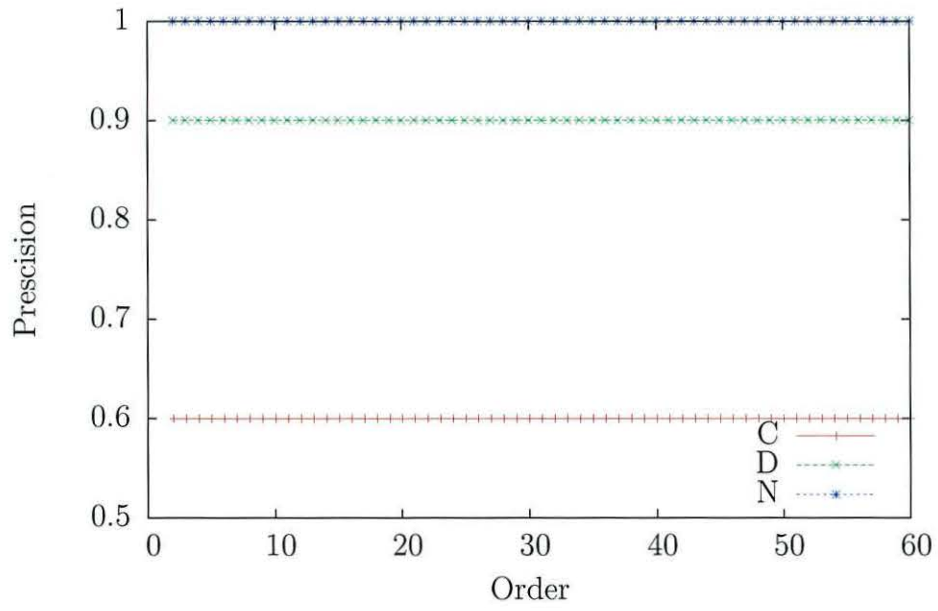


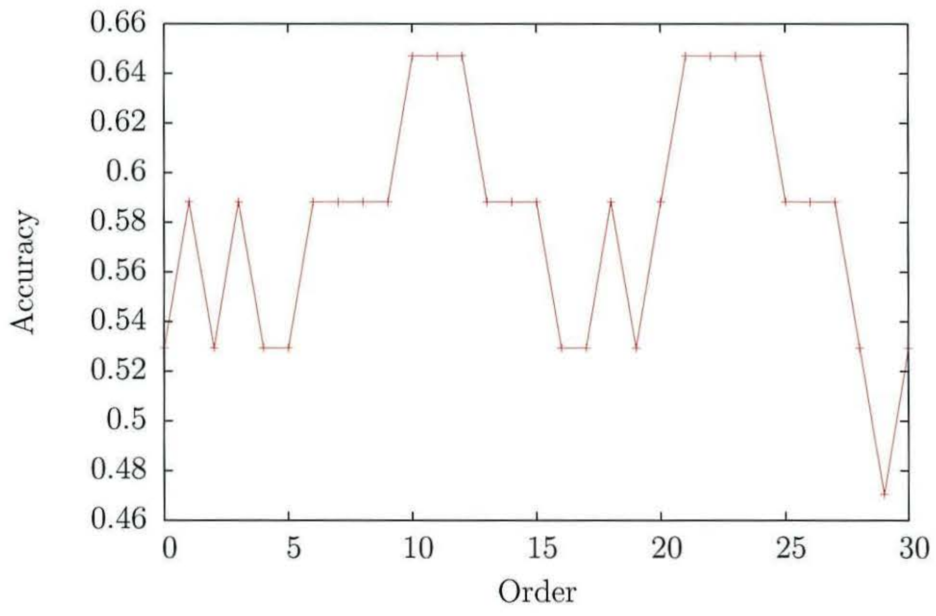
Table 9.9 also shows the recall and precision performance using PPM, SVM, R Measure and C Measure. The overall accuracy is for the R Measure 53 percent.

Table 9.9: Categorisation Performance for Three Users.

	PPM		SVM		R Measure		C Measure	
User	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision
D	0.9	0.9	0.7	0.636	0.75	0.6	1.0	0.8
C	0.6	1.0	1.0	0.556	0.0	0.0	0.0	0.0
N	1.0	0.714	0.1	1.0	0.75	0.3	0.75	0.3

Figure 9.10 shows the performance of C Measure with the same data set. As can be seen the C Measure performs better than the R Measure but not as well as PPM. The maximum accuracy achived by the C Measure is 65 percent. Both the C Measure and R Measure have 0 precision and 0 recall indicating that neither of these algorithms attributed any of the streams to author C.

Figure 9.10: Precision against Order for Top Three Users (C Measure).



Three Users

The users for this section of the BUIS corpus were all students undertaking a course as part of their studies. It is clear that there is a significant difference between the top three users and the other users as shown by the number of Action Events produced by the users. As the users were anonymous it was not possible to correlate between final marks for this module and the number of Action Events generated. The logging version of WEKA was only available in the Computer Science Laboratory, students who started in the laboratory and completed the assignment using their own computers would have only generated action event streams for the part of the work undertaken in the laboratory. It is hypothesised that the top three students were highly motivated and so completed all parts of the assignments successfully in the computer science laboratory.

Strengths

As has been shown, the categorisation performance for the top three users is 83 percent accurate when using cross validation. The categorisation performance has been analysed with a number of different algorithms:- PPM, C Measure, SVM and R Measure. PPM was shown to be the most effective.

Weaknesses

Of the sixteen users who participated in the experiment, only three produced enough Action Events for accurate categorisation to take place.

9.2 BUIS Asteroids Analysis

This section will examine the data collected for Part Two of the BUIS Corpus. This part of the Corpus contains User Interaction Streams. Section 8.6 describes the collection of this data.

9.2.1 Statistical Analysis

This section examines the raw data collected to create the Asteroids section of the BUIS corpus. There were ten users of asteroids who generated a total of 68148 events. On average, each user generated 6195.27 Action Events. The standard deviation is 6475.04. Table 9.10 shows a complete breakdown of user against number of Action Events.

Table 9.10: User against Events Generated

Person	Number Of Action Events
F	18140
A	16732
C	11351
E	7448
I	5634
J	2548
D	1914
B	1660
G	1531
H	1190

Target Symbols

As the Asteroids game has a much simple interface than WEKA, only 16 unique Target Symbols were generated. The ten users interacted with all 16 Target Symbols. However, not all users interacted with each target symbol. Table 9.11 shows the number of unique target symbols against each user.

As can be seen from the table, four of the users only interacted with one target symbol. These users operated and played only using the keyboard. None of these users interacted with parts of the application that other users did not.

Table 9.11: User against Unique Target Symbols

Person	Number of Unique Interactions	Target Symbols Unique To User
D	16	0
H	12	0
E	9	0
F	8	0
A	4	0
I	3	0
B	1	0
C	1	0
G	1	0
J	1	0

Action Symbols

The users generated 38 unique Action symbols. Table 9.12 shows a breakdown of user against the number of unique Action Symbols and the number of these symbols that were unique to that user.

Table 9.12: User against Unique Action Symbol

Person	Number Unique Interactions Types	Action Events Unique To User
A	22	2
B	13	0
C	24	5
D	15	0
E	23	0
F	20	1
G	14	0
H	18	0
I	16	0
J	16	1

Action Events

Table 9.13 shows the break down of user against number of unique Action Events. The users participating in the Asteroids experiment have a much smaller alphabet and vocabulary than the WEKA users.

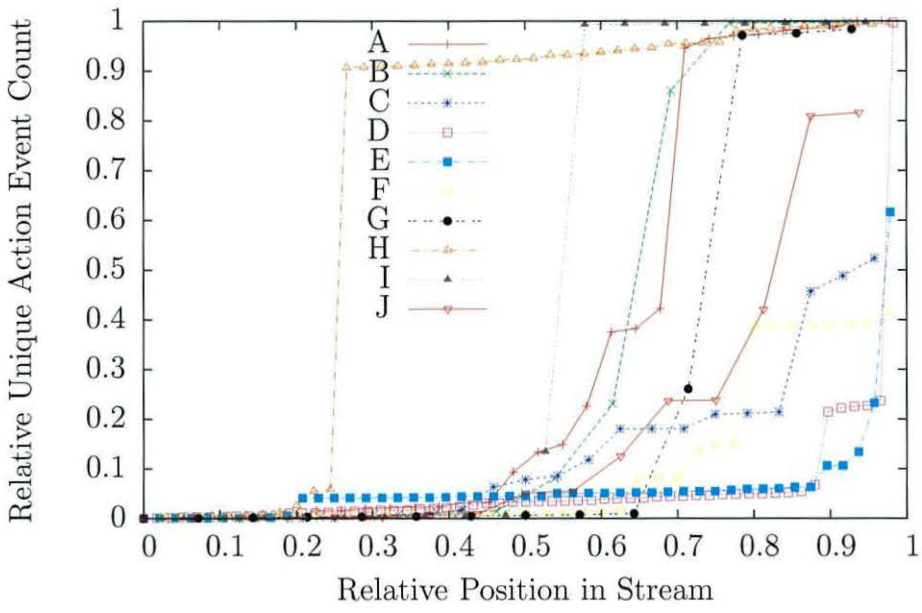
Table 9.13: User against Unique Action Event

Person	Number Unique Interactions Types	Action Events User Unique To
F	40	2
A	31	6
K	30	1
C	24	5
E	48	1
I	19	0
J	16	2
D	58	7
B	13	0
G	14	0
H	45	7

Uniqueness Across a User Interaction Stream

Figure 9.11 shows the relative position of the first Action Event within each User Interaction Stream.

Figure 9.11: Relative Position of The First Occurrence of Action Event.



Comparing this to the equivalent figure in the WEKA section of the analysis (figure 9.1) we can see that users generated the majority of their interactions at the start of the stream.

9.2.2 Similarities to a Natural Language

This section will examine the similarities of the User Interaction Streams and a natural language.

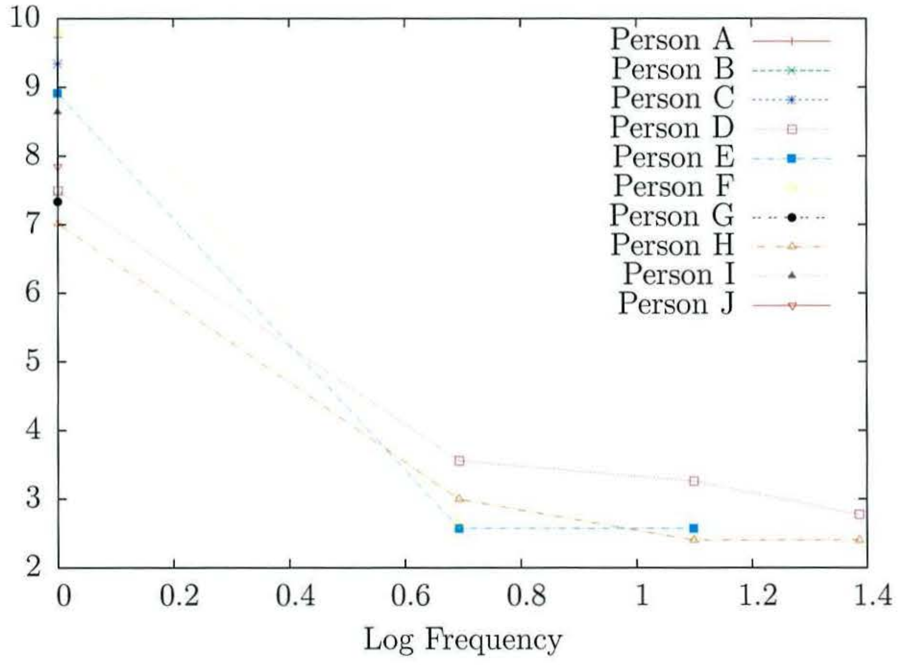
Letters

In this section the similarities between Target Symbols and Action Symbols to letters in a natural language is examined. As with section 9.1.2 it is shown that the data conforms to Zipf's law.

Target Symbol

Figure 9.12 shows log frequency plotted against log Target Symbol rank for each user.

Figure 9.12: Log Frequency against Log Word Rank for Target Symbols.

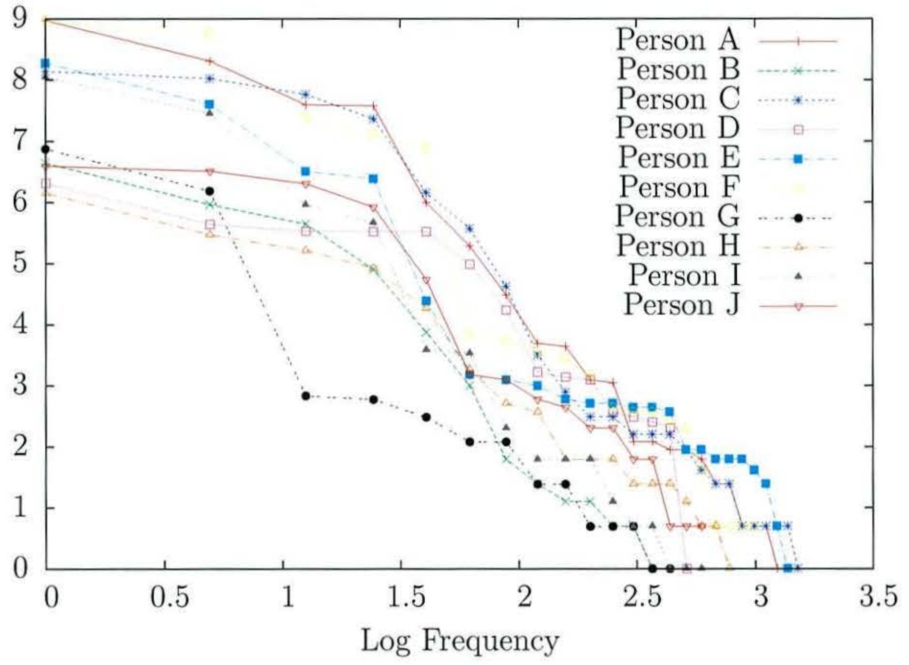


As the number of Target Symbols encountered by users was small, at most sixteen, there is very little information in this graph. It is unclear what the relationship is. Many of the users only interacted with one target symbol so have a single data point on this graph at x=0.

Action Symbol

Figure 9.13 shows log frequency plotted against log Action Symbol rank for each user. As can be seen, this relationship is clearly linear, although there are still very few data points, as only 38 unique action symbols were recorded. The linear relationship fits Zipf's law.

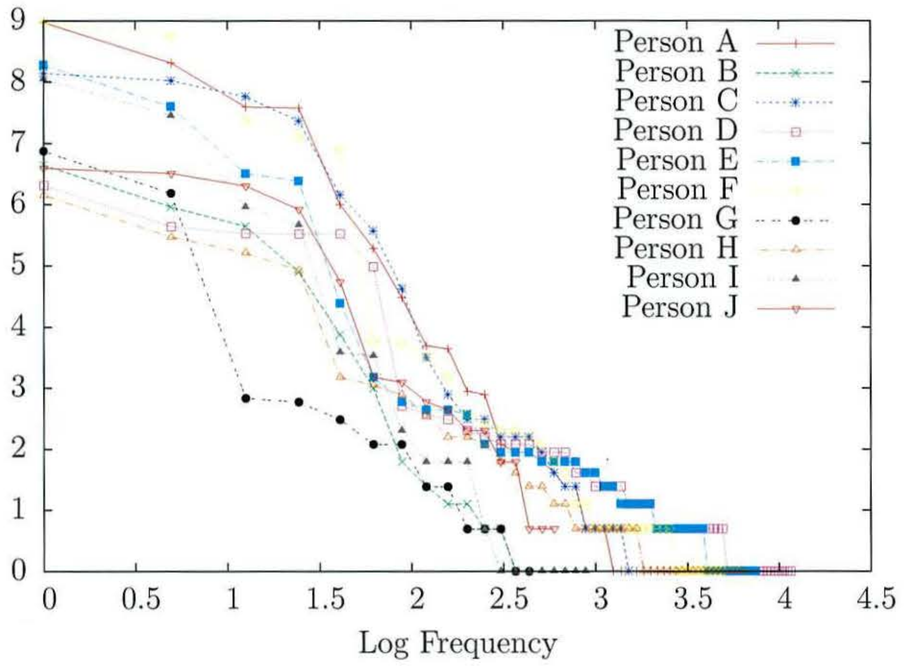
Figure 9.13: Log Frequency against Log Word Rank for Action Symbols.



Words

Figure 9.14 shows log frequency plotted against log rank Action Event for each user.

Figure 9.14: Log Frequency against Log Word Rank for Action Events.



9.2.3 Categorisation

This section will examine the performance of categorisation at the authorship task.

Categorisation of All Users

The categorisation performance will be evaluated using the basic cross validation method. It is not possible to split the users by task, as there was no identified tasks for the users to carry out. Figure 9.15 shows the overall accuracy against order size for PPM and C Measure. As can be seen from this figure, the accuracy using PPM jumps from 27 percent to 82 percent at order nine. At order ten further improves this to 83 percent.

Figure 9.15: Accuracy against Order Size.

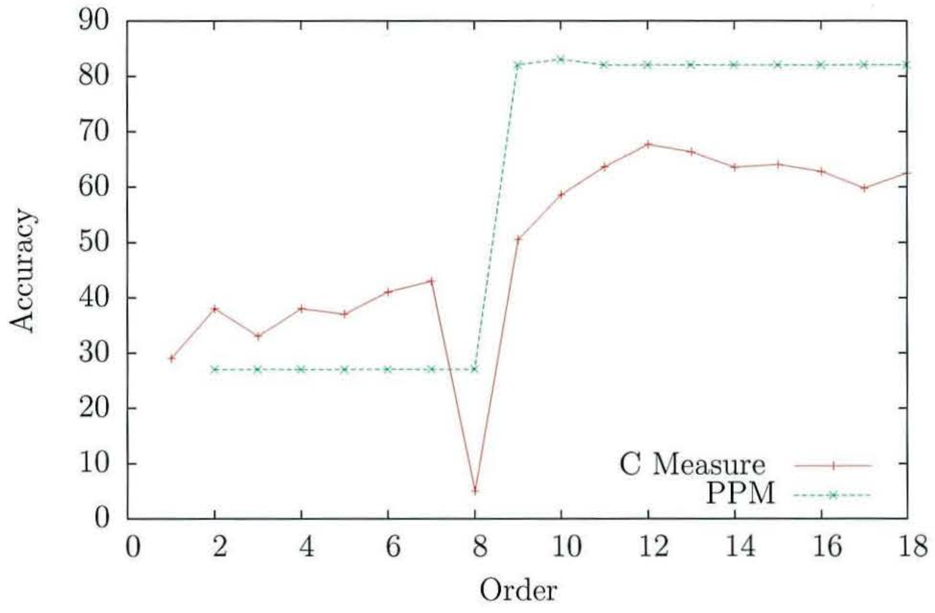


Table 9.14 shows the recall and precision for all users at order ten.

Table 9.14 also shows the recall and precision for the ten users using R Measure and C measure. The overall accuracy was for the R Measure is 68 percent. Figure 9.15 shows the performance of C Measure against order size. As can be seen from this graph, the optimal performance is at order 12 and the performance is the same as the R Measure, 68 percent. The recall and precision for the C Measure at order 12 is shown in table 9.14.

9.3 Conclusion

This chapter has analysed the two parts of the corpus created in the previous chapter. The captured User Interaction Streams have been shown to conform to Zipf's law. The task of authorship was successfully carried out on both sections of

Table 9.14: Categorisation Performance for All Users (PPM).

	PPM_{10}		R Measure		C_{12} Measure	
User	Recall	Precision	Recall	Precision	Recall	Precision
I	1.0	0.714	0.8	0.8	1.0	1.0
D	0.9	0.818	0.6	0.6	0.4	0.4
A	0.9	1.000	1.0	1.0	0.9	0.9
F	0.9	0.750	1.0	1.0	0.9	0.9
H	0.3	0.750	0.0	0.0	0.0	0.0
J	0.9	0.750	0.9	0.9	0.9	0.9
C	0.8	1.000	0.7	0.7	0.5	0.5
B	0.9	1.000	0.1	0.1	0.3	0.3
G	0.7	1.000	0.7	0.7	0.8	0.8
E	1.0	0.714	1.0	1.0	1.0	1.0

the corpus with excellent results.

These results confirm the hypothesis stated in the thesis statement that:-

> The interaction between a person and a graphical application is as unique as their writing style.

The results have shown this by correctly attributing the authorship of a stream to greater than 80 percent for both parts of the BUIS Corpus. This corpus is available for further research. Chapter 10 will examine the compression properties of User Interaction Streams.

The accuracy of over 80 percent archived when performing the authorship task is significant. Although 80 percent accuracy is not as high as the best accuracy achieved at authorship attribution for natural language (see the results in chapter 4). 80 percent is as high as the current best performance at gender and age attribution of natural language. This performance has been achieved on a very limited set of interactions more *natural* use of the applications would result in a greater diversity within the User Interaction Streams and hence higher accuracy at authorship attribution.

Chapter 10

Compressing User Interaction Streams

This chapter will describe how the User Interaction Streams can be used to perform the same function as a screen recorder. It will show that the captured User Interaction Stream is much smaller than the equivalent files produced by typical screen recording software. As described in chapter 6, a system for capturing and storing User Interaction Streams was developed to capture *meaningful* interactions between the user and the application. This chapter will describe how the captured streams can be used to playback the interactions between the user and the application.

10.1 Terminology

Typically in this thesis, the compression systems described have been lossless. Both typical screen recorders and the User Interaction Stream are lossy compression systems. This is explained in more detail on the section on equivalence in section 10.4. In this chapter, however, the word *compression* is used specifically to refer to the process of capturing, encoding and storing the interactions between the user and the application. The word *decompression* is used specifically to refer to the process of decoding the information stored by the compression process and displaying this on a screen.

10.2 Screen Recorder

A description of how a screen recorder is used for both compression and decompression is outlined below, followed by a discussion of the attributes that can be altered to change the quality and size of the captured video.

10.2.1 Compression

The screen grabber compressor consists of a separate application that is run at the same time as a user runs the application. The screen grabber will produce a series of images showing the content of the application window. The images are captured at regular intervals, the interval being set by the frame rate. The captured images are passed as a stream to a video encoder. The encoding is then carried out by whatever compression algorithm (or codec) has been selected. The codec used to perform the encoding has a significant effect on how much information is lost. This is discussed in more detail below (section 10.2.2).

Decompression

To decompress the screen grabber video, a video player is used. Measuring the size of the decompression program is difficult. The screen grabber utilises video codecs and application frameworks that are part of the operating system, for example Directshow Linetsky (2001), Adobe Flash Adobe (2008) and quicktime Apple (2006) etc..

10.2.2 Compression Quality

With the screen recorder it is possible to reduce the size of the compressed video by altering various parameters, such as:-

Frame Rate How often a new image is grabbed. The lower the frame rate, the more 'jumpy' the video appears. However if the frame rate is set too low then events can be missed.

Image Compression The grabbers use lossy image compression to further reduce the size of the video. As more compression is used, the image quality reduces. If it is set too high, then text and image data can be lost from the video.

The type of algorithm used to perform the encoding also has a great effect. There is a large body of research in the area of video compression International Organization for Standardization (2007).

10.3 User Interaction Stream Compressor

10.3.1 Compression

The compressor for the event logger makes use of the application modified to contain the event logger described in chapter 6. As the user uses the application, the captured User Interaction Stream is written to a file stored on disk. The data written to disk contains the meaningful interactions and as such is a compressed form of the interactions between the user and the application. The event logger is the compression system.

10.3.2 Decompression

The decompression program for the Action Event playback consists of the application and an Action Event Injector. The Action Event Injector inserts the recorded events into the application as if they had been generated by a user. The Action Event Injector, unlike the compressor, is a new component and will therefore be described in more detail.

Action Event Injector

The Action Event Injector opens a User Interaction Stream. The stream is then split into tokens of individual Actions Events. As described in section 5.4.4, the Action Event contains a Target Symbol and an Action Symbol. First, the Target

Symbol is decoded. Next, the actual Java Object in the current JVM that would be the target of the interaction, is identified.

The identification process is complicated slightly because the Java Help System mapping is between the Java Object and its HelpID. An inverse mapping, i.e. between helpID and Java Object, is not stored. As a result of the lack of an inverse map, it is necessary to iterate through the key set for the object Map of the Java Help System in order to identify the correct target.

The Action Symbol is much simpler to process. An equivalent Java Action Object (hereafter referred to as a Synthetic Action) to that represented by the Action Event, must be constructed. It is necessary to decode the Target Symbol first, as the Action classes in Java all require a target for the action as part of the construction process. The exact Action Class is based on the information encoded in the original Action Symbol. The relevant Synthetic Action can now be constructed. The synthetic action is injected into the Java event stack at the appropriate point. The point is identical to the point where the capture component (described in section 6.2.2) captures User Interactions. As the playback is happening within the application, any external data used by the user (for example, a data file that has been loaded) also has to be present for the decompression to take place.

10.3.3 Compression Quality

The decompression of these User Interaction Streams is, as has already been stated, lossy. The loss in the stream is not in image reproduction quality. The loss is caused by the fact that, although the User Interaction Stream preserves the order that the Action Events occurred in, the interactions that are recorded are only those defined as *meaningful*. Interactions such as moving the mouse around within a graphical component are not recorded. The time that an event as a delta from when the application was started is also recorded with the Action Event.

10.4 Equivalence

This section will examine differences between a typical screen recorder and a User Interaction Stream Compressor, to show that a comparison between the files produced by each is valid.

The User Interaction Stream Compressor records only the meaningful interactions with no temporal information. The screen recorder records all interactions, meaningful or not. Though the User Interaction Stream Compressor does not record all interactions, any that alter the state of the application are recorded. For the purpose of recording a user performing a task, the two systems are therefore equivalent.

10.4.1 Interaction Capture of 3D interfaces

The Interaction capture system described in this thesis captures interactions between a typical desktop application and a user. Some applications have interfaces that represent 3 dimensions. These applications create a large number of dynamic objects that a user can interact with. The implementation of the interaction capture system outlined in this thesis would not capture the meaning of these interactions accurately. The interaction capture system could however be extended to capture these interaction in a meaningful way. This extension would require the capture system to create new target symbols for each object that a user could interact with.

10.4.2 Practical Equivalence

The interaction capture system can only be used to playback a user's events if the same data used by the user is available to the person performing the playback. This data may, under some circumstances be larger than a video recording, which would offset one advantage of the interaction capture system. The playback of a captured interaction stream can be paused, however unlike a traditional video it can not be rewound, and it is not possible to jump to a point in the capture stream. The interaction capture streams can not be edited to only show selected sections, for example for use in a presentation.

The interaction capture system has one significant advantage for a software developer. It can used in conjunction with an integrated development environment (IDE). This allows developers to add breakpoints to the software to examine the internal state of the software, or alternatively to profile the usage pattern produced by the user.

10.5 Compression Performance

As the two different systems are equivalent, the performance of each system can be evaluated by comparing the size of files produced when recording a user performing a specific task. When evaluating the performance of the different compression systems, the size of both the stream and the compressor/decompressor should be compared. This is especially difficult with screen recorders, as the operating system provides many of the facilities needed for playback (see section 10.2.1.) As a result, the only size that is being considered in this research is that of the file needed to perform playback.

10.6 Experimental Setting

To compare the performance of a screen grabber with the User Interaction capture system, three different screen recorders were selected. They are:-

CamStudio CamStudio *CamStudio* (2007) is an opensource screen recording software for Microsoft Windows XP.

Freez Screen Video Capture Freez Screen Video Capture *Freez Screen Video Capture* (2007) is a screen-capture and screen-recording tool to record screen activities and sounds into standard AVI video files.

Instruments, UI Recorder The UI Recorder tool is provided as part of the Instruments application Apple (2008). Instruments is part of the developer tools for OS X. One package provided by Instruments is the UI Recorder. This is used to record and playback User Interactions into an application, to automate the driving of applications for testing.

The application used to perform the capturing was the same modified version of WEKA as was used for the experimentats detailed in chapter 8.

To perform the experiment, part three and part four of lab one (described in appendix A) were carried out. This was done once for each of the screen grabbers and once to capture the interactions with the User Interaction capture system. The size of the resultant files were then compared.

The performance of the application using Instruments was noticeably slower. Although it was attempted to use Instruments to capture the interactions, a bug in Instruments meant it was impossible to change the classification algorithm. As a result of this, it was not possible to use Instruments to perform this recording. The capture system developed for this thesis had no noticeable impact on the performance of the application.

The second experiment consisted of performing tasks five six and seven from lab one. This second experiment was again performed 3 times for each screen recorder. Experiment three consisted of performing all of Lab two. Experiment four consisted of playing the Asteroids game for two minutes.

10.7 Results

After performing the tasks described above, the size of each file produced by each of the two screen grabbers and the User Interaction Stream was measured. This is shown in the table 10.1. The tasks were performed three times and an average was taken. The file size was measured in bytes.

As can be seen from the table, using a User Interaction Stream to capture the meaningful interactions between the user and the application results in a compression system that is considerably better than a screen recorder. On average 10 - 20 times better.

Table 10.1: Compression Performance of Screen Capture Systems

Recorder Application	**File Size in Bytes**	**Compressed with `Bzip2`**
	Experiment One	
User Interaction Stream	10537.33	647.33
CamStudio	2290176	77412.67
Freez	2710528	88693
	Experiment Two	
User Interaction Stream	47508	3393
CamStudio	12656981.33	404231
Freez	21441194.67	880262.67
	Experiment Three	
User Interaction Stream	90111.33	6088.67
CamStudio	64473088	1771962
Freez	94799189.33	2162960.33
	Experiment Four	
User Interaction Stream	17453	2103.33
CamStudio	4042752	413720.33
Freez	6569130.67	643527.33

10.8 Lossless User Interaction Compression

This section will examine the compressibility of the user interaction streams in a lossless way. The compression algorithm that will be used is PPMD. In this section, unlike the previous sections of this chapter, the term compression here refers to lossless compression. The compression performance of the two parts of the BUIS corpus will be examined. Initially the WEKA part of the corpus is analysed followed by the analysis of the Asteroids section of the corpus.

10.8.1 Weka Compression performance

In the following section, the data is analysed to determine how compressible it is.

Compression is measured in bits per symbol. The captured data was turned into a stream of numbers. This was then compressed using the PPM numerical compressor from the Text Mining Toolkit (TMT) *Text Minning Toolkit* (2008). Which was used to evaluate the modelling of streams of Chinese text Wu & Teahan (2005). This algorithm allows each symbol to be represented as a whole number. As Chinese characters are represented as a series of two bytes, the compression algorithm generated for this is very appropriate for the streams that were produced in this experiment which was also a series of two bytes.

The analysis of the BUIS corpus data was divided into two sections: the compression performance on streams consisting only of the Target Symbol and the compression performance on streams of Action Events.

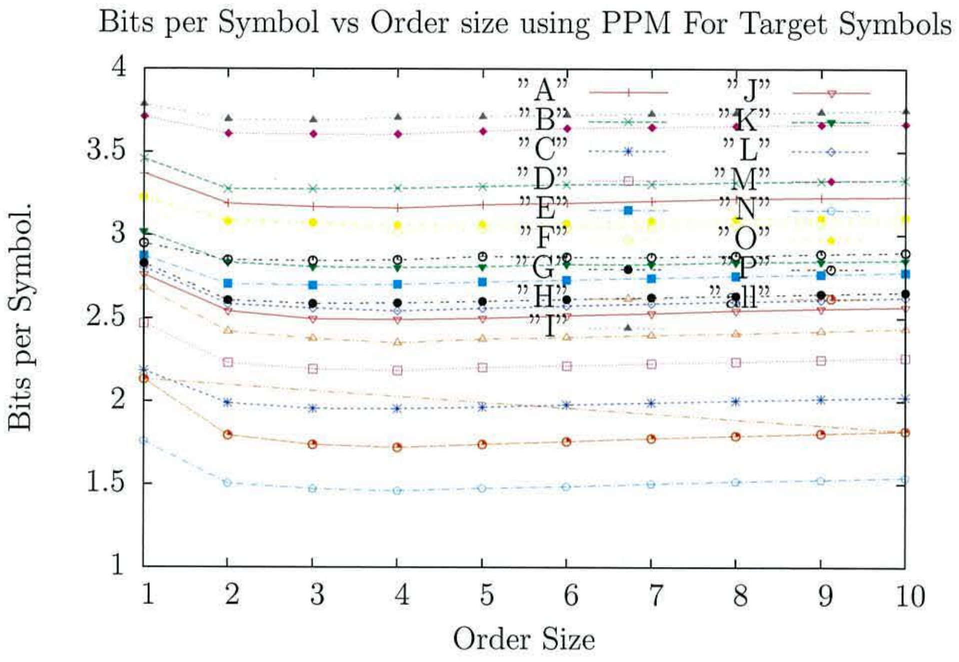
Target Symbol Compression

In this section the compression of Target Symbols will be examined.

Target Symbols were sorted and assigned numbers, starting from 0. A User Interaction Stream was then produced, by ordering the Target Symbols sequentially. Next, the files were compressed using the TMT's number compressor. As the TMT uses PPM as a compression algorithm, the variable that can affect compression is the order, or maximum model size. (Chapter 3.5.1 describes this in detail). There has been much research into picking the optimum order size and how this relates to the domain of the stream. The optimum size for this type of stream has obviously not been determined.

Maximum compression can be defined as the lowest number of bits needed to encode a symbol. As described in section 9.1.1, there were 236 unique Target Symbols. Without any compression, this would need 7.88 bits per symbol. Figure 10.1 shows the number of bits per symbol against order size, when carrying out Target Symbol compression for each of the users and *all users*. The User Interaction Stream *all users* was obtained by concatenating the Target Symbol stream for each user together.

Figure 10.1: Target Symbol Compression.



As can be seen from figure 10.1 the Target Symbol stream created by user N was the most compressible, followed by the Target Streams of all other users concatenated together.

Table 10.2 shows the lowest compression against user and model size. The users who generated many events (see table 9.1) users N D and C have all produced more compressible data than the all the other users. The average compression ratio is: 2.75 bits per symbol with a standard deviation of 0.59. As stated earlier in this section, encoding the symbols with no compression would result in an encoder encoding 7.88 bits per symbol. The average figure of a compression ratio, 2.75, shows greater than fifty percent compression.

Table 10.2: Minimum Compression of Person against Target Symbol.

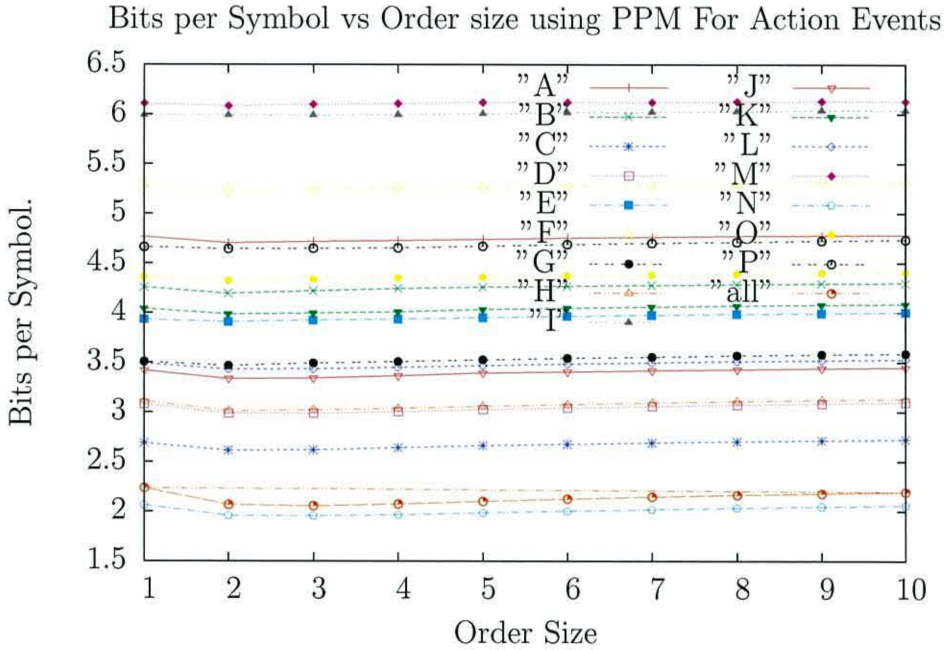
Person	Bits per Symbol	Order Size
N	1.465	4
all	1.730	4
C	1.966	4
D	2.197	4
H	2.363	4
J	2.508	4
L	2.556	4
G	2.602	3
E	2.713	3
K	2.816	4
P	2.859	3
F	3.032	4
O	3.069	4
A	3.182	4
B	3.290	2
M	3.617	4
I	3.705	3

Action Event Compression

In this section, the compression performance of *Action Symbols* will be evaluated. The performance of PPM for carrying out compression will then be examined. As described in chapter 5.4.3, the Action Symbols are made up of two components; the Target Symbol and the Action Event. To encode these two symbols, they were considered as a single number. A list of all unique Action Symbols was created and each assigned a number from 0. Each time a symbol was encountered in a stream, this was replaced by the number assigned to that symbol.

Modelling as a single number is simple. It is directly analogous to modeling a western language such as English, as each of the characters in ASCII maps each English character to a single byte. Figure 10.2 shows the relationship between order size and compression.

Figure 10.2: User Interaction Stream Compression



The graph is in many ways similar to the compression of just the Target Symbol, shown in figure 10.1. There are, however, some important differences. The User Interaction Stream has a much greater variance between the best and worst compression.

Table 10.3 shows the minimum compression against user and model size.

Table 10.3: Minimum Compression of Person against Action Symbol.

Person	Bits per Symbol	Order Size
N	1.957	3
all	2.063	3
C	2.623	2
D	2.999	2
H	3.021	2
J	3.345	2
L	3.439	2
G	3.478	2
E	3.923	2
K	3.997	2
B	4.203	2
O	4.332	2
P	4.661	2
A	4.725	2
F	5.249	2
I	6.001	3
M	6.106	2

From this it can be seen that, like the Target Symbol compression, user N is more compressible than all the streams concatenated together. The order of compression performance is preserved for the first nine users. The order with maximum compression is also of note. The optimum model size is 2 for all users apart from N and I. N being the user with the most events and I being the user with the least. The average compression is 4.00 bits per symbol and the standard deviation is 1.16. This indicates a much greater range of compression than when compressing the Target Symbols. The difference in compression ratios appears to be inversely related to the number of Action Events that a user generated.

10.8.2 Asteroids Compression Performance

As with the first part of the BUIS corpus, the captured User Interaction Streams' compressibility was examined. As has already been discussed, there are only 16 Target symbols and many users only interacted with one of them. Similarly, the Action Symbols that users generated were limited to only 38 unique symbols. As a result of this, only the compression of Action Events was examined.

Action Event Compression

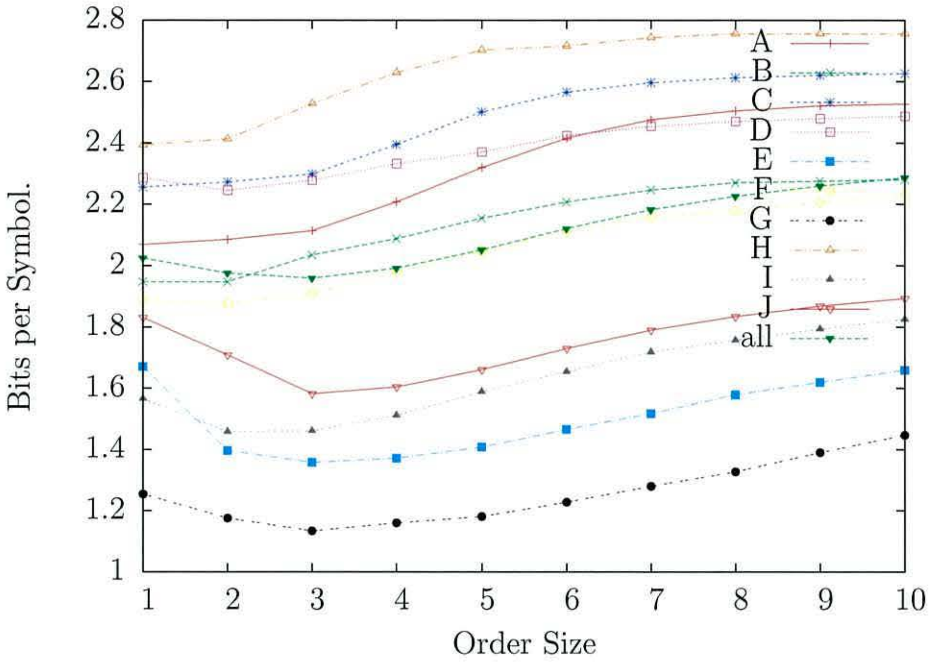
Figure 10.3 shows the relationship between order size and compression. As can be seen in the figure, there are two distinct sets of users. The graph shows the number of bits per symbol increasing as the order size increases for five users — A, B, C, F and H. The other five users, D, E, G, I, and J have a more typical curve. The number of bits per symbol decreases as the order size increases, until a minima is established. Then the number gradually increases. Of the five users that exhibit this behavior, all but one, D, have higher compression ratios than all the other users. The average compression (shown by the user labeled all) is achieved at order three.

Table 10.4 shows the minimum compression ratio for each user an the order size at which this occurs.

Table 10.4: Minimum Compression of Person against Action Symbol.

Person	Bits per Symbol	order size
A	2.069	1
B	1.947	1
C	2.255	1
D	2.245	2
E	1.357	3
F	1.877	2
G	1.134	3
H	2.393	1
I	1.458	2
J	1.582	3

Figure 10.3: User Interaction Stream Compression.



10.9 Summary

This chapter has shown how the captured User Interactions Streams can be used as a screen recorder. This usage as a screen recorder is a lossy compression system, although the information lost is different to a typical screen recorder. The experiments have shown that the compression is an order of magnitude better than a screen recorder.

The lossless compression characteristics of the User Interactions Streams when compressed using PPMD have been examined. This examination looked at both parts of the BUIS corpus. The order size with maximum compression was shown to be different for both parts of the corpus.

Chapter 11

Conclusions and Discussion

This chapter summarises the techniques that have been developed and evaluates their contribution to the field of Computer Science. It discusses ways forward in applying and developing the techniques.

11.1 Contributions

The introduction described the main aim of this thesis as to:-

> perform the task of authorship on a stream produced by capturing interactions between a human and a computer.

In order to accomplish this task a collection of techniques and tools were developed. This section will summarise these contributions.

11.1.1 Capture System

In order to perform the task of authorship on interactions between a human and a computer, it was necessary to design and implement a system to capture the interactions as a stream of symbols. The design methodology for creating an interaction capture system that captures meaningful interactions between a user and a graphical desktop application was explored. New terminology, 'Target Symbol', 'Action Symbol' and 'User Interaction Stream' was introduced and defined. These are summarised below.

The Target Symbol refers to the graphical widget that is the target of an interaction. Action Symbols are used to refer to the *kind* of interaction made by a user, e.g. mouse movement. The combination of the Target Symbol and Action Symbol is referred to as an Action Event.

The implementation of the capture system in Java was described. As part of this implementation, a tool was developed to automate the process of assigning Target Symbols to an existing application.

In addition to capturing the meaningful interactions, it was demonstrated that the captured User Interaction Stream could be used to replay the interactions that a user had made with the application. The playback of interactions into the application allowed the capture system to be used as a screen recorder with a high level of compression.

11.1.2 Interaction Corpus

After designing and building the capture system, two very different applications were modified to produced User Interaction Streams from users using the applications. The applications chosen were very different. WEKA is a powerful data mining application and has a complex GUI. The other application, Asteroids, is an early computer game with a limited GUI. The first application, WEKA, was used by sixteen University undergraduate students over two years, as part of their studies. A detailed rationale leading to the choice of both WEKA and Asteroids was outlined. A description of the tasks the users performed on these applications for the purposes of this research, was given.

The User Interaction Streams captured during both experiments together form a new corpus. The corpus, referred to as the BUIS corpus, is available for further research from *Bangor User Interaction Stream Corpus* (2008).

11.1.3 C Measure

A new text categorisation algorithm, called the C Measure, was developed. This algorithm was shown to have excellent performance at the task of authorship. The relationship between compression performance and categorisation performance was explored in detail. From this it was shown that the optimal order size for compression did not correspond with the optimum model size for text categorisation.

This is an interesting result and is further indication that there are still improvements to be made in compression performance, as there is additional information to be exploited at the higher orders. This is also indicated by the fact that Shannon has estimated the entropy of English to be around $0.6 - 1.1$ bits per character, and PPM only compresses English to 1.4 bits per character. Although the C Measure performed well in natural language, it did not perform well at the authorship attribution task for User Interaction Streams. The reason for this is not clear. It may be that the common sub-strings occur less often.

11.1.4 Visualisation

Two visualisation techniques were developed. One was a simple tree view of the trie data structure. This was developed in order to allow manual verification of the data structure. The other visualiser showed a directed graph view of a stream. This directed graph view can be used to explore the contents of a stream and highlight contexts.

11.2 Results And Conclusions

A detailed analysis of the BUIS corpus, produced by capturing User Interaction Streams using WEKA and playing Asteriods, was given in chapter 9. This analysis was split into two sections. First, the WEKA section of the corpus was examined. This was followed by an analysis of the Asteroids section of the corpus.

11.2.1 WEKA

The similarities between the components of a User Interaction Stream and a natural language were shown. Following this, the performance of the authorship task was evaluated. The initial results were poor. The poor results were caused by a combination of the method of cross validation used and the relatively small number of Action Events that were generated by all but three of the participants. By using the three top users and a different cross validation technique, an accuracy of 81 percent was achieved for the best categorisation algorithm, PPM.

11.2.2 Asteroids

This same analysis was then carried out on the Asteroids section of the BUIS corpus. The similarities between the components of a User Interaction Stream and a natural language were again evaluated. The simpler interface of the Asteroids game resulted in very few Target Symbols. The authorship task was again evaluated. The results, 83 percent, showed a high degree of accuracy, again using the PPM algorithm.

11.2.3 Summary

It is interesting that WEKA, with a much more complex interface, required many more events to achieve accurate performance at the authorship task. This may be as a result of the combination of the strict laboratory script and much more complex interface of WEKA. Overall accuracy (over 80 percent) was achieved for both parts of the corpus. Although this is not as high as the current research in authorship of natural languages (over ninety percent) it is still a significant result, supporting the hypothesis given in the thesis statement that:- "the interaction between a person and a graphical application is as unique as their writing style."

11.3 Further Work

The work undertaken for this thesis was concerned with testing the hypothesis stated in the thesis statement. In order to do this a new User Interaction Capture System was created and a new corpus, the BUIS corpus, was produced. At the conclusion of this work, it is evident that the techniques and tools that have been developed as a result of this thesis have the potential to be used for further research. Some of these experiments would require access to specialist equipment, such as a HCI lab, facilities that were unavailable for use in this research.

11.3.1 Corpus

The BUIS corpus has been successfully used to analyse the authorship task. There is scope for the corpus to be extended. The WEKA section of the corpus was limited by the number of students who took the Artificial Intelligence course and the constraints of the course syllabus. This part of the corpus could, therefore, be extended. For example, it could be augmented with additional users. Also, the tasks carried out by the users could be expanded. As described in section 9.1.1 the lab script only caused users to encounter 236 of the 545 Target Symbols that were assigned to WEKA. A larger set of tasks could be constructed, in order to encourage the user to interact with every unique widget in the application. In addition to covering *more* of the application, additional tasks could be added to the Lab script A, to ensure that more Action Events were generated by the users. This would then ensure that sufficient Action Events were available to carry out the authorship task with more participants.

For the purposes of this thesis, the BUIS corpus was used only to evaluate the authorship task. However, it could also be used for other tasks. For example, the corpus contains a distinction between tasks. During this research, as described in section 9.1.3, many users forgot to record when they had completed a task. If a mechanism could be devised to accurately record the completion of a task, then the job of task detection could be attempted. It might be possible to determine the task boundaries by manual examination of the User Interaction Stream and looking for new Target Symbols that would be used in different tasks.

In the BUIS corpus, the users were anonymous. If, however, the experiment were to be repeated, the age and sex of the users could be recorded. With this additional information, the tasks of age and gender identification could be attempted.

11.3.2 Potential Uses of the Interaction Capture System

As described in the thesis statement, a technique to capture human computer interaction as a stream of symbols (User Interaction Stream) was designed and implemented. To support this technique, a toolkit was developed to automate the task of creating an application that captures user interactions. The mechanism for recording has been shown to be unobtrusive during the use of an application. A discussion of further applications that may benefit from this technique follows.

Cheap HCI

As described in the literature review 2.5.3, a technique to perform *cheap* HCI studies has been developed. The use of the interaction capture system as a compression system would enable a researcher to watch the interactions between the user and the application, using considerably less bandwidth than a traditional screen capture system.

The User Interaction Streams produced by the user could be played back, allowing a developer to see how a user was interacting with an application. Although this does not allow the developer to see the user, the simple addition of a cheap web cam would provide this information.

As the streams are meaningful interaction, the User Interaction Stream could be examined algorithmically to look for common patterns. The common patterns may well be indicative of a common task, or a problem with the interface. These *hot spots* could be found automatically then examined in detail to determine if there was a more optimal way of helping the user with the task. A developer would then be able to improve the application by, for example, providing a shortcut, or if the shorcut already existed then hinting to the users the existinace of a shortcut.

Black Box Recorder

By saving the User Interaction Stream constantly to disk, or a remote database, as described in chapter 5, should the application ever crash, a developer would be able to see all the interactions that led to the crash aiding debugging.

Continuous Authentication

By identifying users based on their interactions with an application, the presence of a different user could be detected, as the stream would change. This would be similar to detecting boundaries in text documents Rey & Reynar (1998). As this authentication happens during *normal* use of an application, it is unobtrusive to the user.

User Tuition Analysis

The streams produced by users could be evaluated before and after tuition is given. This would show up whether or not the tuition had been effective, as there would be a change in the models. In addition, the models might indicate areas where additional tuition would be beneficial.

11.3.3 Testing System

Automated testing of graphical applications is much more difficult than automated testing of non graphical applications. There are some tools that exist to provide automation. For example, `jemmy` Netbeans (2008) is an automated test system for swing applications. One of the major problems with automated systems like `jemmy` is the time taken to produce each test. Typically, each test is a script written by hand. These scripts tend to be brittle, as they rely on the absolute positions of the GUI components. The User Interaction Streams could be recorded during *normal* use of an application. This stream can then be played back into the application to test it. The design of the capture system ensures that it is not brittle.

11.3.4 Cognitive Monitoring

There has been some previous work Jimison et al. (2004) which has shown that it is possible to detect mild cognitive impairment by monitoring the use of the game Freecell. The rich data stream captured using the techniques outlined in this thesis provides researchers with an additional data source to analyse. It is hypothesised that analysis of this data stream could be used to augment the research in this area of detection.

11.3.5 Tool Extensions

Tagging Tool

A major further development of this system would be to make it idempotent and self updating. To make the tool idempotent, it should keep a track of which lines of code have already been altered and not alter these lines again. A self updating system would allow re runs after adding new widgets, without hand coding, to avoid collisions.

The system could also use reflection to instantiate all classes that are constructed and then see if they are GUI components. This would allow the tagging system to tag objects that extend the basic GUI widgets without intervention.

Stream Visualiser

The stream visualiser, described in chapter 7.2, has potential for a large amount of further extension.

Clustering Analysis of the stream would show that there are clusters of symbols. For example, in English the word "the" is very common. The visualiser could be enhanced to detect and show clusters like this one.

Component Display When playing a user interaction stream, the component represent by a Target Symbol could be imaged and displayed rather than the target symbol. A set of icons to encode the meaning of the Action Symbol would also increase the information displayed to a user.

11.3.6 Grammar Discovery

Chapter 5.5.1 described how every graphical application has an associated grammar. The grammar for a small application was fully described. In a more complex application such as WEKA, the grammar is also much more complex. Evaluating the grammar by hand would be extremely time consuming. It could be possible to generate the grammar for a Java Swing application automatically. The potential technique to do this is outlined below.

Every Java GUI component has a parent. The parent is accessible via the `getParent()` call method. This method is defined in the `Component` class (Flanagan 1998), that is extended by all Swing components. Using this method, the graphical *root* object can be discovered. This *root* of the tree will be the *main* frame. Given the root, a standard treewalker algorithm can be used to determine the locations, and hence the grammar, of the application. This technique does have some limitations, however. Applications that add new graphical components when loaded with data have a grammar for each data set. The outlined technique would be tied to a specific loaded data set.

Grammar Uses

This grammar could be used to perform automated application testing. Fuzz testing, as described in Miller et al. (1990) started by inputting random data into console applications, whilst monitoring the application to see if it had crashed. This has been extended to cover graphical applications (Forrester & Miller (2000) and Miller et al. (2007)). If the grammar of an application was known, then fuzz patterns could be generated with the knowledge of the state that they will be driving the application to. By comparing the User Interaction Streams generated under normal use with the known states that an application could be in, testing could be improved by targeting states that are not typically encountered. This would mean there would then be a higher chance that the fuzzer would cause the application to hang or crash.

11.4 Conclusion

This thesis has introduced a new and novel technique to capture interactions between a user and a graphical desktop application. This technique has been used to capture the use of two different graphical applications. These captured interactions form the BUIS corpus which has been made available for use by other researchers. This corpus has been analysed and the performance at the task of authorship categorisation has been examined and shown to be robust. In addition, a new algorithm for performing text categorisations has been developed.

As a result of this research, the production of the BUIS corpus and the resulting development of creating a new algorithm for performing text categorisation have opened up further channels for interesting research in this field.

Appendix A

Lab Script

The following pages contain the full lab script used in chapter 8. The lab took place in two sessions these have been split into lab one and lab two.

A.1 Lab One

ICP3037 Lab 09

Data Mining with WEKA: Introduction

These labs provide sample exercises and a guide to materials that are relevant to the module ICP3037. The exercises are to help you become familiar with the material and help you with the assignment.

Lab work will be assessed for this module. It will count for 15% towards your final overall mark, and should be your own individual work. Some of the lab exercises may also appear as exam questions, so it is in your own best interests to complete the lab ….

For today's lab, write the answers to the exercises and include the requested screenshots in a Microsoft WORD document, archive in a zip file the exercise 3 files and email the lecturer (wjt@informatics.bangor.ac.uk).

Background reading: *Data Mining* by Ian H. Witten/Eibe Frank.

Main WEKA website: http://www.cs.waikato.ac.nz/ml/weka/

For some lectures on Data Mining, go to /homedir/course/icp3037/weka-docs or N:\course\icp3033\weka-docs.

Note: All the files you will need for this lab are in the weka subdirectory of the main module directory on the course drive (/homedir/course/icp3037/weka or N:\course\icp3037\weka).

Introduction to data mining

In this lab you will learn about classification through decision trees by using a powerful tool written in Java/Swing called the Waikato Environment for Knowledge Analysis (WEKA). WEKA is a general tool for data mining, and can analyse data using a variety of methods. Today we will concentrate on classification of data and become familiar with the basics of WEKA's operation.

Task 1 (2 marks):

How many times have you used WEKA before?

---

1. ARFF data files

The first, most obvious requirement, is that we need to supply WEKA with a data file to work on.

We will first look at a very small data file pertaining to the prescription of contact lenses. Before opening the file under WEKA, it is worthwhile having a brief look at the data file in a text editor. WEKA uses "ARFF" format data files, which are effectively comma-separated value (CSV) format files, with some additional information.

Using the text editor/viewer of your choice, open up the file "contact-lenses.arff" which can be found in the weka sub-directory.

The first part of the file is simply a human-readable description of the data (all lines starting with a "%" symbol are comments). Next, the file defines the *attributes* for the data. These attributes correspond exactly, in order, to each of the data fields separated by commas present in the data section of the file (delimited by the "@data" tag). The curly braces enclose the possible values the attribute may take. The attributes are the particular entities for which we are actually defining the data. The "@relation" line specifies the particular *feature* we desire to extract from the file; here, we wish to use the data to determine the appropriate type of contact-lens suitable for a patient, so we define relation to be "contact-lenses" corresponding to that attribute.

The simplicity of the format means that it is very simple to create your own ARFF files from either a spreadsheet or database for use with WEKA.

2. Starting WEKA

Using your favourite Web browser, type in the following URL:

`http://yella.informatics.bangor.ac.uk:8080/`

Then enter your username as your first name, and last name. Then an "Enter task details" pop-up box appears. Then for each of the tasks below, type in the task number into the "Task Name" field, plus a short description of the task details. (Note: if you have to revisit a task, please add details that you are changing your answer for this task).

Now click the New Task button. When the "GUI Chooser" window appears, click on the "Explorer" button.

Task 2 (5 marks):

Spend some time familiarizing yourself with the WEKA environment. The purpose of this exercise is to find out as much as possible by using the software directly. The software itself has been altered to include the ability to log everything you do (mouse movements, tab selection, what you type in, etc.). We are recording this for this lab and next weeks' lab as part of a research project. You will receive the full 5 marks for this exercise simply by making a reasonable attempt at becoming familiar with the user interface provided by WEKA.

Enter whether you feel comfortable with using WEKA at this stage. Also highlight any problems with the user interface, or what features you like the most about it.

When finished, click the Task Complete button, and then enter details for the next task.

---

3. Loading a data file

Click on the "Open file..." button in the top-left of the "Preprocess" tab. Open the file "contact-lenses.arff" which we looked at earlier.

Once the file has loaded it will display a numbered list of the attributes in the "Attributes in base relation" pane. If you click on each of these in turn (make sure not to uncheck them) you can view a brief summary for each attribute in the lower right pane, with a count listing for each particular attribute value.

Classification in data mining

One of the most important tasks in data mining is classification. Given a large corpus of data, what we ideally want to do is to automatically classify new instances of data in accordance to our data set - we want to *infer* rules, or classification schemes, based on our existent data. For example, if we wished to produce a system that could perform automatic diagnosis of patients suffering from cancer, we could provide it with a large database of the various forms of cancer and their antecedent factors (preconditions), and through data mining allow the system to automatically adjudge a new patients risk of contracting cancer.

Classification is this process of identification. The key point is that we wish the necessary rules for accurate classification to be constructed *automatically* from the data, not manually created by us.

Decision trees

Once such technique used in data mining to achieve this automatic "rule-creation" is the use of *decision trees*. We will now get WEKA to generate a simple decision tree to classify our contact lens data.

---

4. Performing the classification

Click on the "Classify" tab at the top of the Explorer window.

Next, we need to select the classification algorithm used by WEKA. WEKA provides a large number of classification algorithms; to select the one we're interested in, click on the thin classifier button at the top of the window (it should currently be reading "ZeroR"). From the pop-up box that appears, select **weka.classifiers.trees.J48** from the drop down menu. Leave the other options at their defaults, and click on the "OK" button.

Make sure that "Cross-validation" option is checked in the "Test Options" pane (it should be by default), then click on the "Start" button just below it.

Take some time to look over the output produced.

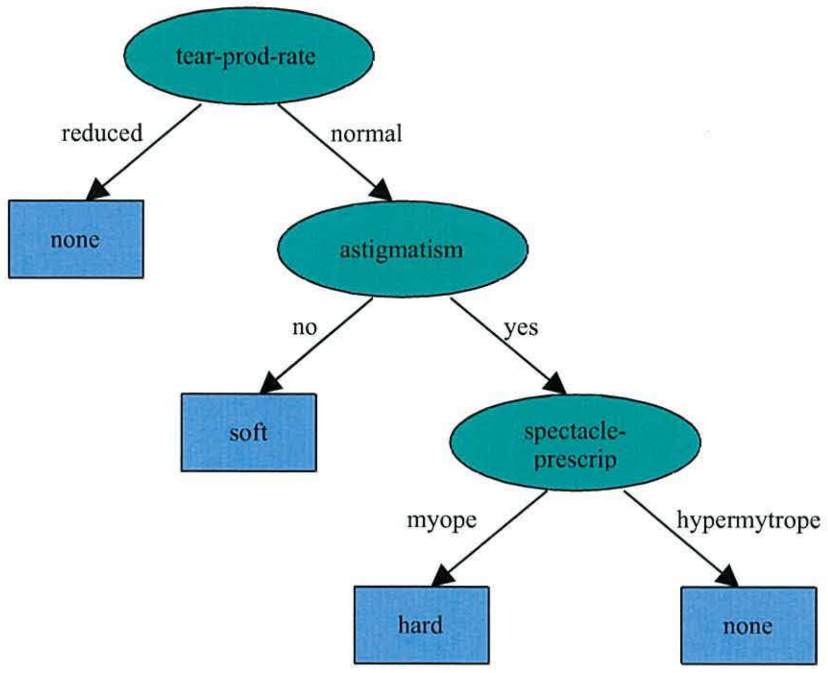
The first section is simply a summary of the data file, with a listing of the attributes and the number of items (instances) present in the file.

Next, you should have something like the following lines:

```
J48 pruned tree
---------------

tear-prod-rate = reduced: none (12.0)
tear-prod-rate = normal
|    astigmatism = no: soft (6.0/1.0)
|    astigmatism = yes
|    |    spectacle-prescrip = myope: hard (3.0)
|    |    spectacle-prescrip = hypermetrope: none (3.0/1.0)

Number of Leaves    :    4
Size of the tree    :    7
```

These lines depict the decision tree WEKA has produced from the data. Each level of indentation represents a level in the tree. The values specified after the colons are the contact-lens value (label) that should be prescribed based on meeting those conditions.

Graphically, the decision tree looks like this:



As you can clearly see, there are four square leaves - each here corresponding to the appropriate types of contact lens prescription, the "decision" - and the tree has a total of seven nodes.

It should be fairly clear how a decision-tree enables us to determine an appropriate decision based on the instance data by simply following the edges. The decision tree behaves like a series of "IF...THEN..." rules; so we could read the tree above as:

```
IF tear production rate = REDUCED
    assign them no contact lenses

IF tear production rate = NORMAL
    AND astigmatism = NO
        assign them soft contact lenses

IF the patients tear production rate = NORMAL
    AND astigmatism = YES
```

```
        AND spectacle prescription = MYOPE
            assign them hard contact lenses

IF the patients tear production rate = NORMAL
    AND astigmatism = YES
        AND spectacle prescription = HYPERMETROPE
            assign them no contact lenses
```

These rules could, of course, be written more efficiently and compactly if we assume ordering in the rules and allow ELSEs, but written this way it is clear how we can start from the root node each time and arrive at the decision. This decision tree is compact and simple, but the trees can become considerably larger if more complicated chains of logic need to be followed, and if we have large numbers of attributes and values.

Click the Task Complete button, and then enter details for the next task.

Task 3 (5 marks)

Experiment with changing some of the options to the J48 classifier algorithm, and try running it on some of the other data files in the directory. In particular, experiment with the "binarySplits" and "unpruned" options and observe their effects on the generated decision tree. Post a summary of what you observe here.

Click the Task Complete button, and then enter details for the next task.

Task 4 (5 marks)

Compare the performance and trees generated by the divide-and-conquer decision tree algorithm, ID3 (**weka.classifiers.trees.Id3**) with J48, noting any differences.

Click the Task Complete button, and then enter details for the next task.

Task 5 (10 marks)

Experiment on the data sets with some of the other classifiers present in WEKA. Write a summary of what you have found out here.

Click the Task Complete button, and then enter details for the next task.

---

5. Clustering

Clustering is an extremely complex topic within data mining, so we will only touch on the basics here. Essentially, clustering refers to aggregating the data into groups based on certain criteria and mathematical transformations. Whereas with classification we already know the classification categories *prior* to performing the classification, with clustering we attempt to infer these too.

Make sure you have the file "zoo.arff" open.

Click on the "Cluster" tab. In the "Cluster mode" pane, select the "Classes to cluster evaluation" option, and select "(Nom) type" from the drop-down menu. Next, click on the "Clusterer" button at the top, and select "**weka.clusterers.SimpleKMeans**" from the drop down menu. Change the "numClusters" field to 3. Click on "OK" then click on the "Start" button to perform the analysis.

You should get some output which includes the following:

```
 0   1   2   <-- assigned to cluster
 0   0   41   |   mammal
 1   19  0    |   bird
 4   1   0    |   reptile
13   0   0    |   fish
 4   0   0    |   amphibian
 0   8   0    |   insect
 7   2   1    |   invertebrate

Cluster 0 <-- fish
Cluster 1 <-- bird
Cluster 2 <-- mammal
```

Here, we are performing the clustering based on the *type* of animal. We specified that the algorithm should generate three clusters, and these are the three clusters it has generated based on our data. The matrix of data shows the number of instances of each type of animal, and the cluster it was assigned to. The cluster name is determined by the majority membership by the type of animal; whilst cluster 0 has 1 bird, 4 reptiles, 4 amphibians, and 7 invertebrates, it has 13 fish, so it is therefore classified as "fish".

Click the Task Complete button, and then enter details for the next task.

Task 6 (5 marks)

Experiment with different parameters to the clustering algorithm and observe the effects. Next, open the file "primary-tumor.arff" and use clustering to determine which types of tumor are most prevalent. Post your answer here.

Click the Task Complete button, and then enter details for the next task.

Task 7 (5 marks)

WEKA also provides a convenient option for data visualisation. With the file "primary-tumor.arff" still open, click on the the "Visualize" tab. By default, both the X and Y axis will be set to the instance number, which is not very informative. You can select the attribute displayed on the axis by selecting it from the drop-down menu for the corresponding axis (i.e. the drop-down menus currently reading "X: Instance_number (Num)" or "Y: Instance_number (num)").

Along the right-hand side panel you can view miniature previews of the graphs against each parameter, whilst the two other drop-down boxes allow you to alter the colours used for displaying the data. The "Jitter" slider allows you to introduce some "randomness" into the data.

Take some time to look over the data by viewing different attributes along each axis (in particular look at "Instance_number "against "class"). Post your observations here.

Click the Task Complete button, and then enter details for the next task.

Task 8 (10 marks)

We have now briefly looked at all of the areas present in the Explorer within WEKA. Spend any remaining time exploring other data files, performing analyses, using any of the features as you wish. Post a summary of what you did into the box provided.

If you have some suitable data (if not, invent some!), you could create your own ARFF data file. Simply export the data from your spreadsheet/database or data source in CSV format, then manually edit the file in a text editor to add the other information. If you encounter any problems, refer to the previous lab for a description of the ARFF format, or simply look at some of the existing ARFF files in a text editor.

Once you have your data prepared, you can get WEKA to analyse it for you.

If you have created your own ARFF file, append it to the answer you typed in the box provided below.

Click the Task Complete button.

A.2 Lab Two

Data Mining with WEKA: Text classification

These labs provide sample exercises and a guide to materials that are relevant to the module ICP3037. The exercises are to help you become familiar with the material and help you with the assignment.

Lab work will be assessed for this module. It will count for 15% towards your final overall mark, and should be your own individual work. Some of the lab exercises may also appear as exam questions, so it is in your own best interests to complete the lab ….

For today's lab, write the answers to the exercises and include the requested screenshots in a Microsoft WORD document, archive in a zip file the exercise 3 files and email the lecturer (wjt@informatics.bangor.ac.uk).

Using WEKA as a text classifier.

Continuing from last week we will use WEKA to classify text.

Using your favourite Web browser, type in the following URL:

`http://yella.informatics.bangor.ac.uk:8080/`

Then enter your username as your first name, and last name. Then an "Enter task details" pop-up box appears. Then for each of the tasks below, type in the sub-task number into the "Task Name" field, plus a short description of the task details. (Note: if you have to revisit a task, please add details that you are changing your answer for this task).

(Or you can run WEKA directly by going to the weka-1 sub-directory and running the weka-3-4-6.exe to install the latest release of WEKA.)

The data we will be working with is from the Reuters news agency (http://www.reuters.com/).
The aim of this week's assignment is to classify news articles into 5 different categories:

1. corporate acquisitions
2. corn
3. crude oil
4. grain

The arff files for the Reuters data are located in the sub-directory Reuters on the course drive. Note that the arff files include the labels that Reuters personnel have manually assigned to each of the news articles (i.e. by hand, not by computer).

The arff files for this week's assignment are paired so for each category there are 2 files – one for training and one for testing. E.g.:

ReutersCorn-train.arff
ReutersCorn-test.arff

Have a look at the files to see what they look like. Note that under the "@data" field, all the text has been included as one large string for each of the articles that have been categorized into one or more of the five categories above. The purpose of this lab's experiments is to see how accurately the classifiers can assign the labels to the test data once they have been trained on the training data.

You may have heard of Bayesian filters being used in SPAM filters. WEKA has an implementation of naiveBayes, plus many other learning algorithms.

Your task for this lab is to discover which of several classifiers – NaiveBayes, SMO, IBk or J48 (one for each of the tasks in this lab) has the best performance at classification on the Reuters articles over all the categories above. Part of your job is also to find out which combination of WEKA options work best for each classifier.

(Note that a "greedy" search of the options is OK. You are not expected to search all combinations exhaustively! All of these classifiers will work with their respective default options. However, for SMO and IB*k*, you should try a few values of *k* to find the one that works best for each dataset.)

Task One (Using a Naive Bayes classifier).

Subtask One A

In WEKA, load the *ReutersCorn-train.arff* training file in the explorer. Switch to the classifier tab. To make things a little more difficult than last week, the data must first be preprocessed before the classifier can be applied. To do this you should choose weka.classifiers.meta.FilteredClassifier

DO NOT USE THE PREPROCESS TAB

After selecting the classifier, set the properties for the classifier. The properties for this classifier are as follows:

classifier classifiers.bayes.NiaveBayesMultinomial

filter filters.unsupervised.attribute.StringWordVector

No other classifier options need to be set.

In the Test options, set the supplied test set to *ReutersCorn-test.arff*

You should find that 94.0397% of the test data is correctly classified.

Subtask One B

Repeat task one with the corporate acquisitions data.

Subtask One C

Repeat task one with the crude oil data.

Subtask One D

Repeat task one with the grain data.

Repeat sub-tasks A through E by trying out different options.

Exercise 1 (8 marks): Enter your results here.

Task two (Using the support vector machine classifier).

Select the support vector machine classifier:

 `weka.classifiers.functions.SMO`

Repeat sub-tasks A through E as for Task one. Try out different options.

Exercise 2 (8 marks): Enter your results here.

Task three (Using the nearest neighbour classifier).

Select the nearest-neighbour classifier:

 `weka.classifiers.lazy.IBk`

Repeat sub-tasks A through E as for Task one. Try out different options.

Exercise 3 (8 marks): Enter your results here.

Task four (Using the decision tree classifier).

Select the decision tree classifier :

 `weka.classifiers.trees.J48`

Repeat sub-tasks A through E as for Task one. Try out different options.

Exercise 4 (8 marks): Enter your results here.

Exercise 5 (8 marks): Enter a summary of your overall results here. What did you find out?

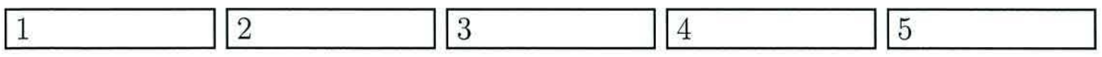
Appendix B

Cross Validation Technique

This appendix details the cross validation technique used in chapter 9. The typical method of cross validation is to split the data set into n slices. If each data point is independent i.e. the order of the data is unimportant, then the position of the split is unimportant. When performing cross validation on text streams, the order of the characters in the stream is important. If a corpus contains many documents then the cross validation can take place at the level of individual documents.

If this is not possible, i.e. when there is only a single document, then a different approach must be taken. The first (and most commonly used) approach is to split the file into n subsections. This is shown in figure B, where the stream is split into five sections labelled one to five. One section is reserved for testing and the remaining four sections are used for training. After testing the first section the split is cycled through so all sections are used for testing once.

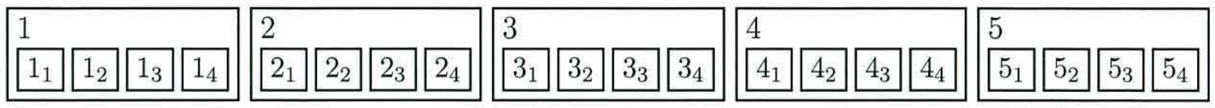
Figure B.1: Cross Validation Size Five



The User Interaction Streams have Action Events distributed in a non-linear way. The Action Events are clustered together. This can be seen in section 9.1.1 where new Action Events are generated throughout the User Interaction Stream. To overcome this an alternate cross validation technique was developed.

The alternate technique splits the stream into n sections. Each of these n sections is split again m times to form $n * m$ subsections. Each subsection is labelled in the form n_m. This is shown in figure B.2.

Figure B.2: Multi Cross Validation Size Five



After perfoming the split a testing document is produced by assembiling all the sub splits with the same number i.e. In figure B.2 testing document one would be made up of 1_1, 2_1, 3_1, 4_1 and 5_1. The unused subsections form the training document. After testing with split N_1 the testing is repeated for each subsection. The next sections to be used for testing in this example would be: 1_2, 2_2, 3_2, 4_2 and 5_2

Bibliography

3GPP (2007), Short Message Service Cell Broadcast (SMSCB) support on the mobile radio interface, TS 44.012, 3rd Generation Parnership Proejct (3GPP).
URL: `http://www.3gpp.org/ftp/Specs/html-info/44012.htm`

50000 euro Prize for Compressing Human Knowledge (2007).
URL: `http://prize.hutter1.net/`

ACM's Special Interest Group on Computer-Human Interaction (2008).
URL: `http://www.sigchi.org/`

Adams, D. (1995), *The Hitchhiker's Guide to the Galaxy*, Pan Books.

Adobe (2008), 'Adobe flash player'.
URL: `http://www.adobe.com/products/flashplayer/`

Alata, E., Nicomette, V., Kaaniche, M., Dacier, M. & Herrb, M. (2006), 'Lessons learned from the deployment of a high-interaction honeypot', *Dependable Computing Conference, 2006. EDCC '06. Sixth European* pp. 39–46.

Apple (2004), 'Apple human interface guidelines'.
URL: `http://developer.apple.com/documentation/UserExperience/Conceptual/AppleHIGuidelines/`

Apple (2006), 'Quicktime framework reference'.
URL: `http://developer.apple.com/documentation/QuickTime/Reference/QT_Framework_Ref/QT_Framework_Ref.pdf`

Apple (2008), 'Instruments user guide'.

Association for Computing Machinery (1947).

Ayewah, N., Pugh, W., Morgenthaler, J. D., Penix, J. & Zhou, Y. (2007), Evaluating static analysis defect warnings on production software, *in* 'PASTE '07: Proceedings of the 7th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering', ACM, New York, NY, USA, pp. 1–8.

Bangor User Interaction Stream Corpus (2008).
URL: `http://aiia.cs.bangor.ac.uk/PhDs/David_Hunnisett`

Bates, D. W., Leape, M. L. L., Cullen, M. D. J., Laird, M. N., Petersen, M. L. A., Teich, M. J. M., MD, Burdick, P. E., Hickey, M. M., Kleefield, M. S., Shea, M. B., Vliet, M. M. V. & Seger, R. D. L. (1998), 'Effect of computerized physician order entry and a team intervention on prevention of serious medication errors', *Journal of the American Medical Association* **280**(15).

Bell, T., Witten, I. H. & Cleary, J. G. (1989), 'Modeling for text compression', *ACM Computing Surveys* **21**(4), 557–591.

Benedetto, D., Caglioti, E. & Loreto, V. (2002), 'Language trees and zipping', *Physical Review Letters* (88), 2002.

Better Desktop Data (2008).
URL: `http://www.betterdesktop.org/wiki/index.php?title=Data`

Borodovsky, M. & McIninch, J. (1993), 'Genmark: parallel gene recognition for both dna strands', *Comp. Chem* (17), 123–133.

Bratko, A. & Filipič, B. (2005), Spam filtering using compression models, Technical report.

Bratko, A., Filipič, B., Cormack, G. V., Lynam, T. R. & Zupan, B. (2006), 'Spam filtering using statistical data compression models', *Journal of Machine Learning Research* **7**, 2673–2698.

Brooks, R., Hunnisett, D. & Teahan, W. J. (2007), 'A practical implementation of automatic text categorisation and correction for the conversion of noisy ocr documents into braille and large print'.

Burge, C. & Karlin, S. (1997), 'Prediction of complete gene structures in human genomic dna.', *J Mol Biol* **268**(1), 78–94.
URL: `http://dx.doi.org/10.1006/jmbi.1997.0951`

CamStudio (2007).
URL: `http://camstudio.org/`

Carroll, L. (1865), *Alice's Adventures in Wonderland*, Project Gutenberg Literary Archive Foundation.

Cavnar, W. B. & Trenkle, J. M. (1994), N-gram-based text categorization, *in* 'Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval', Las Vegas, US, pp. 161–175.

Chen, K., Gao, S., Zhu, Y. & Sun, Q. (Oct. 2006), 'Music genres classification using text categorization method', *Multimedia Signal Processing, 2006 IEEE 8th Workshop on* pp. 221–224.

Cleary, J., Teahan, W. & Witten, I. (1995), 'Unbounded length contexts for ppm'.

Cleary, J. & Witten, I. (1984), 'Data compression using adaptive coding and partial string matching', *Communications, IEEE Transactions on [legacy, pre - 1988]* **32**(4), 396–402.

Copeland, T. (2005), *PMD Applied*, Centennial Books.

Corney, M., de Vel, O., Anderson, A. & Mohay, G. (2002), Gender-preferential text mining of e-mail discourse, *in* 'ACSAC '02: Proceedings of the 18th Annual Computer Security Applications Conference', IEEE Computer Society, Washington, DC, USA, p. 282.

Coull, S., Branch, J., Szymanski, B. & Breimer, E. (2003), Intrusion detection: A bioinformatics approach, *in* 'ACSAC '03: Proceedings of the 19th Annual Computer Security Applications Conference', IEEE Computer Society, Washington, DC, USA, p. 24.

Darragh, J., Witten, I. & James, M. (1990), 'The reactive keyboard: a predictive typing aid', *Computer* **23**(11), 41–49.

Dengo (2006).
URL: `http://sourceforge.net/projects/dengo/`

Deutsch, P. (1996), GZIP file format specification version 4.3, RFC 1952, Internet Engineering Task Force.
URL: `http://www.rfc-editor.org/rfc/rfc1952.txt`

Deutsch, P. & l. Gailly, J. (1996), ZLIB compressed data format specification version 3.3, RFC 1950, Internet Engineering Task Force.
URL: `http://www.rfc-editor.org/rfc/rfc1950.txt`

Eclipse Foundation (2008*a*), 'eclipse'.
URL: `http://www.eclipse.org`

Eclipse Foundation (2008*b*), 'Swt: The standard widget toolkit'.
URL: `http://www.eclipse.org/swt/`

Eye Toy (2008).
URL: `http://www.eyetoy.com/index.asp`

Fenwick, P. (1996), 'The Burrows-Wheeler transform for block sorting text compression – principles and improvements'.

Ferizis, G. & Bailey, P. (2006), Towards practical genre classification of web documents, *in* 'WWW '06: Proceedings of the 15th international conference on World Wide Web', ACM, New York, NY, USA, pp. 1013–1014.

Fisher, P. & Sless, D. (1990), 'Information design methods and productivity in the insurance industry.', *Information Design Journal* **6**, 103–129.

Fitts, P. M. (1954), 'The information capacity of the human motor system in controlling the amplitude of movement.', *Journal of Experimental Psychology* **47**(6), 381–391.

Fitts, P. M. & Peterson, J. R. (1964), 'Information capacity of discrete motor responses', *Journal of Experimental Psychology* **67**, 103–112.

Flanagan, D. (1998), *Java Foundation Classes in a Nutshell*, O'Reilly & Associates, Inc., Sebastopol, CA, USA.

Forrester, J. E. & Miller, B. P. (2000), An empirical study of the robustness of windows nt applications using random testing, *in* 'WSS'00: Proceedings of the 4th conference on USENIX Windows Systems Symposium', USENIX Association, Berkeley, CA, USA, pp. 6–6.

Free Software Foundation, I. (2007), 'GNU general public license'.

Freez Screen Video Capture (2007).
URL: `http://www.smallvideosoft.com/screen-video-capture`

FSF (2006), 'Bash'.

Gailly, J. & Adler, M. (2008), 'The gzip algorithm'.
URL: `http://www.gzip.org/algorithm.txt`

Gale, W. A., Church, K. W. & Yarowsky, D. (1992), 'A method for disambiguating word senses in a large corpus', *Computers and the Humanities* **26**(5), 415–439.
URL: `http://dx.doi.org/10.1007/BF00136984`

Garg, A., Rahalkar, R., Upadhyaya, S. & Kwiat, K. (2006), 'Profiling users in gui based systems for masquerade detection', *Information Assurance Workshop, 2006 IEEE* pp. 48–54.

GENSCAN (1997).
URL: `http://genes.mit.edu/GENSCAN.html`

Gibson, J. J. (1979), *The Ecological Approach to Visual Perception*, Lawrence Erlbaum Associates.

GNOME (2008), 'Gnome human interface guidelines'.
URL: `http://developer.gnome.org/projects/gup/hig/`

GNOME Usability Analysis Tool (2006).
URL: `http://sourceforge.net/projects/guat`

Graham, P. (2004), *Hackers and Painters: Big Ideas from the Computer Age*, O'Reilly Media, Inc.
URL: `http://www.paulgraham.com/spam.html`

Greenberg, S. & Witten, I. H. (1988), How users repeat their actions on computers: principles for design of history mechanisms, *in* 'CHI '88: Proceedings of the SIGCHI conference on Human factors in computing systems', ACM, New York, NY, USA, pp. 171–178.

Grünwald, P. D. (2007), *The Minimum Description Length Principle*, The MIT Press.

Gutenberg, P. (1992), 'Project gutenberg'.

Higgins, D., Burstein, J. & Attali, Y. (2006), 'Identifying off-topic student essays without topic-specific training data', *Nat. Lang. Eng.* **12**(2), 145–159.

HippieExpand (2004).
URL: `http://www.emacswiki.org/cgi-bin/wiki/HippieExpand`

Hoad, T. C. & Zobel, J. (2003), 'Methods for identifying versioned and plagiarized documents', *J. Am. Soc. Inf. Sci. Technol.* **54**(3), 203–215.

Holmes, D. I. (1998), 'The evolution of stylometry in humanities scholarship', *Literary and Linguistic Computing* (13), 111–117.

Hornof, A. J. & Kieras, D. E. (1997), Cognitive modeling reveals menu search in both random and systematic, *in* 'CHI '97: Proceedings of the SIGCHI conference on Human factors in computing systems', ACM, New York, NY, USA, pp. 107–114.

Huang, R. & Hansen, J. (15-20 April 2007), 'Dialect classification on printed text using perplexity measure and conditional random fields', *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on* **4**, IV–993–IV–996.

Human Factors Society (1984), number New Frontiers for Science and Technology.

Hunnisett, D. & Teahan, W. (2004), Context-based methods for text categorisation, *in* M. Sanderson, K. Järvelin, J. Allan & P. Bruza, eds, 'SIGIR 2004: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval', ACM.

Imsand, E. & Hamilton, J. (2007), 'Gui usage analysis for masquerade detection', *Information Assurance and Security Workshop, 2007. IAW '07. IEEE SMC* pp. 270–276.

International Organization for Standardization (2005), *Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s - Part 3: Audio*, International standard; ISO 11172-3, 1 edn, International Organization for Standardization, Geneva, Switzerland.

International Organization for Standardization (2007), *Information technology - Coding of audio-visual objects - Part 10: Advanced Video Coding*, ISO/IEC 14496-10:2005, 1 edn, International Organization for Standardization, Geneva, Switzerland.

Jacob, R. J. (1983*a*), Executable specifications for a human-computer interface, *in* 'CHI '83: Proceedings of the SIGCHI conference on Human Factors in Computing Systems', ACM, New York, NY, USA, pp. 28–34.

Jacob, R. J. K. (1983*b*), 'Using formal specifications in the design of a human-computer interface', *Communications of the ACM* **26**, 259–264.

James, C. & Longé, M. (2000), Bringing text input beyond the desktop, *in* 'CHI '00: CHI '00 extended abstracts on Human factors in computing systems', ACM, New York, NY, USA, pp. 49–50.

JavaCC (2008).
URL: `https://javacc.dev.java.net/`

Java Event Queue API (2004).
URL: `http://java.sun.com/j2se/1.5.0/docs/api/java/awt/EventQueue.html`

Java Web Start Technology (2008).
URL: `https://java.sun.com/javase/technologies/desktop/javawebstart/index.jsp`

Jimison, H., Pavel, M., McKanna, J. & Pavel, J. (2004), 'Unobtrusive monitoring of computer interactions to detect cognitive status in elders', *Information Technology in Biomedicine, IEEE Transactions on* **8**(3), 248–252.

JPEG File Interchange Format (1992).
URL: `http://www.w3.org/Graphics/JPEG/jfif3.pdf`

Karwowski, W. (2006), *International Encyclopedia of Ergonomics and Human Factors, Second Edition - 3 Volume Set*, CRC Press, Inc., Boca Raton, FL, USA, p. 610.

Kessler, B., Numberg, G. & Schütze, H. (1997), Automatic detection of text genre, *in* 'Proceedings of the 35th annual meeting on Association for Computational Linguistics', Association for Computational Linguistics, Morristown, NJ, USA, pp. 32–38.

Khmelev, D. V. & Teahan, W. J. (2003), A repetition based measure for verification of text collections and for text categorization, *in* 'SIGIR '03: Proceedings of the

26th annual international ACM SIGIR conference on Research and development in informaion retrieval', ACM, New York, NY, USA, pp. 104–110.

Kim, H. (2005), Developing semantic digital libraries using data mining techniques, PhD thesis, Gainesville, FL, USA. Chair-Su-Shing Chen.

Kim, J. Y., Allen, J. P. & Lee, E. (2008), 'Alternate reality gaming', *Commun. ACM* **51**(2), 36–42.

Knuth, D. (1997), *The Art of Computer Programming, Sorting and Searching*, Vol. 3, Addison-Wesley, chapter 6, pp. 492–512.

Koppel, M., Argamon, S. & Shimoni, A. R. (2003), 'Automatically categorizing written texts by author gender', *Literary and Linguistic Computing* (17), 401–412.

Koppel, R., Metlay, J. P., Cohen, A., Abaluck, B., Localio, A. R., Kimmel, S. E. & Strom, B. L. (2005), 'Role of computerized physician order entry systems in facilitating medication errors', *J. Am. Med. Assoc.* **293**(10), 1197–1203.
URL: `http://dx.doi.org/10.1001%2Fjama.293.10.1197`

Lab Resources (2004).
URL: `http://cscl.ist.psu.edu/public/resources.html`

Larkey, L. S. (1998), Automatic essay grading using text categorization techniques, *in* 'SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval', ACM, New York, NY, USA, pp. 90–95.

Lee, Y.-B. & Myaeng, S. H. (2002), Text genre classification with genre-revealing and subject-revealing features, *in* 'SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval', ACM, New York, NY, USA, pp. 145–150.

Leggett, J. & Williams, G. (1988), 'Verifying identity via keystroke characteristics', *Int. J. Man-Mach. Stud.* **28**(1), 67–76.

Leung, C.-H. & Chan, Y.-Y. (2007), A natural language processing approach to automatic plagiarism detection, *in* 'SIGITE '07: Proceedings of the 8th ACM SIGITE conference on Information technology education', ACM, New York, NY, USA, pp. 213–218.

Lewis, K. (2000), *Creating effective JavaHelp*, O'Reilly & Associates, Inc., Sebastopol, CA, USA.

Li, W. (1992), 'Random texts exhibit zipf's-law-like word frequency distribution', *Information Theory, IEEE Transactions on* **38**(6), 1842–1845.

Linetsky, M. (2001), *Programming Microsoft Directshow*, Wordware Publishing Inc., Plano, TX, USA.

Lukashenko, R., Graudina, V. & Grundspenkis, J. (2007), Computer-based plagiarism detection methods and tools: an overview, *in* 'CompSysTech '07: Proceedings of the 2007 international conference on Computer systems and technologies', ACM, New York, NY, USA, pp. 1–6.

Luo, X. & Zincir-Heywood, A. N. (2005), *Evaluation of Two Systems on Multi-class Multi-label Document Classification*, Vol. 3488/2005 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg.

Mantei, M. M. und Teorey, T. J. (1988), 'Cost-benefit analysis for intercorporating human factors in the software lifecycle'.

Manuals, G. S. (1935), 'H. j. m. milne', *The Journal of Hellenic Studies* **55**, 252–253.

Maron, M. E. (1961), 'Automatic indexing: An experimental inquiry', *J. ACM* **8**(3), 404–417.

Marton, Y., Wu, N. & Hellerstein, L. (2005), On compression-based text classification, *in* 'ECIR', pp. 300–314.

Massie, T. H. & Salisbury, K. J. (1994), Phantom haptic interface: a device for probing virtual objects, Vol. 55-1 of *Proceedings of the 1994 International Mechanical*

Engineering Congress and Exposition, ASME, Massachusetts Inst of Technology, Cambridge, United States, pp. 295–299.

Matthews, R. A. J. & Merriam, T. V. N. (1964), 'Neural computation in stylometry i: An application to the works of shakespeare and fletcher', *Literary and Linguistic Computing* .

McCallum, A. & Nigam, K. (1998), 'A comparison of event models for naive bayes text classification'.
URL: `http://lans.ece.utexas.edu/ulg/papers/nigam-mccallum-bayes.pdf.gz`

Microsoft (2005), 'Top rules for the windows vista user experience'.
URL: `http://msdn2.microsoft.com/en-us/library/aa511327.aspx`

Miller, B. P., Cooksey, G. & Moore, F. (2007), 'An empirical study of the robustness of macos applications using random testing', *SIGOPS Oper. Syst. Rev.* **41**(1), 78–86.

Miller, B. P., Fredriksen, L. & So, B. (1990), 'An empirical study of the reliability of unix utilities', *Commun. ACM* **33**(12), 32–44.

Moffat, A. (1990), 'Implementing the ppm data compression scheme', *Communications, IEEE Transactions on* **38**(11), 1917–1921.

Monrose, F. & Rubin, A. (1997), Authentication via keystroke dynamics, *in* 'CCS '97: Proceedings of the 4th ACM conference on Computer and communications security', ACM, New York, NY, USA, pp. 48–56.

Mosteller, F. & Wallace, D. L. (1966), 'Inference and disputed authorship: The federalist', *Review of the International Statistical Institute* **34**(2), 277–279.

Netbeans (2008), 'Jemmy module, a java ui testing library'.
URL: `http://jemmy.netbeans.org/`

Nielsen, J. (1993), 'Noncommand user interfaces', *Commun. ACM* **36**(4), 83–99.

Nielsen, J. (1994*a*), 'Guerrilla hci: using discount usability engineering to penetrate the intimidation barrier', pp. 245–272.

Nielsen, J. (1994*b*), *Usability Engineering*, Elsevier Science Ltd.

Norman, D. A. (1988), *The Psychology of Everyday Things*, Basic Books.

OpenOffice.org 2.x Writer Guide, Using word completion (2008).
URL: `http://wiki.services.openoffice.org/wiki/Documentation/OOoAuthors_User_Manual/Writer_Guide/Using_word_completion`

Pang, B. & Lee, L. (2004), A sentimental education: sentiment analysis using subjectivity summarization based on minimum cuts, *in* 'ACL '04: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics', Association for Computational Linguistics, Morristown, NJ, USA, p. 271.

Pang, B. & Lee, L. (2005), Seeing stars: exploiting class relationships for sentiment categorization with respect to rating scales, *in* 'ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics', Association for Computational Linguistics, Morristown, NJ, USA, pp. 115–124.

Pang, B. & Lee, L. (2008), 'Opinion mining and sentiment analysis', *Foundations and Trends in Information Retrieval* **2**(1-2), 1–135.

PostgreSQL Global Development Group (2003), 'Postgresql database'.
URL: `http://www.postgresql.org/`

Pusara, M. & Brodley, C. E. (2004), User re-authentication via mouse movements, *in* 'VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security', ACM, New York, NY, USA, pp. 1–8.

Ratushnyak, A. (2007), 'The paq8hp12 compressor'.
URL: `http://www.cs.fit.edu/~mmahoney/compression/`

REUTERS CORPUS (2000).
URL: `http://about.reuters.com/researchandstandards/corpus/`

Rey, J. & Reynar, C. (1998), Topic segmentation: Algorithms and applications, Technical Report IRCS-98-21, University of Pennsylvania Institute for Research in Cognitive Science.

Rosé, C. P., Roque, A., Bhembe, D. & Vanlehn, K. (2003), A hybrid text classification approach for analysis of student essays, *in* 'Proceedings of the HLT-NAACL 03 workshop on Building educational applications using natural language processing', Association for Computational Linguistics, Morristown, NJ, USA, pp. 68–75.

Ross, N. E. (1928), 'How to write telegrams properly'.
URL: `http://www.telegraph-office.com/pages/telegram.html`

Scheurer, C. (2005), 'Unique rar file library faq'.
URL: `http://www.unrarlib.org/faq.html`

Seo, J. & Cha, S. (2007), Masquerade detection based on svm and sequence-based user commands profile, *in* 'ASIACCS '07: Proceedings of the 2nd ACM symposium on Information, computer and communications security', ACM, New York, NY, USA, pp. 398–400.

Shackel, B. & Richardson, S. J., eds (1991), *Human factors for informatics usability*, Cambridge University Press, New York, NY, USA.

Shannon, C. E. (1948), 'A mathematical theory of communication', *Bell System Tech. J.* **27**, 379–423, 623–656.

Skibiński, P., Grabowski, S. & Deorowicz, S. (2005), 'Revisiting dictionary-based compression: Research articles', *Softw. Pract. Exper.* **35**(15), 1455–1476.

SMITH, Edward TAYLOR, R. (1995), 'Cockpit conversion from three to two members', PCT/US1994/010689.

Soukoreff, R. W. & MacKenzie, I. S. (2004), 'Towards a standard for pointing device evaluation, perspectives on 27 years of fitts' law research in hci', *Int. J. Hum.-Comput. Stud.* **61**(6), 751–789.

SPAM Track Guidelines (2005 - 2007).
URL: `http://plg.uwaterloo.ca/~gvcormac/spam/`

Special Interest Group on Information Retrieval (2008).
URL: `http://www.sigir.org/`

Stamatatos, E. (2008), 'Author identification: Using text sampling to handle the class imbalance problem', *Inf. Process. Manage.* **44**(2), 790–799.

Stig Johansson, GEOFFREY N. LEECH, H. G. (1978), 'The lancaster-oslo/bergen corpus'.

Strobl, R. (2008), 'Netbeans code completion'.
URL: `http://editor.netbeans.org/project/editor/doc/UserView/completion.html`

SUN (2006), 'Java heap analysis tool'.
URL: `http://java.sun.com/javase/6/docs/technotes/tools/share/jhat.html`

Teahan, W. J. (1998), Modelling English Text, PhD thesis, University of Waikato.

Teahan, W. J. & Harper, D. J. (2001), Using compression-based language models for text categorization, *in* 'Workshop on Language Modeling and Information Retrieval'.

Teahan, W. J. & Harper, D. J. (2003*a*), 'Using compression based language models for text categorization'.

Teahan, W. J. & Harper, D. J. (2003*b*), Using compression-based language models for text categorization, *in* W. B. Croft & J. Lafferty, eds, 'Language Modelling for Information Retrieval', Kluwer Academic Publishers, chapter 7, pp. 141–65.

Teahan, W. J., Thomas, D. & Hunnisett, D. (2009), Protocols for stream-based text categorization. Submitted to ECIR09.

Text Minning Toolkit (2008).
URL: `http://aiia.cs.bangor.ac.uk/TMT/TMT-0.08.tgz`

The Better Desktop Project (2006).
URL: `http://www.betterdesktop.org`

Theofanos, M. F. & Redish, J. G. (2003), 'Bridging the gap: between accessibility and usability', *interactions* **10**(6), 36–51.

The usability lab photo gallery (2008).
URL: `http://www.noldus.com/site/doc200406061`

The Xito platform (2005).
URL: `http://xito.sourceforge.net/`

Tjalkens, T., Volf, P. & Willems, F. (1997), 'A context-tree weighting method for text generating sources', *Data Compression Conference, 1997. DCC '97. Proceedings* pp. 472–.

Volf, P. A. (2002), Weighting Techniques in Data Compression Theory and Algorithms, PhD thesis, Technische Universiteit Eindhoven.

Wang, T. & Desai, B. C. (2007), An approach for text categorization in digital library, *in* 'IDEAS '07: Proceedings of the 11th International Database Engineering and Applications Symposium', IEEE Computer Society, Washington, DC, USA, pp. 21–27.

Warrender, C., Forrest, S. & Pearlmutter, B. (1999), Detecting intrusions using system calls: Alternative data models, *in* 'In IEEE Symposium on Security and Privacy', IEEE Computer Society, pp. 133–145.

Welch, T. A. (1984), A technique for high-performance data compression, *in* 'Computer', Vol. 17, pp. 8–19.

Willems, F. M. J., Shtarkov, Y. M. & Tjalkens, T. J. (1995), 'The context-tree weighting method: basic properties.', *IEEE Trans. Info. Theory* pp. 653–664.

Witten, I. H. (1982), An interactive computer terminal interface which predicts user entries, *in* 'Man-machine Interaction', IEE Conference on Man-machine Interaction, Manchester, England.

Witten, I. H. & Frank, E. (2005), *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*, Morgan Kaufmann.

Wu, P. & Teahan, W. J. (2005), Modelling chinese for text compression, *in* 'DCC '05: Proceedings of the Data Compression Conference', IEEE Computer Society, Washington, DC, USA, pp. 488–488.

Xerox Alto (1972).

Xerox Palo Alto Research Center (1999).

Yang, Y. & Liu, X. (1999), A re-examination of text categorization methods, *in* 'SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval', ACM, New York, NY, USA, pp. 42–49.

Zipf, G. K. (1968), *The psycho-biology of language;: An introduction to dynamic philology*, M.I.T. Press.