Using Data Mining for Digital Ink Recognition

Rachel Venita Blagojevic

A thesis submitted in fulfilment of the requirements for the degree of PhD in Computer Science,
The University of Auckland, 2011.

ABSTRACT

Computational recognition of hand-drawn diagrams has come a long way but is still inadequate for general use. This research uses data mining techniques to improve the accuracy of recognition. We focus on text-shape division as a challenging example that benefits from this approach. Surprisingly, although text is a fundamental part of diagrams it has been largely ignored.

A review of the literature will show that feature-based recognisers are ideal candidates for solving these types of problems. Such recognisers require a good feature set and a suitable algorithm. For recognition to be successful, the features fed into the algorithms must provide good distinguishing characteristics between classes of interest. While small feature sets have been reported, currently there is no extensive survey of existing features employed for sketch recognition. Such a survey could act as a library for algorithms to employ for a given problem in sketch recognition. In addition, while various algorithms have been tried, there has been no extensive study of algorithms to determine the most optimal fit for accurate text-shape dividers.

To build our text-shape dividers, we have assembled a comprehensive library of ink features that can be used for sketch recognition problems and compiled a large repository of labelled sketch data. To collect this data we built our own tool, DataManager, which includes support for collecting and labelling sketches as well as automatically generating datasets. Using this feature library and data repository a systematic investigation and tuning of machine learning algorithms has identified the algorithms best suited to text-shape division. The extensive evaluation on diagrams from six different domains has shown that our resulting dividers, using LADTree and LogitBoost, are significantly more accurate than three existing dividers. To our knowledge, these algorithms have not been used for text-shape division before.

For Marko, Mum and Dad

Acknowledgements

First and foremost I want to thank my main supervisor Dr. Beryl Plimmer. I have always counted myself very blessed to have you as a supervisor. You have so much time for your students and although we may not always show it, we appreciate your guidance and support very much. Thank you.

To my supervisors, Prof. John Grundy and Dr. Yong Wang, thank you for always being so willing to help. John, your input has been invaluable. Yong, thanks for always having an open door and having the patience to explain...again.

I am indebted to Associate Prof. Eibe Frank. Your advice and guidance on the use of data mining and Weka has been invaluable.

Of course I can't forget the HCI research group. You guys have made the last three and a half years so memorable. Thank you all for your support, advice and friendship. Special thanks to Paul and Samuel for understanding what I'm talking about (most of the time). Also thanks to Laura and Samuel for collecting and labelling data, Paul for writing converters and all of you for your work on DataManager.

Special thanks to Yuriy Halytskyy for setting me up to use the Auckland Cluster and providing technical support.

Thanks to Gregory Rowe for proof reading this thesis.

I am very appreciative of the Tertiary Education Commission for awarding me with the Top Achiever Doctoral Scholarship to support this work.

Thanks to my awesome friends and family, I would have gone insane without you!

To Marko, Mum and Dad, yes it is finally finished, I am done being a student! As much as this thesis is mine, it is yours also.

Marko, for the past three and a half years you have put me before yourself, for this I can never thank you enough. Your patience, love and understanding know no bounds and your ability to put things into perspective kept me going.

To my Lord and Saviour, Jesus Christ, whose plans are perfect and love is unfailing.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This research investigates the use of data mining to improve sketch recognition accuracy. Sketch recognition is the automatic identification of hand-drawn digital elements. Automatic recognition of the components in digital sketches can offer several benefits. We focus on text-shape division for sketched diagrams as a challenging example that benefits from this approach. With accurate division of text and shapes in diagrams, each group can be sent to separate handwriting and shape recognisers to continue the recognition process.

The field of sketch recognition includes, but is not limited to, character recognition. Johnson et al (2009b) describe the kinds of elements we are interested in recognising as geometric shapes, spatial features, characters, entities, artistic nuance, commands, specific sketch genre and intention. Character recognition is a much more mature field than other areas of digital ink recognition. Therefore, we are interested in advancing the recognition of the remaining areas and in particular, recognition of hand-drawn diagrams.

Sketch recognition is often referred to as "online" recognition as dynamic information is available. Recognition of scanned images such as in optical character recognition (OCR) is referred to as "offline" recognition as it deals with static information. Sketches in online recognition are composed of strokes, where a stroke is a collection of points drawn from pen-down to pen-up. A stroke contains information on the (x, y) coordinates of each point in the stroke as well as a time stamp and pressure value for these points. This information differs significantly from the bitmap representations from static scanned images or documents available as inputs to image recognition and OCR.

With recognition algorithms used to understand sketches, computational support can offer automatic translation between sketch and formal diagram and an executable sketch model. For example, the automatic translation shown in Figure 1, between a unified modelling language (UML) class diagram and code stubs, can begin with a user sketching a UML class diagram rather than constructing a diagram using widget based tools. With automatic recognition of the user's initial sketch the UML

class diagram can be translated to the formal diagram and code stubs shown in Figure 1. An example of automatic translation to formal diagram and executable model is shown in Figure 2. This shows a node and edge graph drawn using the graph drawing tool, SketchNode (Plimmer et al. 2010). SketchNode is able to recognise hand-drawn graphs and render a beautified version of that graph.

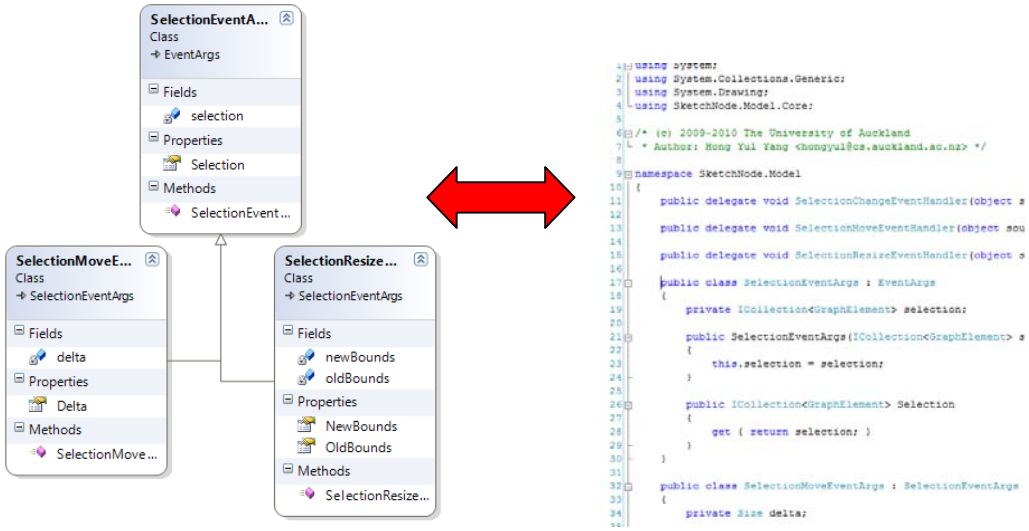


Figure 1 Automatic Translation of a UML Class Diagram to Code Stubs in Microsoft Visual Studio and Vice Versa (Example from SketchNode (Plimmer et al. 2010))

In addition, the information gained from automatic recognition of sketches allows for intelligent editing functions. For example, Figure 3 shows intelligent editing that is possible within SketchNode where node ‘c’ is moved and the connections of that node to other nodes automatically move with it. In addition to intelligent editing by the user, automatic editing can also be applied once information on diagram components has been obtained via sketch recognition algorithms. An example of this is shown in Figure 4 where automatic layout algorithms are applied to a hand-drawn graph in SketchNode to improve graph layout.

All of these advanced functions rely on the ability to automatically recognise the components drawn in diagrams accurately. SketchNode is able to achieve 100% recognition of text as all text is contained inside nodes. The recognition rates for distinguishing nodes from connectors are also high. Therefore, these advanced functions of intelligent editing, beautification and the application of automatic layout algorithms can be successfully employed in this tool. However, for more complex domains, recognition is far more challenging.

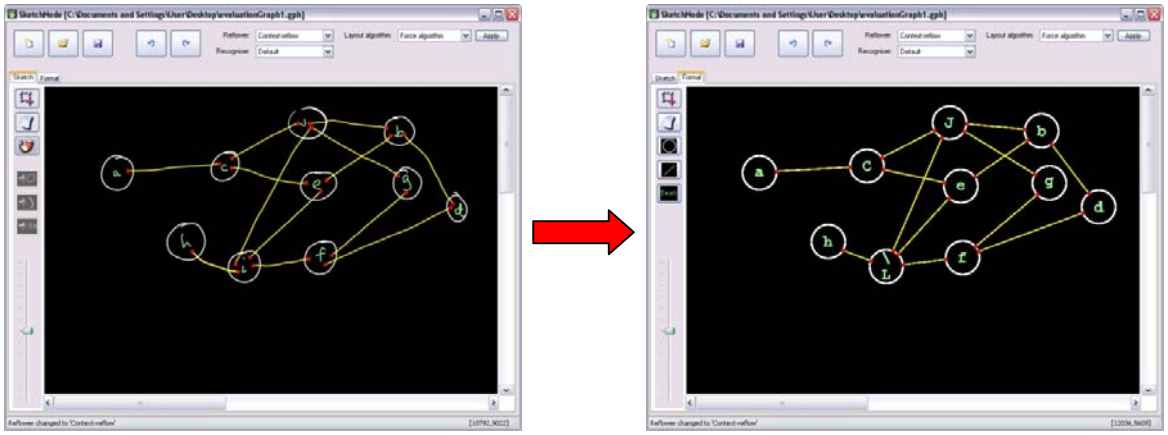


Figure 2 Automatic Translation of a Sketched Node and Edge Graph to a Formal Diagram (Plimmer et al. 2010)[1].

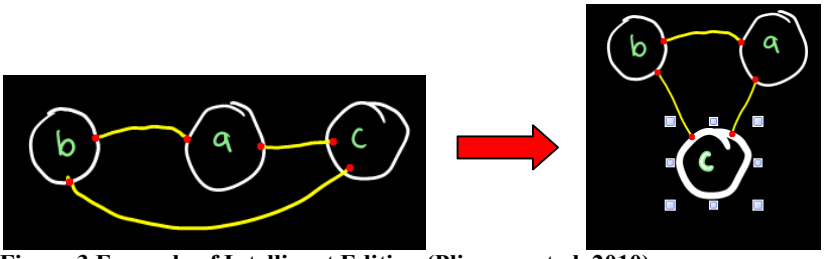


Figure 3 Example of Intelligent Editing (Plimmer et al. 2010)

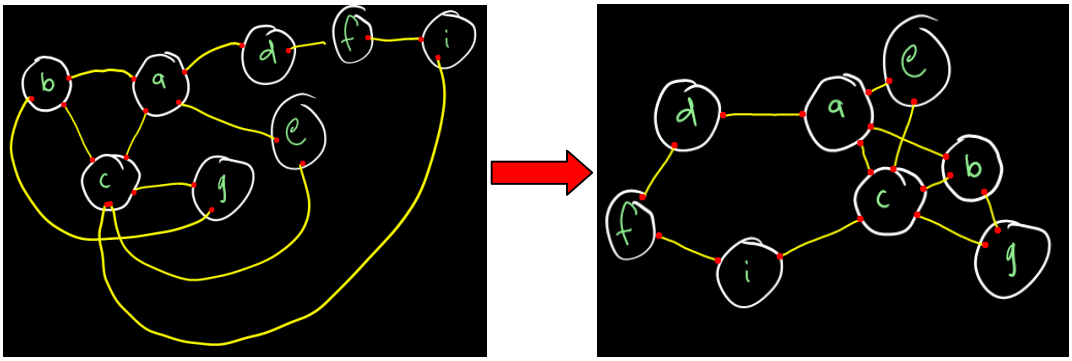
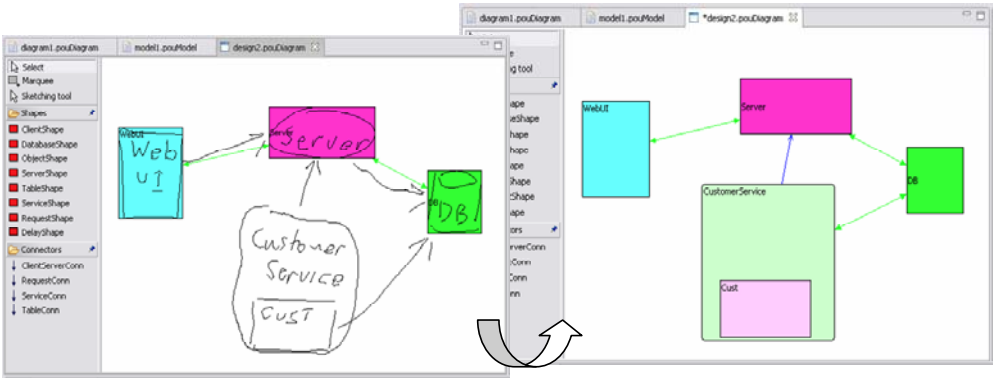


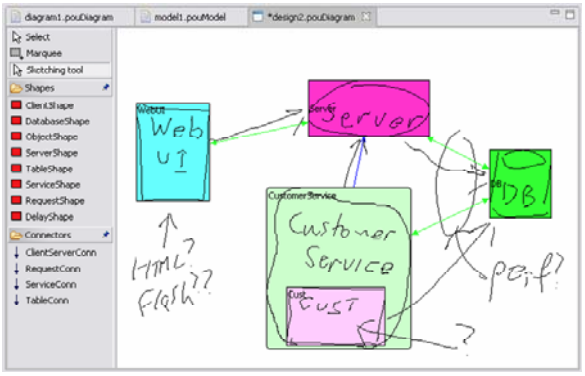
Figure 4 Example of Automatic Layout Algorithms Applied to Sketched Graphs (Plimmer et al. 2010).

Sketched content and formalised content are not necessarily incompatible in a tool. Figure 5 shows screen dumps from MaramaSketch (Grundy et al. 2007), where an architecture design diagram has been sketched, recognised, and the sketch and formalised diagram components are preserved concurrently. Additional sketch content can be more formal diagram components or informal annotations to support collaborative discussion.

[1]Image obtained from Plimmer, B., H. Purchase, et al. (2010). SketchNode: Intelligent sketching support and formal diagramming. OZCHI 2010. Brisbane, ACM: 136-143. Reproduced with permission from the author.



a) Software Architecture Sketch b) Recognised and Formalised Components



c) Concurrent use of Sketch Input and Formalised Components

Figure 5 MaramaSketch (Grundy et al. 2007)[2].

A number of sketch recognition tools and techniques have been developed (Lin et al. 2000; Fonseca et al. 2002; Hammond et al. 2002; Yu et al. 2003; Plimmer 2004; Grundy et al. 2007; Plimmer et al. 2007; Wobbrock et al. 2007; Chen et al. 2008; Paulson et al. 2008a). However they have yet to gain general acceptance. One of the unresolved challenges is to achieve considerably more accurate recognition than is currently available.

The ambiguity of hand-drawn diagrams makes recognition problems hard to solve. Recognition rates from laboratory experiments are typically in the range of 98% to 99% and above (Rubine 1991; Wobbrock et al. 2007; Paulson et al. 2008a). However, rates achieved in less controlled conditions, where data is not limited to produce optimal performance, are usually much lower. For example accuracy rates between 84% and 93% are reported in (Plimmer 2004; Young 2005; Wobbrock et al. 2007; Schmieder et al. 2009).

[2]Image obtained from Grundy, J. C. and J. G. Hosking (2007). Supporting generic sketching-based input of diagrams in a domain-specific visual language meta-tool. IEEE/ACM International Conference on Software Engineering, Minneapolis, USA.282-291. Reproduced with permission from the author.

1.1 Motivation

The motivation for this project is the need for better support for sketch tools: in particular, more accurate recognisers.

People commonly use sketches in many areas. Examples include early phase brainstorming discussion, collaborative meeting note taking, and informal sketches of more formal design artefacts like electrical circuits, software design UML and user interface diagrams, musical notations, to-do lists, and building designs. Sketching can be performed on paper, a whiteboard or a supporting electronic surface. The human-centric nature of sketches means they are easy to draw, provide rapid crystallisation of ideas, can be easily modified, and their informality is often an aide to the design and collaboration processes (Yeung et al. 2008).

People naturally sketch diagrams as a way to externalise ideas, so allowing them to explore the problem they are solving (Romer et al. 2000) and as a notation to communicate their ideas to others (Johnson et al. 2009b). Hand-drawn pen and paper sketches are commonplace for capturing early phase designs and diagrams. Pen and paper offer an unconstrained space suitable for quick construction and allow for ambiguity. Buxton (2007) describes sketches as: quick; timely; inexpensive; disposable; plentiful; having a clear vocabulary and distinct gesture; minimally detailed; ambiguous; with an appropriate degree of refinement; and meant for suggestion and exploration rather than confirmation.

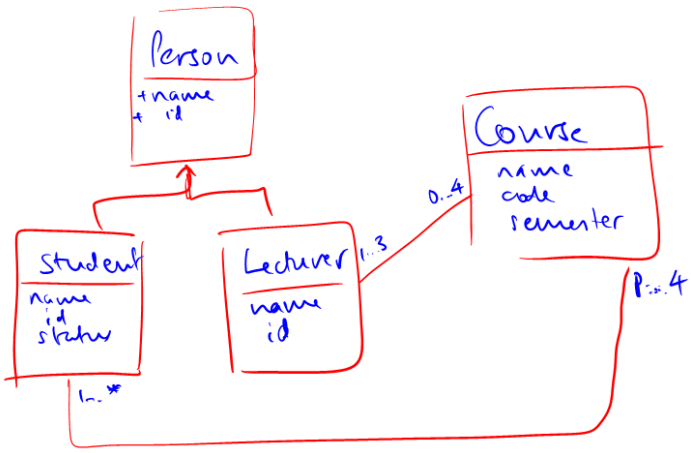


Figure 6 Example Sketched Diagram (UML Class Diagram)

As an example, consider the UML class diagram sketch shown in Figure 6. This represents classes, including class name, attributes and operations, and relationships between classes e.g. inheritance and association. Such a sketch is very easy and quick to draw, alone or in a small design group. It can be easily modified by over-writing and additions. It can be shared by copying (photocopying or

electronic). No syntax or semantic checking is done except by the humans reading it. Informal, non-compliant secondary notation such as scribbled annotations or corrections can be added at any time.

Computer-based diagramming, on the other hand, with traditional mouse and keyboard interfaces, is much more restrictive. Consider constructing the same UML class diagram shown in Figure 6 using such interfaces: the experience is normally awkward and time consuming. Constructing a similar diagram with widget-based tools is also restrictive. The users' attention is most likely to be focused more on the tools required to construct the diagram rather than the design of the diagram itself (Goel 1995; Plimmer et al. 2003a). The interaction required is disruptive to thought processes, hampering creativity and good design practice (Black 1990; Goel 1995).

For example, Figure 1 (left hand side) shows an example of a UML diagram drawn with the Visual Studio 2008 graphical design tool. To construct this, the user is required to select a Class icon tool from a toolbar, drag-and-drop the required size of a class, then add the class name, attribute and operations in specified text areas or in various form-based areas. Connecting two class icons requires selecting e.g. the Association connector tool, selecting the start class and then dragging and dropping an Association connector to the target class shape. The user then needs to specify arity and role name for the Association connector in specified text editing areas. Possible errors include wrong types of text content, incorrect connections specified between icons, overlapping icons confusing the tool, and inability to locate tools or areas to add text. Compared to sketching the example in Figure 6, using the Visual Studio UML editor (which is not a poor quality editor by any means), takes far, far longer and is highly disruptive to the creative design process (Goel 1995; Bailey et al. 2003; Plimmer et al. 2003a).

However, the computer offers several very attractive advantages over pen and paper or whiteboard sketches. These include ease of digital storage, transmission and replication gained from computerisation. They also include checking of required semantics e.g. naming conventions, syntax and semantics and automated layout for complex diagrams. Modifying complex containment structures can be much easier e.g. resizing groups of icons and connectors, copy and paste, and reusing templates of diagram components. In addition, use of specialised software can offer translation between complex forms. An example of this is shown in Figure 1 where a UML class diagram has been automatically translated into code stubs using Microsoft Visual Studio. This translation can also be done the other way - from code to diagram.

With recent advances in hardware such as Tablet PC's, computer based sketch tools offer a similar pen-based interaction experience to pen and paper. By imitating the pen and paper environment, use of

sketch tools allows for ambiguity and quick construction of diagrams (Bailey et al. 2003; Plimmer et al. 2003a). For example, the construction of Figure 6 with a pen-based interface would be a much simpler task compared to using a traditional mouse and keyboard in Visual Studio's UML designer. This is advantageous for early phase design due to its unconstrained nature; it minimises cognitive load and decreases interruptions to the flow of creativity (Plimmer et al. 2004; Grundy et al. 2007; Chen et al. 2008). This flexibility is in stark contrast to conventional widget-based environments. Potential uses include office automation, software design, electronics design, architecture and civil engineering, and education.

Automatic recognition of sketches adds even greater benefits such as automatic translation and execution of sketch models and intelligent editing as described earlier. By developing more accurate recognisers, better support can be offered by sketch tools.

1.2 Thesis Objectives

The objective of this thesis is to improve sketched diagram recognition through the development of more accurate recognisers using data mining. To achieve this, there are three sub-objectives:

- Improve recognition of hand-drawn diagrams through the development of more accurate recognisers using data mining.
  - Assemble a comprehensive ink feature library.
  - Build a repository of hand-drawn diagrams for analysis and evaluation purposes.
  - Use text-shape division as an exemplar to systematically identify the most optimal algorithms for this problem, using data mining techniques.

There are several approaches to sketch recognition. Johnson et al (2009b) categorise existing approaches into hard coded algorithms, visual matching, which includes template matching and feature-based recognition, and use of textual descriptions. These approaches are described further in Chapter 2. As discussed there we believe that feature-based recognition is the most flexible approach and has the most potential for the text-shape divider problem. Other approaches to recognition are not suitable for text-shape division due to the large within-class variation in the classes of text and shapes.

Feature-based recognition engines rely on features of sketches to aid recognition (Rubine 1991; Hutton et al. 1997; Sezgin et al. 2001; Fonseca et al. 2002; Hammond et al. 2002; Yu et al. 2003; Sezgin et al. 2005; Plimmer et al. 2007; Paulson et al. 2008a), as well as specific algorithms to combine and select the appropriate features. Features measure various characteristics of the ink; therefore they provide

important information to recognition algorithms when describing sketch components or making comparisons to pre-classified examples. Previous text-shape dividers (Bishop et al. 2004; Patel et al. 2007; Bhat et al. 2009) have also used a feature-based approach to recognition. As shown in Figure 7 the main requirements for this type of recogniser are features and an algorithm to combine those features to produce a classification for a sketch component.

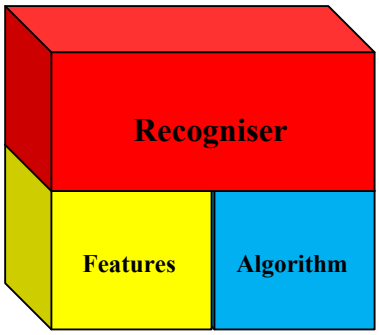


Figure 7 Feature-Based Recogniser

To build a feature-based recogniser, we need features that provide good distinguishing characteristics between classes of interest. Therefore, our first objective is to assemble a comprehensive ink feature library. The use of good features is critical to the success of the recognition, as this denotes the quality of the information passed to recognition algorithms. Many features have been employed by past systems. At the commencement of this project there was no comprehensive survey of existing features used that could act as a library from which to choose an appropriate sub-set for a given problem in sketch recognition. Given that features provide such value as input to recognition algorithms, the construction of a feature library is essential for the development of recognition techniques. As part of this research we undertook such a survey and developed a new ink feature taxonomy.

The feature set compiled in our previous work (Patel 2007) was extended to include newly found features and features from other work in recognition. We also formulated four new features to address specific misclassification problems identified. Building program code libraries for feature calculation is essential to facilitate the exploration of various feature combinations. This feature library forms the first building block for our new recognisers.

The second objective is to build a repository of hand-drawn diagrams for analysis and evaluation purposes. The development of accurate recognition techniques requires large amounts of quality digital ink data to aid the training and evaluation stages. There is very little sketch data publicly available therefore we had to build our own repository. Such data must be representative of common diagram components and be collected from a large number of people working in a natural manner (i.e. not

copied) in order to limit any bias. The repository should also include available data collected by other research groups. All data must be labelled and is most useful, if it is in a consistent format.

Data collection, labelling and dataset generation is a time consuming and laborious task. There is very little support available for obtaining such data in an efficient manner. A tool to manage these aspects of data collection and management was needed in order to provide an efficient way to manage such an ink data repository. More specifically, such a tool should assist in:

- collecting sketched diagrams from participants,
- efficient labelling of sketched diagrams for analysis purposes using automatic and manual labelling, and
- automatic generation of datasets from the diagrams for analysis based on our ink feature library.

There was no existing tool able to perform all these tasks. Development of such a data collection tool was undertaken as part of this research. It has been used to collect a large corpus of disparate diagram sketches from a diverse range of users. It has been used by a number of other researchers to collect data for their projects. It has been further extended by other researchers to include support for recognition algorithm testing, generation and comparison.

The third objective is to use data mining techniques, with text-shape division as the exemplar recognition problem, and systematically identify the most optimal algorithms, in combination with our feature library, for this problem. Data mining takes advantage of machine learning algorithms to analyse and identify patterns in data. Data mining tools are available, such as Weka (Witten et al. 2005), that can assist in performing such analyses.

A narrow range of algorithms has been employed to build feature-based recognition engines (Rubine 1991; Fonseca et al. 2002; Bishop et al. 2004; Patel et al. 2007) for different recognition problems. However, a systematic study of a larger range of algorithms has not been carried out and thus we do not know which algorithms are most appropriate for recognition. Therefore, the third stage of our research analyses and evaluates possible classification techniques: these techniques are applied to the dataset collected. Limited investigation of machine learning for text-shape division has been found to be effective (Bishop et al. 2004; Waranusast et al. 2009). This work builds on that research by drawing on a more comprehensive feature library and investigating a wide range of data mining techniques that have been systematically selected and tuned.

Text-shape division is used as an example here: while recognition of handwriting is well advanced and there have been many recognition approaches proposed for hand-drawn shapes and gestures (Rubine 1991; Sezgin et al. 2001; Fonseca et al. 2002; Wobbrock et al. 2007; Paulson et al. 2008a), there has been less attention paid to the division of text and shapes. Many existing sketch tools are limited as they are not able to distinguish between drawing elements (shapes) and text strokes in a sketch (Rubine 1991; Wobbrock et al. 2007; Paulson et al. 2008a). However, this is essential as most natural diagrams consist of both writing and drawing as shown in Figure 6 and Figure 2. Text-shape division is a difficult problem as there is large within-class variation compared with other recognition problems.

People can comprehend writing and drawing seamlessly, yet there is a clear semantic divide that suggests, from a computational perspective, that it is sensible to deal with them separately. If we consider the sketched UML class diagram shown in Figure 6, it is clear that the text and shapes in this diagram are different in nature and meaning. Because of these differences, text and shapes in sketched diagrams must be recognised independently. By dividing the text and shapes in sketched diagrams we can pass each group to separate recognisers as shown in Figure 8.

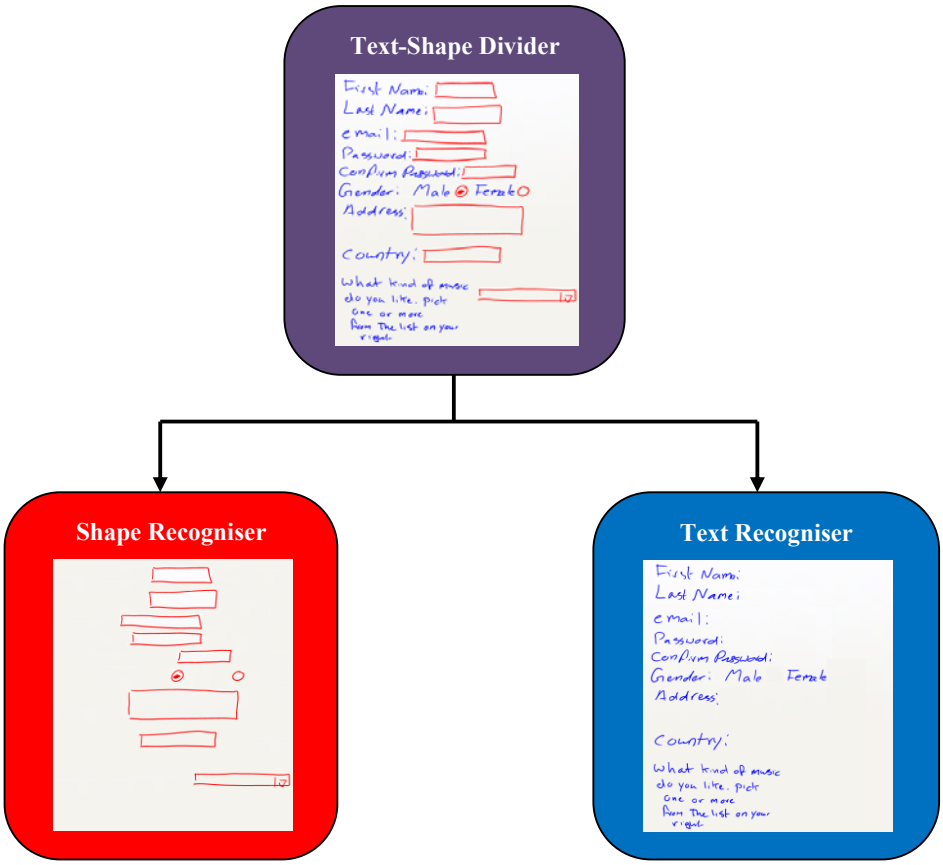


Figure 8 Sketch Recognition Process with a Text-Shape Divider

A modal interface can be adopted to separate text and shapes as provided by Freeform (Plimmer 2004). This requires the user to specify when they are drawing and when they are writing, by explicitly

changing modes through the interface. But modal interfaces cause distraction and interrupt users in completing their task, doing little to preserve the creativity and flexibility of a pen and paper-like environment that pen-based interfaces offer (Plimmer 2004).

Other solutions identify specific areas for adding text. For example, MaramaSketch (Grundy et al. 2007) and SUMLOW (Chen et al. 2008) use automatically added "text area annotations" where the user is expected to sketch text (see Figure 9). This is less intrusive and eases the burden on the recognisers as different recognisers can be applied to the content added to the text areas: it is expected to be text and not shape. However, it is still more disruptive to the user than fully-automated division and requires domain knowledge to direct the placement of text areas. For these reasons, an automatic divider of text and shapes is required.

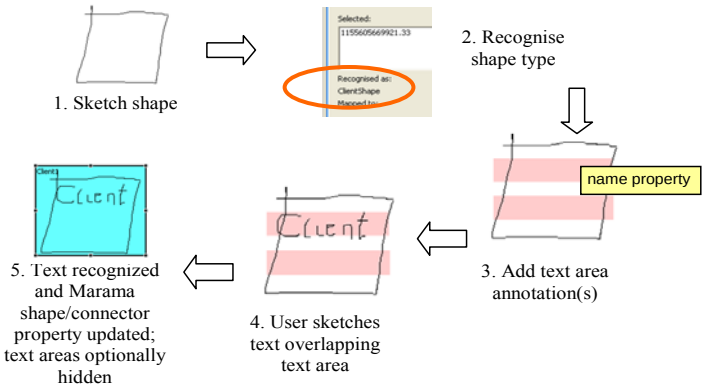


Figure 9 Text Recognition in MaramaSketch (Grundy et al. 2007)[3].

Several recognisers (Bishop et al. 2004; Patel et al. 2007; Bhat et al. 2009), commonly referred to as dividers, have been proposed for this purpose, but recognition rates in realistic situations are still unacceptable. In this research we seek to improve recognition techniques of sketched diagrams using data mining by addressing the need for more accurate division of text and shapes as an example.

Finally, to fulfil the main objective of this research, an evaluation of the best recognisers developed was conducted against three existing recognisers. The evaluation focuses on the accuracy of these recognisers. An independent set of diagrams is used for this comparison to ensure that the evaluation is fair. If the new dividers are found to be more accurate than the existing dividers on this test set of diagrams, then we can conclude that the main objective of this thesis has been met: - that recognition techniques for hand-drawn diagrams have been improved using data mining techniques, using the development of more accurate text-shape dividers as an exemplar.

[3]Image obtained from Grundy, J. C. and J. G. Hosking (2007). Supporting generic sketching-based input of diagrams in a domain-specific visual language meta-tool. IEEE/ACM International Conference on Software Engineering, Minneapolis, USA.282-291. Reproduced with permission from the author.

The key contributions of this thesis are as follows.

- A comprehensive library of ink features for sketch recognition. Two papers have been published describing this feature library.
    - Blagojevic R., S. Chang, B. Plimmer, The Power of Automatic Feature Selection: Rubine on Steroids, 7th Eurographics Symposium on Sketch Based Interfaces and Modelling (SBIM '10), 2010, Annecy, France, p 79-86.
    - Blagojevic, R., P. Schmieder, B. Plimmer, Towards a Toolkit for the Development and Evaluation of Sketch Recognition Techniques, Proceedings of Intelligent User Interfaces (IUI'09) Sketch Recognition Workshop, 2009, Florida, USA.
- A data collection tool for sketches including support for labelling and automatic dataset generation. A paper describing this tool has been published.
    - Blagojevic, R., B. Plimmer, J. Grundy, Y. Wang, A Data Collection Tool for Sketched Diagrams. 5th Eurographics Workshop on Sketch Based Interfaces and Modelling (SBIM '08), 2008, Annecy, France, p 73-80.
- A repository of labelled sketch data in a consistent format.
- A systematic investigation of data mining techniques using text-shape division of hand-drawn diagrams as an exemplar. Two papers describing this analysis have been published.
    - Blagojevic R., B. Plimmer, J. Grundy, Y. Wang, Using Data Mining for Digital Ink Recognition: Dividing Text and Shapes in Sketched Diagrams, Computers & Graphics, Volume 35, Issue 5, 2011, p 976–991
    - Blagojevic R., B. Plimmer, J. Grundy, Y. Wang, Building Digital Ink Recognizers using Data Mining: Distinguishing Between Text and Shapes in Hand Drawn Diagrams, Industrial, Engineering & Other Applications of Applied Intelligent Systems (IEA-AIE 2010), Cordoba, Spain, 2010, p 358-367
- An improvement of sketch recognition techniques for diagrams as a result of more accurate recognisers.

1.3 Thesis Outline

The remainder of this thesis is organised as follows:

Chapter 2 contains an overview of related work to this research. It begins by summarising previous work on text-shape division. This is followed by a review of the use of features in sketch recognition. Past work in the area of sketch data collection tools and in methods of data collection are summarised. Finally, existing tools and techniques used for data mining are explored.

Chapter 3 describes the methodology used to improve the accuracy of recognisers, in particular text-shape dividers. This includes a discussion of the processes used to search for features to include in our feature library; the method used to collect sketched data for our analysis; the systematic approach adopted for analysis using data mining techniques and the evaluation for the resulting recognisers.

Chapter 4 provides details of our comprehensive feature library. Features found from related work and new features we have formulated are described. The features are presented using a taxonomy we have developed to assist in categorising and describing the feature library.

Chapter 5 describes the data collection phase of this project. It begins by presenting DataManager, a tool we developed for collecting and labelling sketch data as well as automatically generating datasets for analysis with the assistance of our feature library. This includes details of a usability study performed on the tool. The sketch data collected using DataManager for training our recognisers, is also described here.

Chapter 6 presents the systematic data mining analysis performed on our sketch data to build recognisers. This includes a preliminary analysis of a large range of classifiers; tuning the parameters of seven classifiers that display promising results; the application of three different feature selection methods to these seven classifiers; the use of various ensembles of classifiers and finally, a second round of analysis exploring possible ways to improve common areas of misclassification.

Chapter 7 provides the results of the final evaluation of our five best recognisers against three existing recognisers. Independent test data used for this evaluation is described, including data obtained from other research groups. In addition, a small investigation of domain-specific recognisers is presented for more challenging diagram domains.

Chapter 8 presents the implications of this work in the wider field of sketch recognition as well as a more detailed discussion of features and the feature search conducted; data collection and the tool developed to assist with this stage; the methods used for our analysis and the results obtained, and the final evaluation results.

We conclude in Chapter 9 and present suggested directions for future work.

1.4 Definition of Terms

Term	Definition
Algorithm	"The mechanism used to generate a classifier based on input feature values." (Chang 2010)
Attribute	Same as "feature" below.
Classifier (Classifier Model)	"The artefact generated by a data mining algorithm, which can be used to classify the input data" (Chang 2010)
Divider	A recognition component that separates hand-drawn text and shapes into two groups.
Division	The process of separating hand-drawn text and shapes.
Feature	A measurable characteristic of a stroke.
Gesture	A stroke that invokes a command such as delete, edit or select.
Ink	Any hand-drawn text or shapes in a sketch.
Recogniser	"Tool implemented to classify input strokes" (Chang 2010)
Shape	A graphical symbol not including text.
Sketch	A collection of hand-drawn strokes or a hand-drawn diagram.
Stroke	A collection of points drawn by a user using a stylus from pen-down to pen-up. A stroke contains (x, y) coordinates, a time stamp and pressure value for each point.
Text	A letter, word or part thereof.

Table 1 Definition of Terms

Chapter 2

Related Work

This chapter presents a review of sketch recognition research. It begins by summarising the current state of sketch recognition techniques focusing on the division of writing and drawing. Next, a review of the use of stroke features used in sketch recognition research is outlined. Following this, in Section 2.3, an overview of data collection techniques and existing datasets available for hand-drawn diagrams and the tools available to support this task is provided. Finally, a review of data mining tools and techniques is presented to provide a background for our data analysis methodology.

2.1 Sketch Recognition Techniques

Sketch tools generally include some form of recognition. Early sketch tools include the user interface design software Silk (Landay et al. 1996) and The Electronic Cocktail Napkin (Gross 1996) for sketching early designs (shown in Figure 10). Both these tools provide basic recognition of hand-drawn diagrams. There is also early work in sketching using digital whiteboards such as LiveBoard (Elrod et al. 1992) and Tivoli (Pedersen et al. 1993; Moran et al. 1997; Moran et al. 1998) (shown in Figure 11) which include gesture recognition for commands such as scrolling, page turning, delete, select and move, and allows the user to group text and shapes manually (Moran et al. 1997).

Rubine's work (1991) in feature-based gesture recognition has been used by many other sketch recognition systems, including Silk (Landay et al. 1996). It involves using a linear classifier for single stroke ink recognition with 13 features. Rubine reported a 96.8% success rate. However, further experiments that re-implement Rubine's algorithm have been lower: 86% (Plimmer 2004) and 84% (Young 2005). Despite this, his algorithm has been widely adopted (Landay et al. 1995; Damm et al. 2000; Lin et al. 2000; Chen et al. 2003; Plimmer et al. 2003b; Plimmer 2004; Chung et al. 2005; Young 2005; Freeman et al. 2007), mostly due to its ease of implementation, with various alterations to the feature set reported.

Recognition for many diagram domains has been explored, including CALI (Fonseca et al. 2002) for general shape recognition, mechanical engineering design tools (Stahovich et al. 1995; Sezgin et al.

2001), Tahuti (Hammond et al. 2002) and Lank's system (2000) for UML class diagrams and SketchNode (Plimmer et al. 2010) for graphs, as well as multi domain recognition tools, SketchREAD (Alvarado et al. 2004) and InkKit (Plimmer et al. 2007).

In this project, we are particularly looking at the problem of distinguishing between text and shapes as a first step to recognising sketched diagrams. This is a fundamental problem required to preserve a non-modal user interface similar to pen and paper (Plimmer 2004). When text and shape strokes are accurately divided, the symbols can be passed to specific handwriting and shape recognisers to continue the recognition process.

Two particular applications of dividers are freehand note-taking and hand-drawn diagrams. The research on sketched diagram recognition includes dividers, but has also addressed recognition of basic shapes and spatial relationships between diagram components. This project has drawn on the work from both applications of dividers.

Figure 10 The Electronic Cocktail Napkin (Gross 1996)[4].

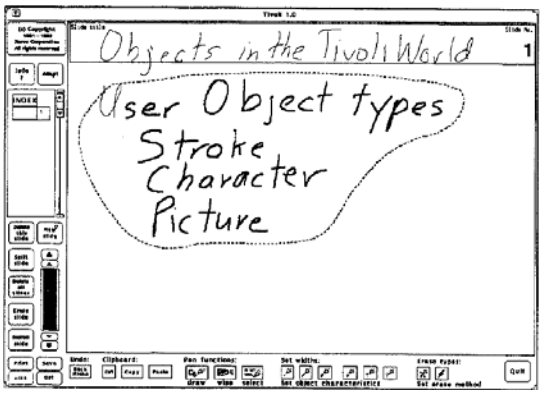


Figure 11 Tivoli (Pedersen et al. 1993)[5]

[4] Image obtained from http://depts.washington.edu/napkin/; Reproduced with permission from the author.
[5] Image obtained from Pedersen, E. R., K. McCall, et al. (1993). Tivoli: An electronic whiteboard for informal workgroup meetings. CHI '93, ACM.391-398. Reproduced with permission from the author.

2.1.1 Sketch Diagram Recognition

In the area of sketch diagram recognition, many systems focus only on shapes (Rubine 1991; Fonseca et al. 2002; Leung et al. 2002; Yu et al. 2003; Szummer et al. 2004; Qi et al. 2005; Wobbrock et al. 2007; Paulson et al. 2008a). Character recognition is also a mature area of research. However, less attention has been given to the division of text and shapes, although they are both present in diagrams.

Lank et al (2000) designed a system for recognising hand-drawn UML diagrams. Their first step is to group strokes into glyphs using intersection tests and temporal context information, and then perform recognition. The glyphs are divided into writing and drawing, based primarily on bounding box size, as character glyphs are usually smaller than shapes. They report that they have not found writing to be misclassified using this method, but they have found small shapes to be misclassified as writing. The Tahuti system (Hammond et al. 2002), another tool for UML class diagrams also performs some division of shape and text strokes. Text is considered to be smaller than shape classes and contained by or close to a class. These domain-specific solutions for division only consider a small range of symbols and can use spatial context more reliably, such as Tahuti's use of stroke location.

There are also domain- independent diagramming tools that have built-in dividers. InkKit (Freeman et al. 2007) is one such tool. The divider in InkKit has two phases. The first phase evaluates the stroke in isolation using Rubine's algorithm (1991) with partially adapted features, trained on predefined writing and drawing samples. The second phase uses spatial context to further identify which class the stroke in question belongs to. This phase rests on the theory that strokes in close proximity to one another are usually from the same class. After text-shape division, the recognition process continues and results in the identification of domain components as shown in Figure 12.

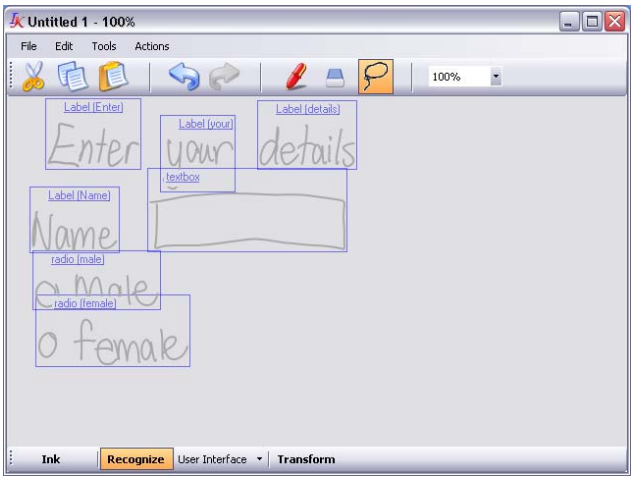


Figure 12 InkKit Example of a Recognised User Interface Sketch (Freeman et al. 2007)

Lineogrammer (Zeleznik et al. 2008) is another domain-independent sketch tool with a text-shape divider. Their approach is a variant of my previous work (Patel et al. 2007) using a decision tree which they have tuned themselves. Their divider is based on a set of heuristics examining size, geometry and spatial and temporal context. Handwriting is limited to a maximum of 2cm in height. Spatial context is used in a similar style to InkKit (Freeman et al. 2007) where strokes intersecting or close to one another belong to the same class. For isolated strokes, the ratio of stroke length to the number of cusps is used to identify cursive writing and a handwriting recogniser is used to classify any other text. Temporal information is used to classify strokes as text when sketched quickly and drawing when the input is slow.

These systems are predominantly rule-based, using various stroke features chosen heuristically to distinguish between writing and drawing.

More recent research has produced a small number of domain-independent dividers for distinguishing between writing and drawing in sketches. A variety of techniques has been used, ranging from rule-based dividers to decision trees and neural networks. All dividers are based on features. However, most studies have focused their development on one or two algorithms and rely on very limited feature sets.

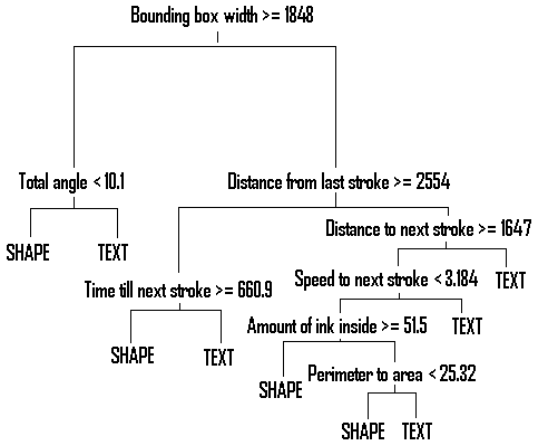


Figure 13. Decision Tree Divider Produced by Patel et al (2007)

In my previous work (Patel 2007; Patel et al. 2007) (referred to as "Divider 2007" in this thesis) we developed a domain-independent feature-based divider for shapes and text using a decision tree. This divider is unique in that it was developed using statistical analysis of a set of 46 stroke features; a much more comprehensive selection of features in comparison with other dividers. A decision tree was built that identified eight features as significant for distinguishing between shapes and text (Figure 13). The results on a test set showed a classification rate of 78.6% for text and 57.9% for shapes. Part of the test

set was composed of musical notes which had a significant effect on this low classification rate. However, when evaluated against the Microsoft (2005) and InkKit (Freeman et al. 2007) dividers, it was able to correctly classify more strokes overall for the test set.

Bishop et al (2004) developed a feature-based divider that uses local stroke features and spatial and temporal context within a Multilayer Perceptron model (this is a type of neural network) and a Hidden Markov Model (HMM) to distinguish between text and shape strokes. The features used are described in Table 2 and Table 3.

They first consider the stroke in isolation using features from Table 2. These features involve simple point-based calculations and more complicated features using principle component analysis and stroke fragmentation. Stroke fragments are used as they believe that if the largest fragment is large (possibly representing the whole stroke) and it has a high length to width ratio (feature 4 in Table 2) then the stroke is considered to be a shape stroke. A Multilayer Perceptron model was trained using these feature vectors and produces a probability for each stroke as to whether it represents text or shapes.

	Features	**Description**
1	Stroke arc length	Cumulative distance between points in a stroke (normalised by scaling using the inverse of the median fragment length of the sketch).
2	Total absolute curvature	Sum of absolute angles between consecutive points in a stroke.
3	Main direction	Calculated by fitting a total least squares model to a stroke to obtain its x and y components.
4	Eigenvalue	Length-width ratio of the total least squares fit of the stroke.
5	Number of fragments	Total number of fragments in a stroke, where a stroke is divided into fragments using the local maxima of its curvature.
6	Arc length of largest fragment	Cumulative distance between points in the largest fragment of the stroke (normalised by scaling using the inverse of the median fragment length of the sketch).
7	Total absolute curvature of largest fragment	Sum of absolute angles between consecutive points in the largest fragment of a stroke.
8	Main direction of largest fragment	Calculated by fitting a total least squares model to the largest fragment of a stroke to obtain its x and y components.
9	Length of longest side of enclosing rectangle	Length of the longest side of the enclosing rectangle of the largest fragment in a stroke (normalised by scaling using the inverse of the median fragment length of the sketch).

Table 2 Feature Set One from Bishop et al (2004) using Independent Stroke Measurements

After building an independent stroke model by training a Multilayer Perceptron with the features in Table 2, the temporal context of a stroke is used to further build upon this model. This is based on the theory that strokes drawn before or after the current stroke are more likely to be from the same class. The probability of this theory for a training set of stroke data is used in conjunction with the probability distribution from the Multilayer Perceptron model using a uni-partite HMM. This produces an overall probability for a stroke.

A variant of this algorithm was also developed which looked at the gaps between strokes. The features used are described in Table 3. A Multilayer Perceptron model was trained using these features in a similar manner as the independent stroke model, and was then combined with temporal context information using a bi-partite HMM.

	Features	Description
1	Log time gap	Log of the difference between start times for surrounding strokes.
2,3	X, Y diff from start point	x and y differences of the start point for surrounding strokes (normalised by scaling using the inverse of the median fragment length of the sketch).
4,5	X, Y diff between strokes	x and y differences of the end point of previous stroke and start point of the next stroke (normalised by scaling using the inverse of the median fragment length of the sketch).

Table 3 Feature Set Two from Bishop et al (Bishop et al. 2004) Measuring Gaps between Successive Strokes.

Three variants of the divider were evaluated, the independent stroke model, the uni-partite HMM and the bi-partite HMM. Two test datasets were used but it is unclear what domains these datasets were from. They report classification rates from 86.4% to 97.0% for these evaluations. The bi-partite HMM performed the best on the first dataset and the uni-partite HMM performed best on the second dataset. They concluded that using local features and temporal context were successful in reference to the uni-partite HMM and the results of using gaps between strokes were unclear due to the varying results of the bi-partite HMM.

A more recent development in this field has been the use of a feature called Entropy (Bhat et al. 2009) as a rule-based divider for shapes and text. Strokes are first grouped into shapes and words/letters using spatial and temporal proximity. Strokes are then re-sampled to smooth their curvature and ensure stroke points are at equal intervals. The angles between every point and its adjacent points in the stroke group are calculated. Each angle from the stroke group is matched to a dictionary containing a different alphabet symbol to represent a range of angles. This results in a text string representation of each stroke group. Using Shannon's Entropy formula (as cited by Bhat et al (2009)) they sum up the probabilities of each letter in the string to find the Entropy of that symbol. The value is normalised by dividing that result by the bounding box diagonal length. This value of Entropy is higher for text than shapes as text is more "information dense" than shapes.

They report that 92.06% of test data for which it had training examples were correctly classified. For data on which the divider had not been trained, they report a classification rate of 96.42%, where 99.21% and 69.23% of text and shapes, respectively, were correctly classified. However, only 71.06% of data was able to be classified, the remaining strokes had values of Entropy that did not fall into the expected ranges for text or shapes.

Both Bhat et al's (2009) and Bishop et al's (2004) work rely heavily on the temporal ordering of strokes. This can be a severe limitation if strokes are interspersed, where strokes belonging to the same object are not always drawn in succession, as commonly observed in sketches (Sezgin et al. 2007).

Avola et al (2009) have also developed a rule-based text-shape divider. Their system is composed of two parts: an Object Detection Engine (ODE) and a Domain Separation Engine (DSE). The ODE decides which strokes belong to the same object by comparing the curvature and entropy of strokes that intersect or are close to one another. For example, one rule to identify strokes that belong to the same object is that if each individual stroke has high curvature and entropy, then the value of curvature and entropy must also be high when the strokes are joined together. There are three similar rules that apply when deciding if strokes belong to the same object. The DSE is then responsible for classifying the objects into either writing or drawing. Three features are used to perform the classification:

1) Enclosing rectangle linearity
   This measures if the enclosing rectangles of the object line up according to their barycentre points. If they do line up, the object is considered to be a line of writing.
2) Bands ratio
   This examines the pixel density of the object by splitting the bounding box into three areas - the top, middle and bottom - to account for both cursive and block letters. Three rules are used to determine if the object is text. The first is if the middle density is high accompanied by a small density at the bottom (for cursive writing); secondly if the density is equally distributed in all areas (for block writing); and thirdly, the density is growing.
3) Enclosing rectangles ratio
   This is the ratio of the total area of all the enclosing rectangles for the object and the total area of intersection between the enclosing rectangles. If this ratio is 15% or below, the object is considered to be writing.

Their divider was evaluated on two datasets: one simple dataset where sketches were composed of either shapes or text – no mixture of classes; and a more complex dataset with mixed shape and text objects. They achieved a 96% success rate on the simple dataset, but only 89% on the more complex dataset. Neither of these datasets is composed of realistic examples as they are isolated components as opposed to full diagrams.

Rodriguez et al (2008) used both an unsupervised clustering technique and linear discriminant analysis to develop dividers. Their divider is based on the motor models of handwriting, where different classes require distinct hand movements when sketching. Using this theory, they propose that text and shapes can be distinguished by studying the frequency of oscillations in strokes.

They first calculate a direction feature vector for a stroke based on the angle between consecutive points. This vector is converted to discrete Fourier transform coefficients and then PCA (principle component analysis) is applied to these values to reduce the values to 2D space.

Two experiments were performed. The first experiment used an unsupervised fuzzy k-means clustering algorithm to see how well the data could be grouped naturally into the text and shape classes. For two clusters; there was 62.5% purity for the text cluster and 97.2% purity for the shape cluster.

The second experiment used two-fold cross validation of the Fisher linear discriminant analysis to distinguish between text and shapes. The data was labelled in this case for training and testing. This method was compared to a handwriting recogniser that gives confidence scores based on the probability of the strokes being text. Average recognition rates are not reported but they found that although the handwriting recogniser performed better, their method was faster. A combination of these systems was also tested and actually performed the best overall.

An unfortunate limitation of their work is that writing must consist of two or more characters to be classed as text to avoid misclassification of crosses and circles with ‘O’ and ‘X’. This is obviously not consistent with naturally drawn sketches.

Microsoft has developed recognisers for ink analysis that are now built into its operating systems (from Windows Vista onwards) (Microsoft Corporation 2008). Part of the ink analysis is able to separate writing from drawing. It is not known what kinds of techniques are used to perform classification for this divider. In our previous work (Patel et al. 2007) we ran a comparative evaluation of an older version of Microsoft’s divider (2005) against our own. We found that the Microsoft divider was heavily biased towards text. On a test set of hand-drawn diagrams, it misclassified 93.1% of shape strokes and only 1.4% of text strokes, indicating that almost all strokes were classified as text. Although this might be suitable for documents where there is a higher proportion of text to shapes, this bias makes it unsuitable for diagrams. A similar bias towards text is present in several dividers (Bishop et al. 2004; Patel 2007; Bhat et al. 2009) although, judging from the results presented, not to the same extent as the Microsoft divider.

Other recent work in this area (Peterson et al. 2010) has involved using AdaBoost and an extension of our previous feature set (Patel et al. 2007) to build a text-shape divider, as well as grouping strokes after division. This divider was evaluated against the Microsoft divider (Microsoft Corporation 2008) and Entropy (Bhat et al. 2009) using two datasets. Their divider correctly classified 90.2% and 97.2%

of each dataset. These results were more accurate than the Microsoft divider, but only more accurate for one dataset against the Entropy divider.

2.1.2 Document Analysis

Research in the area of digital ink document analysis for freehand note-taking has explored text and shape division (Machii et al. 1993; Jain et al. 2001; Shilman et al. 2003; Ao et al. 2006). There has also been some work separating Japanese characters from shapes in documents (Machii et al. 1993; Mochida et al. 2003). This work is summarised below.

Jain et al's (2001) work aims to find the overall structure in online documents. One part of this problem is to distinguish between text and shapes. Their approach is rule based, using stroke length and curvature as its primary features. For text strokes, the longer they are, the more curvy the strokes become. For shapes, the longer they are, the less curvy they become. Their evaluation showed their approach to correctly classify 99.1% of strokes in the test documents. The misclassified strokes were found to be mostly very short strokes that were wrongly classified as text.

Ao et al (2006) have also designed a system for document analysis using a Support Vector Machine (SVM). Their goal is to be able to select objects in ink documents rather than just individual strokes. This requires some intelligence, beginning with a divider. First they group strokes that are likely to be part of the same component, using temporal and spatial context. These groups are then divided into text and shapes on the premise that the layout of text is more ordered than the layout of graphics: therefore they look for differences in document layout.

Thirty features are used to describe the components. This is a large feature set in comparison to other dividers. The features are separated into three categories:

1) Features of the component;
2) Features of the whole text line containing the component, and
3) Features of the adjacent components or text lines.

A SVM is trained on data with the above feature measurements. Their evaluation showed that 92% of their data was correctly classified. Some of their misclassifications were due to incorrect stroke grouping. However, the layout of text in diagrams is not as ordered as in documents, so this method may not be suitable for diagrams.

Waranusast et al (2009) have developed a text-shape divider for patient records also using an SVM. The features used are based on two spatio-temporal graphs that are constructed for each document. The

X-T graph plots each stroke point's time stamp against its x-value, and the Y-T graph does the same for the y-value. Nine features are calculated using these graphs; they are separated into line-based and frequency-based features.

After comparing a k-nearest neighbour classifier with a SVM they found that the SVM was more accurate. A SVM classifier was trained, using 10-fold cross validation within Weka (Witten et al. 2005) to produce a divider. The accuracy obtained from the 10-fold cross validation test was 94.61%. However, the data that was used to train and test this classifier is highly flawed. They were unable to obtain digital ink data of patient records as the doctors in the hospital they worked with only used paper. Photographs of the paper records were taken and traced to produce digital ink documents by displaying the photographs on the tablet screen. It has been shown by Field et al (2009) that training recognisers with copied data can have a significant effect on recognition accuracy when its real purpose is to recognise more natural data.

Shilman and Viola (2004) combine grouping multi-stroke symbols with recognition of these symbols, where text is considered as another class of symbol along with other shapes. They use AdaBoost and boosted decision trees as a classifier and compute features based on curvature, orientation and end point characteristics. Their experiments show much higher errors when the symbol set included writing (in the form of digits) with shapes as opposed to shape-only data.

Using the attributes of document layout has been the focus of Shilman and Wei et al's (2003) work in dividing text and shapes. They suggest that text and shape classification is closely coupled with document analysis as one can use the attributes of handwriting structure to provide context to the problem. They first establish the documents layout, assuming everything is text. The text-shape division is performed using local and global features, although the only features that are mentioned specifically are fragment count and regression error. They report an accuracy of 94.1% for their evaluation of the text-shape divider.

Machii et al (1993) designed a Japanese character and shape divider for documents. Their divider is a simple rule-based system. A stroke is classified as drawing if the stroke length and bounding box side length are above a threshold, as drawing strokes were found to be larger than text strokes. They then look at stroke intersections, as they found that text and shape strokes do not often intersect, so intersecting strokes were assumed to be from the same class. Lastly, they considered the temporal context of strokes. They found that single stroke characters were very rare. Therefore, if the strokes before and after a single character stroke are classed as drawing, then this stroke is re-classified as

drawing also. Their evaluation showed that this method classified 88% of strokes correctly. The results showed a strong bias towards text.

Mochida et al (2003; 2004) also developed a rule based system and a neural network for distinguishing between Japanese characters, mathematical formula and figures in handwritten documents based on local and contextual features. The size of the stroke is studied first, using the length of the stroke's bounding box diagonal. After examining the distribution of stroke size for each class they found that text and formula are usually small and figures range widely from small to large.

Contextual features are then used to further classify each stroke. They found that intersecting strokes are more likely to be from the same class. They detect lines of text using the distance from the first stroke point to the last. Stroke density is also measured as the number of strokes in a line of text divided by the size of the line. They have found that the density of text is greater than a mathematical formula. They also use scores obtained from formula and text recognition engines. Their evaluation results report their system achieves 81% accuracy on average. They also trained a neural network using the same features as their original system and found an average accuracy of 71%. The neural network was more successful at distinguishing figures from text and formulas.

The style of Japanese characters is significantly different to Latin text, so these approaches to division may not be successful in our case. As the content of documents is mainly text, many of the dividers for digital documents, in general, have held some bias which makes them unsuitable for sketched diagrams.

The division of text and shapes in sketched diagrams has received little attention thus far. Past research in this area has employed a number of features and algorithms, although most have concentrated on one or two algorithms and use very limited feature sets. Accuracy rates are wide-ranging among these dividers, but it is difficult to compare performance when different test datasets have been used. Furthermore, some dividers have severe limitations that prevent their use on real sketches.

There has been no extensive study of algorithms to determine the most optimal fit for accurate text-shape dividers. There is also a need for a more comprehensive library of ink features that can be employed for distinguishing between text and shapes as an input to these algorithms.

2.2 Features

Stroke features measure characteristics of strokes such as curvature or size. When we draw or write with a digital pen we create collections of strokes. A stroke is a collection of points from pen-down to pen-up. In addition to the (x, y) coordinates of each point in the stroke, time and pressure data for each point are also available. This data provides inputs for calculating stroke features. With future advances in hardware, the input data is expected to be even richer, containing measurements such as pen tilt or acoustic emissions (Seniuk et al. 2009).

There have been many recognition approaches proposed for hand-drawn sketches, but stroke features are central to most of these systems. Johnson et al (2009b) group the techniques for sketch recognition into hard-coded recognisers, visual matching algorithms and those based on textual descriptions. Hard-coded algorithms are those that are based on fixed rules. They are more suitable for simple recognition problems and are difficult to extend and maintain (Johnson et al. 2009b), therefore we do not believe they are suitable for dividing shapes and text. The use of textual descriptions involves using pre-defined grammar to write descriptions of sketch components and relationships to other components. This approach usually requires some domain-specific knowledge of what is being drawn in order to describe the components accurately and can be difficult if the classes of interest are hard to describe in words (Johnson et al. 2009b). As we are interested in a divider for general diagramming, the use of grammars may not be useful at this stage. Furthermore, the characteristics of the classes' text and shape are so large and varied that describing them textually would be a cumbersome task. We therefore conclude that the use of textual descriptions for a text-shape divider is not suitable.

The remaining category is visual matching. This category is further separated into template matching and feature-based recognition (Johnson et al. 2009b). Template matching relies on distance-based functions to measure the similarity of a sketched component to pre-categorised examples. As with the use of textual descriptions, we believe this is not suitable for a general text-shape divider due to the large variation present in each class. If we were to use template matching, numerous examples of every letter of the alphabet as well as all forms of drawings would be required. Although this is possible, it would not be a very elegant solution to the text-shape divider problem and would be limited to the examples given. Feature-based recognition, on the other hand, is a more flexible solution to the text-shape divider problem. This involves the measurement of various ink features and the use of an algorithm to combine these features to produce classifications. Previous text-shape dividers (Bishop et al. 2004; Patel et al. 2007; Bhat et al. 2009) have also used a feature-based approach to recognition.

Paulson et al (2008a) categorise recognition techniques as feature-based or geometrically based approaches. Although not all techniques are explicitly regarded as feature-based, they all rely on information provided by various measurements of the digital ink strokes (Rubine 1991; Hutton et al. 1997; Sezgin et al. 2001; Fonseca et al. 2002; Hammond et al. 2002; Yu et al. 2003; Sezgin et al. 2005; Plimmer et al. 2007; Paulson et al. 2008a), as well as specific algorithms to combine and select the appropriate features.

For recognition to be successful, the features fed into the algorithms must provide good distinguishing characteristics between classes of interest. While small feature sets have been reported, currently there is no survey of existing features that could act as a library for algorithms to employ for a given problem in sketch recognition. Given the importance of features, such a library would be a significant and useful resource for further development of sketch recognisers.

2.2.1 Existing Ink Feature Sets

Rubine (1991) reported the first feature set composed of 13 features. These features were used in a gesture recognition system. This feature set has been used by many sketch recognition systems since (Landay et al. 1995; Damm et al. 2000; Lin et al. 2000; Long et al. 2000; Chen et al. 2003; Plimmer et al. 2003b; Plimmer 2004; Chung et al. 2005; Young 2005; Bickerstaffe et al. 2007; Freeman et al. 2007; Patel et al. 2007; Zhang et al. 2007; Paulson et al. 2008b; Willems et al. 2008; Meyer et al. 2009; Willems et al. 2009). Several of these have added or removed features from Rubine's base set.

Long et al (2000) also developed a system for gesture recognition. They extended Rubine's feature set (1991) with 11 additional features. Multi dimensional scaling analysis and regression was then used to identify distinguishing features for determining the similarity of gestures.

Others have developed their own feature sets for particular recognition problems including numerous feature sets for shape recognition (Fonseca et al. 2001; Sezgin et al. 2001; Calhoun et al. 2002; Fonseca et al. 2002; Hammond et al. 2002; Leung et al. 2002; Qin 2005; Young 2005; Paulson et al. 2008a) and for text-shape division (Machii et al. 1993; Bishop et al. 2004; Young 2005; Bhat et al. 2009) as mentioned in Section 2.1. Chapter 4 lists the features used in these systems in further detail.

Although many features have been used for sketch recognition, there is a lack of synthesis between the feature sets. There are many overlapping features as well as unique features proposed. Our feature library from previous work (Patel et al. 2007) is one of the few that is based on a compilation of features used in sketch recognition. It was developed for building text-shape dividers, and is discussed in Section 2.1. A library of 46 potential ink features was compiled: these also included our own

additions. We categorised the features into seven categories: pressure; time; intersections; size; curvature; tablet OS recognition (probabilities generated by the Microsoft Tablet OS recognition engine) and inter-stroke gaps (which include features measuring spatial and temporal context). We found that features measuring inter-stroke gaps, size and curvature were important for distinguishing between writing and drawing. This was demonstrated by the accuracy of the new divider using these features in comparison to two other dividers.

More recently, there have been similar feature sets published. Willems et al (2008; 2009) compiled feature sets for recognition of multi-stroke gestures based on previous work in sketch recognition and their own features. They composed three feature sets measuring varying levels of detail for each gesture. The first set, g-48, contains 48 features measuring aspects of the whole gesture (which may be made up of one or more strokes) obtained from previous work. The second set, s-μ-σ, has 758 stroke based features which include g-48 features as well as the mean and standard deviation of each g-48 feature on each stroke and variations of these features based on size and rotation. The last set, c-n, contain point based features from previous work in character recognition. Two subsets of c-n are used, c-30 and c-60, which indicate the level of re-sampling that is done to the gesture to calculate the features. For example, gestures are re-sampled to 30 points for c-30 features.

They explored various combinations of these feature sets to determine which performed the best in relation to recognition accuracy. The data they used for this study comes from objects and events on interactive maps for crisis management, which mostly include characters, symbols and command gestures. Using the best individual N (BIN) feature selection algorithm, a feature ranking algorithm using SVM's, they found that g-48, μ and σ features contribute similarly to recognition accuracy.

The best ranked features were then used to train a Multilayer Perceptron and a SVM classifier. In addition, a Dynamic Time Warping classifier, which uses template matching, was trained with the training dataset. When comparing the g-48 features with and without μ and σ features; the best results were obtained using g-48 features with μ and σ features: there was approximately 25% improvement in accuracy on average. The g-48 with μ and σ features were also approximately 39% better than c-30 and c-60 feature sets on average.

Overall, this work showed that the global features, measuring aspects of the whole gesture for multi-stroke gestures, are as important as stroke based features (μ and σ) and that, in general, the use of a good feature set can produce improvements in recognition accuracy.

Other more recent features have been proposed by Avola et al (2008) as part of their framework for building trainable basic shape recognisers. This library is based on the CALI features (Fonseca et al. 2001; Fonseca et al. 2002) but also includes ten features measuring spatial context for describing the semantics of the shape in terms of line and arc primitives. These contextual features include measurements of length, angles, intersections, distance and relative positions of components in relation to other components in the sketch. Such features are useful for building a semantic model of a sketch.

2.2.2 Ink Feature Studies

Further studies have been conducted regarding feature selection, for distinguishing the types of features that are significant for various recognition problems.

Paulson et al (2008b) conducted a study of what they termed geometric features versus gesture-based features for basic shape recognition. Geometric features are considered to be those measuring the shape of strokes, while gesture based features concentrate on how strokes are drawn. They compiled 44 features from the PaleoSketch system (Paulson et al. 2008a) and Rubine's algorithm (1991) used for basic shape and gesture recognition.

A subset of features was chosen using a 10-fold greedy sequential forward selection technique with a quadratic classifier as a wrapper. Ten subsets were selected, one for each fold. Each subset was analysed according to most frequent features found in each set. For example, features that were present in all ten subsets were weighted as a 100% optimal feature; those that were in nine out of ten subsets were 90% optimal and so on. They found that a feature subset using features with weightings between 50-100% was the most optimal combination.

Two quadratic classifiers were trained using a 50/50 split of the data based on users, where one used the full feature set and the other the subset of features. Another two quadratic classifiers were trained using 25-fold cross validation on the full feature set and the selected subset. The results showed that the quadratic classifiers using the selected feature subset performed better than those using the full feature set. The results also showed that geometric features were more successful for basic shape recognition than gesture based features. Only one gesture based feature (total rotation) was found to be optimal.

Zhang et al (2007) conducted a small study comparing features with three algorithms for multi-stroke shape recognition. The features they tested were mostly from previous work in sketch recognition and included Rubine's features (1991), centroid radius, curvature, normalised curvature, compositional features (including dynamic features and means and standard deviations of other features), speed and a modified turning function. They tested the accuracy of three algorithms; SVM, Hidden Markov Model

(HMM), and a Bayesian Belief Network (BN); when each feature is used as input to see how successful each feature (or group of features) is at recognising their data. The data they used for training and testing came from electrical diagrams and consisted of 10 different graphical symbols of varying degrees of complexity collected from four participants. Unfortunately, the data was highly biased as participants were asked to draw symbols in specific styles. The study found that speed and turning features produced high recognition results for all algorithms. For BN, the compositional features had optimal results. For SVM, curvature was very successful. The centroid radius and Rubine's features did not produce very good results for any algorithms. Overall, BN produced the best recognition results when taking all features into account.

They also conducted a second experiment where they varied the sample sizes of training and test data to see what was most successful. Using the results obtained for the first experiment, they were able to narrow down the feature set for each algorithm to 3-4 features. For the speed feature, SVM performed better than HMM, and both produced good results with a small collection of 100-200 samples. SVM was also more successful than HMM with the curvature feature but needed a larger sample to obtain such a result. For the turning function, HMM performed best at 400 samples. Both SVM and HMM increased in accuracy as the sample size increased in this case. Overall, BN was found to be the most suitable algorithm for multi-stroke shape recognition as it was able to achieve over 92% correct with only 100 samples and had the shortest training time of all. This study illustrates that feature and algorithm combinations are important to recognition accuracy.

A more recent study (Tumen et al. 2010) investigated feature extraction and the use of classifier ensembles in image-based sketch recognition. Image-based recognition treats sketches as raster images and calculates features based on image patches rather than strokes (Tumen et al. 2010). To obtain feature sets, various feature extraction methods can be applied to these images, such as use of Zernike moments (Hse et al. 2004). After trying five methods of feature extraction with various parameter settings, they found that the choice of method for feature extraction and the parameters used within these methods has a direct effect on classifier accuracy, therefore, they should be chosen very carefully.

Feature selection was also investigated. For the datasets they used, the use of feature selection had a positive effect on classifier accuracy. It was found that the optimal number of features varies, depending on the domain of use. Past this optimal number of features a decrease in classifier accuracy was observed. These results confirm the importance of quality features. In addition, classifier ensembles were tested against individual classifiers: almost all ensembles were found to be significantly more accurate than the individual classifiers. The ensembles were found to produce more accurate results even when the feature set was not optimal.

Chang (Blagojevic et al. 2010) also conducted a recent study of the effects of feature selection using the feature library compiled in this thesis, described in Chapter 4, for basic shape recognition. Eight feature selection algorithms were investigated where feature subsets were selected and used in conjunction with Rubine's (1991) linear classifier. Of the feature selection algorithms tested, Relief F was found to produce the best results where sub-optimal features were not chosen and good levels of accuracy could be achieved with a small number of features. The combination of the Relief F feature set with Rubine's algorithm was evaluated against three other basic shape recognisers: the original Rubine's algorithm (1991); InkRubine (Plimmer et al. 2007); and $1 (Wobbrock et al. 2007). Three datasets were used for the evaluation: basic shapes; directed graphs; and class diagrams, all collected from 20 participants. It was found that the new recogniser was significantly more accurate than any other. These results demonstrate the value of good features and the use of feature selection. Experiments were also conducted using different feature combinations with Rubine's algorithm. It was found that the combination of features used is as important as the individual value of each feature in a feature set.

Choi et al (2008) also compared Rubine's features (1991) and $1 (Wobbrock et al. 2007) to a new feature set they developed for gestures. The main difference in their features is that they are automatically generated using manifold learning, in particular a kernel Isomap. The k-nearest neighbour algorithm was then used with this feature set and a weighted version of the feature set. These recognisers were evaluated against the others using three datasets: one was composed of characters drawn by a single participant; the second were digits drawn by another participant; and the last dataset was of mathematical symbols collected from ten participants. Their recognisers were consistently better than Rubine's but had more difficulty against $1. The latter was better on the first dataset, between the two new recognisers on the second dataset and worse than both on the third. Although automatically generated features require less work to compile, we believe more evidence is required to show that they are more accurate than humanly chosen feature sets.

These studies have consistently shown that features are an important element in developing accurate recognisers. The use of feature selection algorithms, ensemble classifiers and good feature and algorithm combinations can also contribute to more accurate classifications. However, these studies are limited to the field of gesture or shape recognition; such analysis has not been performed for text-shape division.

Many feature sets have been proposed but very little work has been done to assemble a more comprehensive feature library. Although Willems et al (2009) report a feature set of 758 features, these are simply extensions of a base set of 48 features – a similar size to our feature set compiled in previous

work (Patel et al. 2007). Given the importance of features, compiling the existing features used in sketch recognition into a single library would serve as an invaluable resource for recogniser development.

2.3 Data Collection

To perform a comprehensive analysis of stroke features and algorithms we need large amounts of data. Comparative evaluations, to judge the success of algorithms, also require standard datasets. This section reviews previous work in data collection, labelling and corpus management of sketched diagrams. In particular we review past work in collecting sketches and the corpuses that exist as well as the tools available to support data collection, labelling and data management. We discuss limitations to previous work that have motivated our research in this area.

2.3.1 Existing Sketch Datasets

At the outset of this project, there was very little sketch data available that included text and shapes. The ability to develop text-shape dividers was hampered by the lack of public datasets for training and testing. Some datasets that were available are described below.

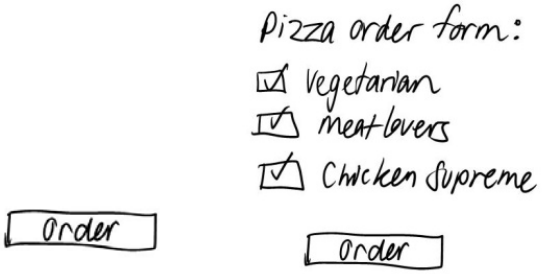
Oltmans et al (2004) built an Experimental Test Corpus of Hand Annotated Sketches (ETCHA Sketches) containing a total of 750 strokes. The process of constructing this database included collecting sketches and then labelling the primitive shapes within the sketches. Their data cover four domains including circuit diagrams, family trees, floor plans and geometric configurations. However, there is no text included in these sketches. Participants were asked to label their diagrams themselves. As different recognition problems require slightly different data from each sketch, four possible types of labels were identified:

(a) “Best in isolation” labels for a single stroke classifier,
(b) Context based labels,
(c) “Is a”, and
(d) “Can be a” labels where a group of labels are assigned to a stroke - for example, a slightly curved line is a line and an arc.

Hse et al (2004) also compiled a test corpus of 7410 examples. Their participants sketched examples of 13 different isolated symbols, which are predominately basic shapes such as rectangles and circles. However, only one single diagram component was drawn for each sketch. In this case, data concerning stroke relationships in a full diagram are lost. Again, there is no writing included in any of these examples.

Alvarado et al (2007) collected a set of hand-drawn logic diagrams containing 8616 strokes. This dataset is one of the few that contains text as well as shapes. Another unique characteristic of this dataset is that it was collected from real world examples in students' assignments, class notes and lab reports.

A dataset of sketches was also built for our previous work (Patel 2007; Patel et al. 2007) containing 1519 strokes from various diagram domains. Participants were asked to copy diagrams from pre-drawn figures on paper. This may have caused some bias in the timing data obtained as we would expect participants to copy diagrams much faster than when constructing their own from scratch. Also, many of the diagrams were not complete but were composed of isolated diagram components as shown in Figure 14a, as opposed to a full diagram like Figure 14b. This data is similar to Hse et al's dataset (2004). This would have influenced some of the information obtained regarding stroke relationships. It was also clear that a more efficient method of collecting, labelling and managing large amounts of data was required as this process was very time-consuming and error-prone.



a. Isolated component b. Full diagram

Figure 14 Isolated Component Example Versus a Full Diagram.

Recently, more data has become available. The NicIcon repository (Niels et al. 2008) of hand-drawn icons for crisis management is publicly available[6]. Fourteen icons are represented in the repository, composed predominantly of shapes. A total of 24, 441 icons were collected from 32 participants. This is the largest repository available in comparison to those described previously. However, the icons are drawn in isolation and contain very little text.

Further studies conducted in data collection (Field et al. 2009; Schmieder et al. 2009) have confirmed that the best recognition rates are obtained when classifiers are trained on task-specific data. For example, recognisers trained on full diagrams classify full diagrams more accurately than recognisers trained on isolated components. Isolated components are often preferred when collecting data as they

[6] http://unipen.nici.ru.nl/NicIcon/index.php?page=download-online

are easier to label. These results are especially significant when user data is present in training and test sets (Field et al. 2009).

The datasets described above provide a good start to building a repository of data for sketch recognition research, but there are many other domains to consider. There are also limitations to overcome regarding the collection of quality data that includes both text and shapes. In order to collect large amounts of data more efficiently, we need support tools for all aspects of data collection and management.

2.3.2 Sketch Data Management Tools

Wolin et al (2007) designed a tool for more efficient labelling of ink data using a stylus. Their tool is able to complete three main tasks; stroke fragmentation (automatic and manual), stroke grouping and symbol labelling. They claim that fragmenting strokes is important before labelling, as users may draw more than one symbol using a single stroke. Fragmenting can also help divide strokes into primitives i.e. lines and arcs. Stroke grouping is for the opposite problem of labelling components made of more than one stroke. Once these tasks have been performed, labelling the symbol in the sketch can be carried out efficiently. Their tool also allows for multiple labels to be applied to a stroke.

Their usability study showed that overall the user interface was intuitive and easy to use. Possible improvements that were discussed were that better support is required for using multiple labels and that undo/redo is helpful in such an interface. Although this tool has very useful features for labelling sketches, it only covers one stage of the overall data collection and management process.

iGesture (Signer et al. 2007a; Signer et al. 2007b) is a framework for developing and evaluating gesture recognisers. There are 3 main parts to iGesture as described below.

1) Test Bench
   A user can draw a single gesture (or use a gesture from an available database) and use it to test the accuracy of different recognition algorithms.
2) Admin
   This interface allows a user to draw new gestures and add them to a library of gesture examples, manage and edit gesture classes and import and export the gestures collected using XML.
3) Test Data
   This function exports test datasets to XML to be used for batch evaluations of the data against different algorithms. Batch evaluations allow the user to comparatively evaluate many algorithms/configurations against the same data.

Four small gesture sets collected using iGesture (containing 20-30 sample gestures) are available online. This data is similar to previous corpuses in that they only contain single gestures as opposed to complete diagrams, therefore lacking any contextual data that may exist between gestures. Training with such data has a direct effect on recognition accuracy if the recogniser is to be applied on more complex sketches (Field et al. 2009; Schmieder et al. 2009).

Other, more recent, tools have been developed focusing more on sketch data collection.

SOUSA (Paulson et al. 2008c; Kaster et al. 2009) is an online applet for collecting and verifying sketches. A researcher can set up a study to collect sketches from particular domains. Users can participate in this study via the website and sketch the required diagrams accordingly. Verification of these sketches occurs when participants classify diagrams according to a list of possible categories. This is to confirm that the user has in fact drawn what was required and to identify any sketches that are ambiguous. SOUSA does not provide any labelling functions as the MIT file format it uses does not support labels. They recommend the use of external labelling tools, such as Wolin et al's labeller (2007). Their web interface provides ease of access to search, collect and download data. However, there is no way of automatically generating datasets from these sketches for use in training algorithms.

There has also been some work in developing web based games to collect sketch data (Johnson 2009; Johnson et al. 2009a) with the premise that using games for sketch data collection provides participants with a fun environment while still obtaining important information for researchers. Picturephone (Johnson 2009; Johnson et al. 2009a) is one such game. It is modelled on the children's game "Telephone". There are three phases in this game.

1) Use words to describe a sketch.
2) Draw using a textual description.
3) Judge how similar two other participants' drawings, are using a Likert scale.

Participants receive points at various stages of the three phases, thus creating a competitive game where success can be quantified. All the data produced by participants is available online and contributes to a public corpus of sketch data.

Another game, Stellasketch (Johnson et al. 2009a) is based on Pictionary. It involves multiple participants playing simultaneously. One player is randomly selected to sketch a clue while the remaining players guess what is being drawn. Many labels (or guesses) can be made while the sketch is constructed: however, each player's guesses are hidden from the other players until the sketch is completed.

Like SOUSA (Paulson et al. 2008c; Kaster et al. 2009), these games focus solely on data collection: there is no labelling facility for particular strokes in a sketch, or automatic dataset generation functionality available.

MacLean et al (2009) have developed a tool for collecting and labelling handwritten mathematical expressions. Their goal was to generate a corpus of data with limited bias. They believe that one source of bias comes from researchers collecting only the type of data that their recognisers are more familiar with. This tool seeks to produce less biased data by ensuring their datasets are as varied as possible. They developed a tool that uses random walks of a grammar-based maths recogniser to generate a large range of expressions for participants to transcribe. Participants write these expressions and strokes are then labelled automatically, using a recognition algorithm for maths expressions. Unfortunately, this tool is limited to the collection and labelling of mathematical expressions.

GestureLab (Bickerstaffe et al. 2007; Meyer et al. 2009) is a tool for generating domain-specific sketch recognisers. It has the following functions.

1) Data collection

   Hand-drawn gestures can be collected, but their data collection is based on single gestures or components at a time, not full diagrams. This can affect recognition accuracy if a recogniser is trained with such data but then used to recognise full diagrams (Field et al. 2009; Schmieder et al. 2009). With single gestures, they are unable to gain important spatial and temporal context information from this data. They categorise a gesture into specific classes as a way of labelling the data. All data is stored in a database that allows remote access for sharing data with other researchers.
2) Feature definition

   The standard features in the tool are the 13 features from Rubine's work (1991).
3) Algorithm definition

   The standard algorithm included in the tool is a SVM.
4) Recogniser training and evaluation

   The data collected for a specific domain is used in conjunction with Rubine's features (1991) to train and test a minimum cost spanning tree SVM and produce a domain-specific recogniser. A probability of that gesture belonging to each class in the domain is produced.

The domain-specific gesture recogniser produced by GestureLab can then be combined with Cider (Jansen et al. 2003), a domain-independent recognition engine that uses production rules to parse the diagram based on a defined grammar. The parser is editable by the user, allowing them to add, delete

and edit diagram components. However, if the gestures are not correctly recognised by the GestureLab recogniser, Cider will be unable to parse the diagram correctly.

Two recognisers have been built using Cider & GestureLab: one for finite state automata and the other for mathematical expressions (Bickerstaffe et al. 2007; Meyer et al. 2009). Their evaluations report a 99.67% recognition rate for the finite state automata recogniser and 98.9% for the maths recogniser. These recognisers do not use a divider to separate text and numbers from shapes in the diagrams. The alphabet and numbers are treated as separate classes in the dataset that is used for training for recognition. This method requires a tedious collection of numerous examples of each character and digit. There is no functionality allowing the generation of datasets for algorithm training. Training is performed using a limited feature set and one algorithm internally. This limits the ability of a researcher to analyse different algorithms using external tools – the algorithm must be built into GestureLab.

Avola et al (2008) have also developed a framework for building trainable basic shape recognisers using a similar approach to InkKit (Plimmer et al. 2007) where symbols are drawn by the user to build a library of data. This system, like others, concentrates on isolated symbols rather than full diagrams, which can be detrimental to recognition accuracy for recognisers meant for full diagrams (Field et al. 2009; Schmieder et al. 2009). One interesting addition is that they allow the inclusion of speech and text attached to each example to provide more semantic meaning to the sketch. This could be used as a form of labelling.

The data available is insufficient for a thorough analysis of features and algorithms for text-shape division. Collecting, labelling and managing sketch data is a time-consuming and tedious task. Various tools have been developed to assist in these tasks, but each of them have their own limitations. Many are limited to the collection of isolated components rather than full diagrams; some are only able to assist in one particular task and others are built for a particular domain. There are no tools that perform data collection, labelling, and automatic dataset generation with a comprehensive feature library for full diagrams from all domains. With appropriate datasets, a systematic analysis of features and algorithms can be carried out using data mining to develop more accurate recognisers.

2.4 Data Mining Tools and Techniques

The process of data analysis can be carried out in many ways. Other than writing our own algorithm implementations, statistical and data mining tools are available that provide libraries of various algorithms that can be employed.

R (R Development Core Team 2006) is an example of such a statistical tool that can be used for data analysis. We have used R in our previous research developing text-shape dividers (Patel 2007). R is a powerful language and environment which has functions that extend beyond classifier training. Its main functions are based around data manipulation, calculation and graphical display.

Data mining tools, on the other hand, focus on the use of machine learning algorithms to search data for patterns (Witten et al. 2005). Machine learning uses many statistical theories: in fact, over history the fields of machine learning and statistics have shared similarities (Witten et al. 2005). For example, very similar techniques, such as classification trees and nearest-neighbour, have been developed in both fields quite independently (Witten et al. 2005). A simplified view of the difference between machine learning and statistics is given by Witten and Frank:

> *"If forced to point to a single difference of emphasis, it might be that statistics has been more concerned with testing hypotheses, whereas machine learning has been more concerned with formulating the process of generalisation as a search through possible hypotheses." (Witten et al. 2005)*

The process of generalisation refers to the search of a finite space (Witten et al. 2005). In this case, we are searching for patterns to predict which class a stroke belongs to - text or shape.

Weka (Witten et al. 2005) is an example of a data mining tool. Its main functions include classification, clustering and feature selection as well as pre-processing and providing visualisations of data. Weka has over 100 machine learning algorithm implementations that can be used to perform data analysis and to build, tune and test models. The types of algorithms available include: Bayesian classifiers, trees, rules, functions (including regression, Support Vector Machines and Neural Networks), lazy classifiers (including Nearest Neighbour classifiers), multiple instance learners and various meta-learning algorithms (including Boosting and ensembles).

There are two main parts to Weka's interface: the Explorer (shown in Figure 15) and Experimenter (shown in Figure 16). The Explorer's main purpose is for quick exploration of different algorithms on data. The Experimenter, on the other hand, is for running more formal experiments where various configurations can easily be compared to each other. It includes useful functions for running large scale experiments, such as connection to remote machines to run experiments in parallel and connecting to databases for writing and retrieving results efficiently.

In addition, there is the Knowledge Flow interface. This allows users to set up a workflow of functions to apply to data which is able to be repeated without manually invoking each function, as required in the Explorer interface. All functions can also be accessed from a command line interface.

As the software is free and open source, further additions and integration of already existing algorithms into other projects are possible. The project is platform independent as it is written in Java.

Weka is a widely used data mining tool in academia and in business settings (Hall et al. 2009). Recent statistics document almost 3000 subscribers to Weka's discussion forum[7] (Bouckaert et al. 2010) and over 1.4 million downloads of the software since April 2000 (Hall et al. 2009). In addition to the discussion forum, there is comprehensive documentation available, including a wiki[8], API[9] and text book (Witten et al. 2005).

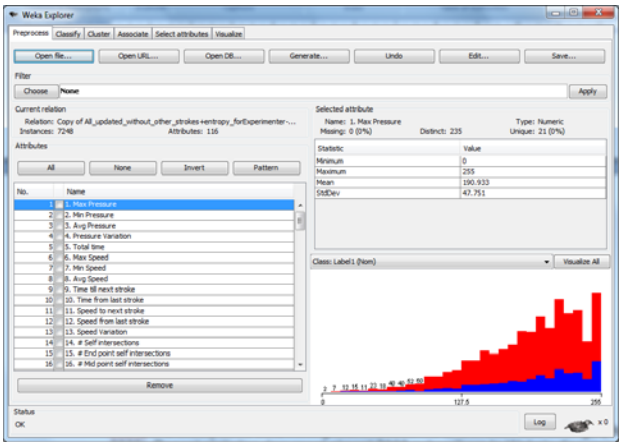


Figure 15 Weka Explorer Interface

[7] https://list.scms.waikato.ac.nz/mailman/listinfo/wekalist
[8] http://weka.wikispaces.com/
[9] http://weka.sourceforge.net/doc/

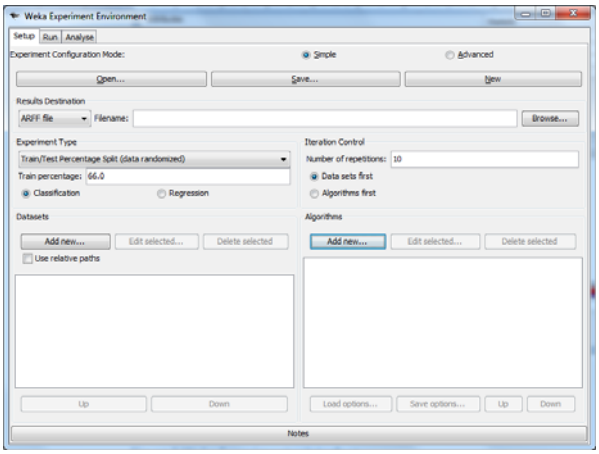


Figure 16 Weka Experimenter Interface

There are some other systems like Weka available, such as KNIME (Berthold et al. 2008), Orange (Demšar et al. 2004) and RapidMiner (Mierswa et al. 2006). KNIME provides a pipelining environment similar to Weka's Knowledge Flow interface. It uses Weka's algorithm implementations and R's statistical and graphical functions (Berthold et al. 2008). Orange (Demšar et al. 2004) is aimed more at assisting the development of new machine learning algorithms and exploratory data analysis. It is designed for researchers to quickly prototype new algorithms in Python script, while taking advantage of the already existing code libraries. It also provides a visual programming environment for data exploration, using the algorithms available. RapidMiner (Mierswa et al. 2006) is designed for rapid prototyping of knowledge discovery and data mining strategies. An operator tree is constructed using the GUI which specifies the functions that should be applied to the data for analysis. RapidMiner provides plug-ins that allow the use of Weka algorithms within the tool. The statistical tool R (R Development Core Team 2006) also has a package that allows the inclusion of Weka algorithms.

Equipped with the appropriate data mining tool, we must decide what techniques to use to analyse our data. The first decision is whether to employ supervised or unsupervised methods. Supervised methods require that data is labelled with information on the correct class of each instance. In our case, each stroke of the sketches we collect must be labelled as either text or shape. Unsupervised techniques do not require class information. Most techniques used by others for text-shape division are supervised as they use labelled data. One exception is Rodriguez et al's (2008) use of clustering in their first experiment. This experiment simply confirmed the presence of two clusters, text and shape, in the data. However, in subsequent experiments they used supervised algorithms, stating that this is a more reasonable approach to text-shape division.

In addition, as we are dealing with a nominal class (text or shape) as opposed to numerical, our data must be analysed using classification algorithms that can handle nominal classes. Regression is not suitable as it is traditionally used for numerical prediction.

Classification algorithms used in the past for text-shape division (as described in Section 2.1) include: Decision Trees (Shilman et al. 2004; Patel 2007; Zeleznik et al. 2008), Neural Networks (Mochida et al. 2003; Bishop et al. 2004; Mochida et al. 2004), linear classifiers (Plimmer et al. 2007; Rodríguez et al. 2008), rules (Machii et al. 1993; Jain et al. 2001; Mochida et al. 2003; Mochida et al. 2004; Avola et al. 2009; Bhat et al. 2009), Support Vector Machines (Ao et al. 2006; Waranusast et al. 2009), K-Nearest Neighbour (Waranusast et al. 2009), Boosting (Shilman et al. 2004; Peterson et al. 2010) and Hidden Markov Models (Bishop et al. 2004).

Although many methods of classification have been explored, there has been little comparison between them. Therefore we are unable to determine which algorithms produce the most accurate division of shapes and text. Weka has implementations of all of these algorithms, except for Hidden Markov Models and some linear classifiers. In addition to the easy to use functions that Weka provides, the wide range of algorithms available within the tool makes it an ideal choice for data analysis.

2.5 Summary

A review of related work has been presented, beginning with sketch recognition techniques. Very little attention has been given to the automatic division of text and shapes for sketched diagrams, where many recognition systems focus only on shape recognition or provide modal interfaces which are disruptive to the user. Many previous dividers are feature-based. They have been developed using a number of features and algorithms. However, no systematic analysis of algorithms has been carried out for text-shape division to determine the most suitable algorithm for this problem. Most concentrate on one or two algorithms and draw on very limited feature sets. In many cases the data used for training and testing is questionable, making it difficult to compare reported accuracy rates.

Numerous studies show the importance of good features to accurate recognition. Many feature sets have been proposed but no survey of existing features has been compiled as a resource for recogniser development.

Our review of data collection shows that the data publicly available is insufficient for a thorough analysis of features and algorithms for text-shape division. Data collection, labelling and dataset

generation is a time consuming and tedious task which is highly error prone when done manually. Numerous tools have been developed to assist in various stages of this process. However, none are able to support data collection, labelling and dataset generation with the use of a comprehensive feature library.

The background presented of data mining tools and techniques identify data mining as an ideal method of analysis for our research. Data mining employs machine learning algorithms to identify patterns in data: in our case we are searching for patterns to distinguish text from shapes. Various data mining tools are available to assist in this analysis. Weka has been identified as a widely used data mining tool with over 100 algorithm implementations available, comprehensive documentation and many useful functions to assist in data analysis.

This review leads us to conclude that in order to improve the accuracy of sketched diagram recognition we should perform a systematic analysis of algorithms with data mining techniques to determine the most optimal algorithms, using text-shape division as our example problem. As part of this research, we must search for quality features and compile a comprehensive feature library to use in our development of recognisers. More data is required for this analysis. In order to assist us in forming a data repository more efficiently, we will build our own data collection tool. The following chapter outlines the methodology used to achieve these goals.

Chapter 3

Methodology

This chapter describes the approach we have used to improve the recognition of hand-drawn diagrams; in particular, the division of writing and drawing. Following the literature review, we decided that the most likely way to improve recognition accuracy would be to data mine example diagrams. Data mining requires computable features, data and algorithms. First, the feature search to develop a library of digital ink features as a foundation for the analysis is discussed. Following this is a description of the method of data collection including the tools that need to be developed to assist in this process. The next section outlines the systematic approach taken to analyse the data and find patterns distinguishing writing from drawing using data mining techniques. Finally, the implementation and evaluation methods for the resulting text-shape dividers are described.

3.1 Feature Search

An important component of any sketch recogniser is the features measuring various characteristics of strokes. Many features are used by recognition systems as summarised in Section 2.2; however, there is little evidence to gauge how significant different features are to each recognition problem. We built a comprehensive library of digital ink features to use as a foundation for developing more accurate recognisers, and particularly to enable us to distinguish between writing and drawing ink.

To build this library, we conducted a feature search. Firstly, features were sought from previous work in sketch recognition. We were also interested in identifying new features that could be used to improve text-shape division. Workshops were run within the Computer Science department to elicit new features. A special focus was given to temporal and spatial context, as we have identified this to be a promising area in our previous work (Patel et al. 2007). Once we had a first prototype of a general divider we looked at common misclassifications and formulated new features to address those occurrences.

As we compiled this comprehensive feature library, each feature was implemented within the data management tool described in Chapter 5. This enables feature measurements to be taken on any diagrams collected with the tool and datasets to be generated automatically for later data analysis.

With such a comprehensive library, it is useful to organise the features into categories. A taxonomy was developed to complement the library – organising things into categories and naming the groups can help us to think about these features in new ways. Using an approach derived from grounded theory (Glaser et al. 1967), the features were first grouped into those sharing similar characteristics of ink. Once groups are formed, category names are assigned to each group according to the types of features that belong to that group. By categorising features in this way we try not to fit features into a particular group but to form the group around the features that share similarities.

When we examine the subsets chosen by data mining methods, having this taxonomy to refer to is a valuable tool. Observing which groups are more important helps us gain a more intuitive understanding of the features that are significant to the problem rather than overwhelming us with individual feature's characteristics. With this knowledge, we learn more about the types of features that are important to the problem at hand, thereby helping us to better solve problems in the future. It may also help us identify characteristics of ink that are not measured and thus devise new features.

3.2 Data Collection

Large amounts of data are needed in order to improve the accuracy of sketch recognition algorithms. As discussed in Section 2.3, when this project began, to our knowledge there were only four small repositories of digital ink data publicly available (Hse et al. 2004; Oltmans et al. 2004; Alvarado et al. 2007; Patel 2007). Most of this data did not include text or held some biases related to the method of data collection, so a new corpus was constructed. In order to compile such a corpus in an efficient manner, it was clear that a tool to collect and manage this data was essential. No data management tools for digital ink were then available that included collection, labelling and automatic dataset generation; so a new tool was built to perform all these functions.

To assist in the design process for this software, three basic design tools were employed: scenarios, lo-fidelity prototypes (Dix et al. 2004) and UML class diagrams. Writing scenarios for typical uses for the tool assisted in determining its key requirements. Also, lo-fidelity prototypes were used to construct preliminary user interface designs. Finally UML class diagrams were used to define the architecture of the data structures to be constructed in the software. We also drew on previous knowledge and

experience from past projects, where digital ink data was collected, labelled and statistically analysed without the use of any specialised tools (Patel 2007).

A usability evaluation was performed to establish how intuitive the interface is to use and to gauge the efficiency of using such a tool. There are two types of users for our system: study participants, who use the tool to draw diagrams for a dataset, and expert users who use the tool to collect, label and manage data for their own research. For participants, we want to ensure that the tool is easy and intuitive to use for drawing their diagrams. Usability experiments were conducted where participants were asked to complete several tasks using the software. They were observed during the experiment and then asked to complete a post-task questionnaire and were informally interviewed on their user experience. For expert users, the tool makes the process of data collection, labelling and dataset generation as efficient as possible. The time taken to label and generate a complex dataset using our tool was compared to previous experiences of performing these tasks without such assistance.

Once completed, the tool was used to collect, label and manage a large corpus of digital ink data for analysis and development of sketch recognition algorithms. At the beginning of this project there was limited knowledge of best data collection practice: how digital ink data should be collected, what should be collected and how much to collect. From previous experience (Patel 2007) we knew that the type of data that is collected must be carefully planned so that a wide variety of drawing and writing combinations and forms are represented in the dataset. It is also very important to ensure that the data is realistic and reflects true drawing styles so as to form an accurate basis for the analysis. Several data collection exercises were carried out where datasets were collected for the purpose of training and testing recognition algorithms: the training and test data must be different to provide a fair test of the algorithms. Example diagram domains include organisation charts, user interfaces and mind-maps.

3.3 Data Analysis

Once sketches were collected and labelled, they were converted into a dataset of feature vectors using the compiled feature library for subsequent data analysis.

Data mining techniques were employed to perform the analysis. Data mining uses machine learning algorithms to search data for patterns (Witten et al. 2005); in this case, we are searching for patterns to predict which class a stroke belongs to - text or shape.

Weka (Witten et al. 2005), an open source data mining tool, was chosen to perform the analysis. Weka has a large number of machine learning algorithms that are used to perform our data analysis and to build, tune and test classifier models for recognisers. In addition, it has advanced functions making the use of remote machines, for parallel processing, and databases, for writing and retrieving results, a simplified task. These features make Weka an ideal tool for this problem. This tool is described in more detail in Section 2.4.

Our analysis was not an exhaustive search of all algorithms and variants of algorithms. Weka has over 100 classification algorithms available, and each algorithm has numerous parameters that can be tested. An exhaustive search would be impossible, given the size of the search space, the computational requirements of such a task and the time available, even with the use of a powerful computational cluster. We began with a large number of algorithms and systematically reduced this list until the best performing classifiers were found for text-shape division. The steps involved in the analysis process are shown in Figure 17.

Figure 17 Process Diagram of Analysis

A preliminary investigation formed the first stage of the analysis. This involved running initial experiments on a large range of data mining algorithms, including algorithms that have been used in previous work for sketch recognition. After running these tests, the list of potential data mining algorithms was narrowed down to those which were most promising and were worthy of further investigation.

With this set of promising algorithms stage two of the analysis began. For this stage, each algorithm was tuned to determine the optimal parameters for the text-shape division problem. Important parameters were identified so that a range of values could be tested in order to optimise each algorithm.

The next stage of analysis involved experimenting with feature selection algorithms to reduce the feature library so that only the most significant features were used in the trained divider. The goal of this stage was to improve the accuracy of the models built with the full feature library as some features may be redundant or even detrimental to recognition accuracy.

For the fourth stage, ensemble algorithms which combine two or more methods were investigated. A sub-problem of using this technique is how to pick a good combination of algorithms. Algorithms were

ranked according to their classification performance at the current stage. The top-ranked algorithms were then combined in an ensemble. Each algorithm can also be evaluated to determine its strengths and weaknesses for each class. With this knowledge, ensembles were built to maximise the potential classification accuracy for each class. Using these techniques, earlier results may be improved.

Finally we conducted a second round of analysis. We evaluated the performance of the divider models built to identify common misclassifications and searched for additional features to correct these problem areas. For example, if small rectangles were commonly classified as text rather than shapes then we would search for new features to help correctly classify these rectangle strokes as "shapes". With the extended feature library, the models were re-trained and tested.

By the end of this data analysis, we achieved our goal to have highly accurate text-shape dividers for general diagrams.

3.4 Evaluation

A comparative evaluation was performed to determine whether the new dividers are more accurate than existing dividers. The best algorithms resulting from the analysis were implemented into DataManager's Evaluator (Schmieder et al. 2009). Existing dividers, Entropy (Bhat et al. 2009), Divider 2007 (our divider from previous work) (Patel et al. 2007) and the Microsoft Ink Analysis Divider (Microsoft Corporation 2008) were implemented alongside the new dividers to provide a comparison of performance.

A new dataset of sketches was collected as a test set for the evaluation. Data from other research groups was also used, including the data collected by Bhat et al (2009) when training the Entropy divider. It was important to evaluate the recognisers using data collected by others to confirm that the accuracy of our divider is independent of the data that we collect.

Based on the evaluation results, the value of domain-specific dividers was investigated. Dividers were built specifically for a particular domain, for example for Euler diagrams, and compared to the general dividers for performance. Domains with special characteristics may benefit from such treatment.

By following the process described here, beginning with a feature search, then data collection and analysis and finally an evaluation, our goal was to improve the accuracy of text-shape dividers as a first step to improving sketched diagram recognition. The discussion in Chapter 8 reflects on the success of

each stage in this process. The next chapter describes the first stage of our investigation, the feature search.

Chapter 4

Feature Search

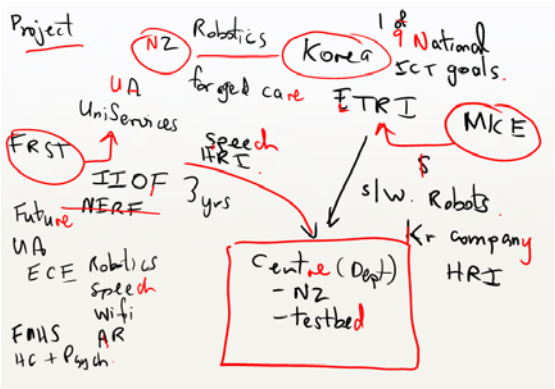
Given that features are a central element to sketch recognition, we believe that a library of ink features is essential for the development of recognition techniques. As a library suitable for our research did not exist previously, we decided to build a library and use this to facilitate the construction of recognisers. Without input such as this, research and development on any new feature-based classification algorithm would be very difficult.

A base for this library comes from the feature set developed during our previous work (Patel et al. 2007) of 46 ink features. This previous set was assembled from related work in sketch recognition and included some of our own additions. The features measure many elements of an ink stroke including pressure, time, intersections, size, curvature, tablet OS recognition values and inter-stroke gaps (Patel et al. 2007).

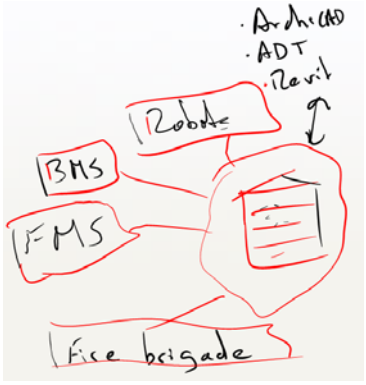
Additional ink features have been found during our review of a wide range of sketch recognition literature. These features were then added to the library. In terms of their implementation, many features documented in related work do not provide sufficient detail for accurate re-implementation. Therefore, we have had to make our own interpretations based on the information given.

After compiling features used in other research, we formulated our own new ink features specifically for the text-shape divider problem. To do this, we have looked at common misclassifications occurring for divider algorithms and formulated features to detect such problematic strokes. We also ran a series of workshops to try to elicit new feature ideas from others working in related Computer Science fields such as Computer Vision research.

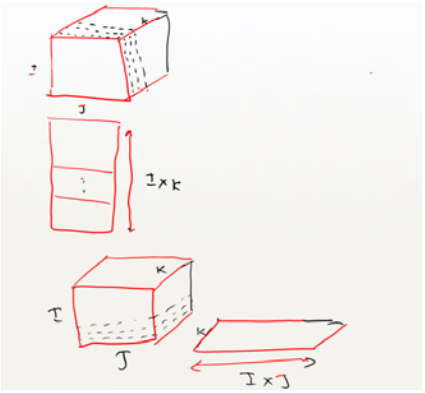
Three one-hour workshops within the University of Auckland Computer Science department were run, each with 15-20 people participating. Participants were asked to draw a diagram, using DataManager on a Tablet PC, to explain something they were working on to others in their group. When everyone in the group had completed this exercise, they were asked to use our previous divider (Patel et al. 2007),



a)



b)



c)

Figure 18 Example Diagrams Drawn by Participants in Workshops. Strokes in red are classified by our previous divider (Patel et al. 2007) as shapes, and strokes in black are classified as text.

which is built into DataManager's labelling interface, to classify the strokes in the diagrams just drawn as text or shapes. At this point, participants had observed how diagrams are drawn in real situations and saw the results of our previous divider to help them understand the common misclassifications that occur with such diagrams. The classification results also highlighted the similarities and differences between writing and drawing. Ethics approval was obtained from the University of Auckland Human Participants Ethics Committee for this and later studies. Further information on this ethics application can be found in Appendix E.

Example diagrams drawn by participants in the workshops are shown in Figure 18. Figure 18a has more text than shapes; shapes included are arrows, ellipses and a rectangle. Misclassifications of text mainly occur at the beginning or end of words. Most shapes have been classified correctly except for two arrows. In Figure 18b, misclassifications have been made at the beginnings of words as well as with some lines and arrows and bullet points. Misclassifications in Figure 18c include dashed lines, arrows, solid lines and parts of letters. Using these observations, participants were asked to brainstorm new features that could be added to a text-shape divider to improve recognition. Features that include more contextual information were discussed especially for identifying letters at beginnings and ends of

words. New features resulting from discussions during these workshops are described later in this chapter.

Our final feature library consists of 114 features: it is available with full implementation in DataManager[10].

In order to better understand the type of things that ink features are measuring, we developed a new taxonomy, shown in Table 4. In some cases a feature reflects more than one entry in the taxonomy. For example, the *entropy* feature is considered to be a measure of density, but it can also be a part of the divider results as it was developed as a one feature divider by Bhat et al (2009).

The taxonomy was developed using an approach derived from grounded theory (Glaser et al. 1967). They state that:

> *"In discovering theory, one generates conceptual categories or their properties from evidence, then the evidence from which the category emerged is used to illustrate the concept" (Glaser et al. 1967).*

This relates to our work as we developed the taxonomy by firstly grouping features measuring similar characteristics of ink. Once groups were formed, category names were assigned to each group according to the types of features that belonged to that group. By categorising features in this way we tried not to fit features into a particular group but to form the group around the features with shared similarities.

There are ten main categories. Two of these, spatial and temporal contexts, have subcategories, described in Table 5 and Table 6, which further describe the features related to these areas. Many of these subcategories are similar to the main categories listed. The key difference is that the features under spatial and temporal contexts measure characteristics of other strokes, rather than the current stroke. For example, features under spatial context-curvature measure the curvature of strokes that are close to the current stroke, whereas features under the main category of curvature only measure the curvature of the current stroke.

[10] www.cs.auckland.ac.nz/research/hci/downloads.

Category	Number of Features	Description
Curvature	23	These features measure various aspects of a strokes curvature commonly by calculating angles within the stroke.
Density	8	Density features measure the concentration of points in a stroke.
Direction	4	These features are related to the overall slope of a stroke. This is related to curvature but we have chosen to categorise these separately. Curvature features measure local curvature points on the stroke whereas direction features give a more global perspective of measurement.
Divider Results	2	These features provide the results of text-shape divider algorithms for the current stroke.
Intersections	3	Various types of intersections can be measured such as the intersections at stroke endpoints, in the middle of a stroke and self-intersections.
Pressure	4	These features measure the pressure applied to the screen for each point when drawing a stroke, including the average, maximum and minimum pressure in a stroke. Pressure is dependent on the capabilities of the hardware.
Size	19	Many measures of size exist ranging from the use of the strokes bounding box, to stroke lengths and the size of convex hulls.
Spatial context	23	Features measuring spatial context are within the following sub categories: curvature, density, divider results, intersections, location and size. Each subcategory contains measurements for strokes in close proximity to the current stroke. These subcategories are described further in Table 5.
Temporal context	22	Features measuring temporal context are within the following subcategories: curvature, density, divider results, length, location/distance and time/speed. Each subcategory contains measurements for strokes that come before or after the current stroke. These subcategories are described further in Table 6.
Time / speed	6	These dynamic features include the total, average, maximum and minimum times or speed for a stroke.

Table 4 Summary of Stroke Feature Categories.

Sub category	Number of Features	Description
Curvature	2	Features measure the curvature of strokes that are in close proximity to the current stroke.
Density	2	Features measure the density of strokes that are in close proximity to the current stroke.
Divider results	1	This feature provide the results of text/shape divider algorithms for strokes in close proximity to the current stroke.
Intersections	5	These features measure various characteristics of intersections of other strokes to the current stroke.
Location	9	These features measure various characteristics of the location of strokes that are in close proximity to the current stroke.
Size	4	These features measure the size of strokes that are in close proximity to the current stroke.

Table 5 Summary of Spatial Context Subcategories

Sub category	Number of Features	Description
Curvature	2	These feature measure the curvature of strokes drawn before and after the current stroke.
Density	2	These measure the density of strokes drawn before and after the current stroke.
Divider results	2	These features provide the results of text/shape dividers algorithms for strokes drawn before or after the current stroke.
Length	2	These features measure the length of strokes drawn before or after the current stroke.
Location / Distance	6	These measure the distance between the current stroke and strokes drawn before or afterwards.
Time/ Speed	8	These features measure the time and speed of strokes drawn before or after the current stroke.

Table 6 Summary of Temporal Context Subcategories

The remainder of the chapter describes features from related work and then provides details of the new features that we formulated by building on ideas gathered from past literature, brainstorming sessions during our workshops and our prior experience in building text-shape dividers.

4.1 Curvature

Using measures of stroke curvature is common to many sketch recognition algorithms. We identified 22 features quantifying various aspects of curvature in the reviewed literature, as listed in Table 7. They commonly rely on calculating angles within a stroke.

In comparison to the other feature categories, curvature has the largest number of features. This suggests two things: that there are many ways of quantifying curvature, and that they are considered important features for sketch recognition techniques. Previous work has found that measures of curvature are significant for distinguishing between writing and drawing (Patel et al. 2007).

The curvature of text is often greater than shapes. For example these features can show that in Figure 19 the writing stroke has a greater curvature than the shape stroke.

Figure 19 Text-Shape Example

Some features for curvature, such as the number of fragments, require strokes to be fragmented at possible corners. The implementation of these features uses the ShortStraw algorithm (Wolin et al. 2008) which finds possible corners in a given stroke in three steps. First, the points in the stroke are re-sampled so that there is a uniform distance between each point. Next, a bottom-up approach is used where the distance between a specified window of points is calculated throughout the whole stroke. This distance gets smaller as the points get closer and approaches a curve; the local minimum in this vicinity is considered as a possible corner.

The final step uses a top-down approach to remove false positives and find any missing corners. To do this, a line test is performed between the adjacent corners that have been found. If the line test fails, then there must exist an additional corner between these points. A more relaxed form of the bottom-up approach is used to locate this additional corner. Line tests between three known corners are also

performed to ensure that there are no false positives. Once the algorithm is complete, the stroke can be fragmented into lines using the location of each corner found.

This method of fragmentation is also used for some features in the direction and size categories.

Feature	Description	Origin
Number of Bezier cusps	Number of bezier cusps.	(Microsoft Corporation 2007; Patel et al. 2007)
Number of polyline cusps	Number of polyline cusps.	(Microsoft Corporation 2007; Patel et al. 2007)
∑ |angle at each point|	Sum of the absolute value of the angle at each point of the stroke.	(Rubine 1991)
∑(angle at each point)2	Sum of the squared value of the angle at each point of the stroke.	(Rubine 1991)
Absolute curve largest fragment	The total absolute curvature of the largest fragment. Fragments are found using ShortStraw (Wolin et al. 2008), as described in section 4.1.	(Bishop et al. 2004)
Angle of bounding box diagonal	Angle of the bounding box diagonal.	(Rubine 1991)
Average curvature	Average curvature (total angle / number of stroke points).	(Paulson et al. 2008b)
Cos from first to last point.	Cosine of the angle between the first and last point of the stroke.	(Rubine 1991)
Cos of initial angle	Cosine of the initial angle of the stroke.	(Rubine 1991)
Curviness	∑ absolute value of the angle at each stroke point below a 19^o threshold.	(Long et al. 2000)
Distance from first to last point	Distance from the first point of the stroke to the last point of the stroke	(Rubine 1991)
Least squares error	Orthogonal distance squared between the least squares fitted line and the stroke points / stroke length.	(Sezgin et al. 2001; Paulson et al. 2008b)
Maximum curvature	Maximum curvature of the stroke.	(Paulson et al. 2008b)
NDDE	Normalised distance between direction extremes.	(Paulson et al. 2008a)
Number of fragments	Number of fragments in a stroke (fragmented according it's to corners). Fragments are found using ShortStraw (Wolin et al. 2008), as described in section 4.1.	(Bishop et al. 2004)
Openness	Distance from the first to last point of the stroke / size of the stroke's bounding box.	(Long et al. 2000)
Overtracing	Total angle / 2π .	(Paulson et al. 2008a)
Sin from first to last point	Sine of the angle between the first and last point of the stroke.	(Rubine 1991)
Sin of initial angle	Sine of the initial angle of the stroke.	(Rubine 1991)
Total angle	Total angle traversed by the stroke.	(Rubine 1991)
Total angle and length ratio	Total angle / stroke length.	(Long et al. 2000)
Total angle ratio	Total angle / ∑ |angle at each point|.	(Long et al. 2000)

Table 7 Curvature Features from Literature

4.2 Density

Eight features were found for measuring density in previous sketch recognition research; they are described in Table 8. Density features measure the concentration of points in a stroke. For example, in Figure 19 the writing has a higher density of points than the rectangle when compared with their bounding box; in general the density of writing is expected to be greater than drawings.

Feature	Description	Origin
Amount of ink inside	Amount of ink inside the strokes bounding box (count the number of points inside the bounding box	(Young 2005)
Density 1	Stroke length / distance between first & last point.	(Long et al. 2000)
Density 2	Stroke length / area of bounding box.	(Long et al. 2000)
Length ratio	Cumulative distance between stroke points/ length from start to end point of a stroke.	Adapted from (Rubine 1991)
Length:perimeter ratio	Stroke length / perimeter of the stroke's convex hull.	(Fonseca et al. 2001; Fonseca et al. 2002)
Point ratio	Number of points in the stroke's convex hull / number of points in the stroke.	(Leung et al. 2002)
Total length/bounding box diagonal length	Length of the stroke divided by the length of the bounding box diagonal.	(Young 2005)
Entropy	See Section 2.1 for a full description of entropy	(Bhat et al. 2009)

Table 8 Density Feature from Literature

4.3 Direction

The four features described in Table 9 measure the overall slope of a stroke as illustrated in Figure 20. Direction is related to curvature features – we have chosen to categorise these separately as curvature features measure local curvature points on the stroke whereas direction features give a more global perspective of measurement for a collection of ink.

Figure 20 Direction Example

Feature	Description	Origin
DCR	Maximum change in direction / average change in direction.	(Paulson et al. 2008a)
Direction	Direction of the stroke (Eigenvector of the largest Eigen value)	(Bishop et al. 2004)
Eigen value ratio	The largest Eigen value/ smallest Eigen value.	(Bishop et al. 2004)
Largest fragment direction	Direction of largest fragment (eigenvector of the largest Eigen value). Fragments are found using ShortStraw (Wolin et al. 2008), as described in Section 4.1	(Bishop et al. 2004)

Table 9 Direction Features from Literature

4.4 Intersections

Various types of intersections can be measured, such as the intersections at stroke endpoints, in the middle of a stroke and self intersections. The diagram in Figure 21 shows intersections at the end and middle of strokes. We expect text strokes to have more intersections than shapes.

Figure 21. Intersection Example

Features	Description	Origin
Number of end point self intersections	Number of self intersections at the endpoints of the stroke.	Adapted from (Qin 2005; Patel et al. 2007)
Number of other self intersections	Number of self intersections that are not at the stroke's endpoint.	Adapted from (Qin 2005; Patel et al. 2007)
Number of self intersections	Number of points where the stroke intersects itself.	Adapted from (Qin 2005; Patel et al. 2007)

Table 10 Intersection Features from Literature

4.5 Pressure

The four features in Table 11 measure the pressure applied to the touch or tablet screen for each point when drawing a stroke, including the average, maximum and minimum pressure in a stroke. The availability of pressure data is dependent on the capabilities of the hardware. Most Tablet PC's are now pressure sensitive.

Feature	Description	Origin
Average pressure	Mean average pressure of the stroke.	Adapted from (Nakai et al. 2002)
Maximum pressure	Maximum pressure value for the stroke.	Adapted from (Nakai et al. 2002)
Minimum pressure	Minimum pressure value for the stroke.	Adapted from (Nakai et al. 2002)
Number of pressure minima	Number of minima in pressure values for the stroke.	(Patel et al. 2007)

Table 11 Pressure Features from Literature

4.6 Size

Many measures of size exist ranging from the use of a stroke's bounding box, to stroke lengths and the size of convex hulls. Size is a popular feature used by many recognition strategies. This is reflected by the 19 size features that were found in the literature, described in Table 12. We believe that the size of writing strokes in diagrams is usually smaller than drawing strokes as illustrated in Figure 18.

Feature	Description	Origin
Arc fit radius	The radius of an arc fitted to the stroke.	(Paulson et al. 2008b)
Aspect	$\|45\pi/180$ – angle of the bounding box diagonal $\|$.	(Long et al. 2000)
Bounding box area	Area of the bounding box of the stroke.	Adapted from (Fonseca et al. 2002; Hammond et al. 2002)
Bounding box diagonal length	Length of the bounding box diagonal line.	(Rubine 1991)
Bounding box height	Height of the bounding box of the stroke.	Adapted from (Fonseca et al. 2002; Hammond et al. 2002)
Bounding box width	Width of the bounding box of the stroke.	Adapted from (Fonseca et al. 2002; Hammond et al. 2002)
Convex hull area ratio	Ratio of area of convex hull to area of the enclosing rectangle of the stroke.	(Fonseca et al. 2002)
Enclosing rectangle ratio	Ratio of strokes enclosing rectangle width to height.	(Fonseca et al. 2002)
Largest fragment length	Arc length of the stroke's largest fragment. Fragments are found using ShortStraw (Wolin et al. 2008), as described in section 4.1	(Bishop et al. 2004)
Length	Total length of the stroke.	(Rubine 1991)
Log area	Log of the stroke's bounding box area.	(Long et al. 2000)
Log aspect	Log of the aspect feature.	(Long et al. 2000)
Log length	Log of the total length of the stroke.	(Machii et al. 1993; Long et al. 2000)
Log longest side rectangle	Log of the length of the longest side of the stroke's bounding box.	(Machii et al. 1993)
Long side of enclosing rectangle of largest fragment	The longest length of the largest fragment's enclosing rectangle. Fragments are found using ShortStraw (Wolin et al. 2008), as described in section 4.1	(Bishop et al. 2004)
Perimeter efficiency	$2 \sqrt{}$ (π stroke's convex hull area) / stroke's convex hull perimeter.	(Leung et al. 2002)
Perimeter to area	Ratio of perimeter to area of the stroke's convex hull	(Fonseca et al. 2002)
Thinness ratio	Perimeter^2 of stroke's convex hull / area of stroke's convex hull	(Fonseca et al. 2001; Fonseca et al. 2002)
Width to height ratio	Ratio of the stroke's bounding box width to height.	Adapted from (Fonseca et al. 2002)

Table 12 Size Features from Literature

4.7 Spatial Context

Features in the spatial context category measure aspects of strokes in close proximity to the current stroke. We believe that strokes with similar feature measurements to those that are close by are more likely to be from the same class (text or shape) so these features have the potential to provide recognisers with very valuable contextual information. The large number of these features in the reviewed literature suggests the perceived importance of these features to sketch recognition.

There are six subcategories of spatial context: curvature, density, divider results, intersections, location and size, as described in Table 5; features found in the literature for each subcategory are listed in Table 13. Features under the divider results subcategory are new and they are listed in 4.11.3.

Feature	Description	Origin
Curvature		
Average curvature of close end point strokes	Average curvature (using total angle) of other strokes with endpoints close to current stroke.	(Ao et al. 2007)
Average curvature of close strokes	Average curvature (using total angle) of other strokes close to current stroke.	(Ao et al. 2007)
Density		
Average density of close end point strokes	Average density (stroke length / bounding box diagonal length) of other strokes close at end points to the current stroke.	(Ao et al. 2007)
Average density of close strokes	Average density (stroke length / bounding box diagonal length) of other strokes close to the current stroke.	(Ao et al. 2007)
Intersections		
Number of other intersections	Number of points of intersection of the current stroke with other strokes (excluding self intersections).	Adapted from (Calhoun et al. 2002)
Number of other strokes intersecting	Number of other strokes that intersect the current stroke (excluding itself).	Adapted from (Fonseca et al. 2002; Hammond et al. 2002)
Total number of intersections	Total number of intersections (includes self intersections).	Adapted from (Calhoun et al. 2002)
Total number of strokes intersecting	Number of strokes that intersect the current stroke (including itself)	(Fonseca et al. 2002; Hammond et al. 2002)
Location		
Number of close end point strokes	Number of other strokes whose endpoints are close to end points of the current stroke.	(Ao et al. 2007)
Number of close strokes	The number of other close strokes to the current stroke.	(Ao et al. 2007)
Number of vertically close strokes	The number of other strokes vertically close to the current stroke.	(Ao et al. 2007)
Size		
Average length of close end point strokes	Average length of other strokes close to end points of the current stroke.	(Ao et al. 2007)
Average length of close strokes	Average length of other strokes close to the current stroke.	(Ao et al. 2007)

Table 13 Spatial Context Features from Literature

4.8 Temporal Context

Features measuring temporal context contain measurements for strokes that are drawn immediately before or after the current stroke. Strokes drawn in succession are more likely to be from the same class except when objects are interspersed (Sezgin et al. 2007).

There are six subcategories of temporal context: curvature, density, divider results, length, location/distance and time/speed as described in Table 6; features found in the literature for each subcategory are listed in Table 14. Features under the divider results subcategory are new and they are listed in 4.11.3.

Feature	Description	Origin
Curvature		
Curvature of next stroke	Total angle of next stroke.	Adapted from (Rubine 1991; Ao et al. 2007)
Curvature of previous stroke	Total angle of previous stroke.	Adapted from (Rubine 1991; Ao et al. 2007)
Density		
Density of next stroke	Length of the next stroke divided by the length of the next stroke's bounding box diagonal.	Adapted from (Rubine 1991; Ao et al. 2007)
Density of previous stroke	Length of the previous stroke divided by the length of the previous stroke's bounding box diagonal.	Adapted from (Rubine 1991; Ao et al. 2007)
Length		
Length of next stroke	Total length of next stroke.	(Ao et al. 2007)
Length of previous stroke	Total length of previous stroke.	(Ao et al. 2007)
Location/Distance		
Distance from last stroke	Distance between current stroke and previous stroke.	Adapted from (Young 2005; Patel et al. 2007)
Distance to next stroke	Distance between current stroke and next stroke.	Adapted from (Young 2005; Patel et al. 2007)
X difference between strokes	Difference in X co-ordinate between current stroke and next.	(Bishop et al. 2004)
X start point difference	Difference in starting X coordinates of current stroke to next stroke.	(Bishop et al. 2004)
Y difference between strokes	Difference in Y co-ordinate between current stroke and next.	(Bishop et al. 2004)
Y start point difference	Difference in starting Y coordinates of current stroke to next stroke.	(Bishop et al. 2004)
Time/Speed		
Log start time from previous	Log of time from start of previous stroke to start of current stroke.	(Bishop et al. 2004)
Log start time to next	Log of time from start of current stroke to start of the next stroke.	(Bishop et al. 2004)
Log time difference from previous	Log of the time between the start of the current and end of the previous stroke.	(Bishop et al. 2004)
Log time difference to next	Log of the time between the end of the current stroke and the start of the next stroke.	(Bishop et al. 2004)
Speed from last stroke	Speed (distance/time) between current stroke and previous stroke.	(Patel et al. 2007)
Speed to next stroke	Speed (distance/time) between current stroke and next stroke.	(Patel et al. 2007)
Time from last stroke	The time between the start of the current stroke and the end of the previous stroke.	(Patel et al. 2007)
Time till next stroke	The time between the end of the current stroke and the start of the next stroke.	(Patel et al. 2007)

Table 14 Temporal Context Features from Literature

4.9 Time/Speed

Features for time and speed from the literature are listed in Table 15. These features include the total, average, maximum and minimum times or speed for a stroke. We believe that users often write more quickly than they draw.

Feature	Description	Origin
Number of speed minima	Number of extreme minima in the speed values for the stroke.	Adapted from(Sezgin et al. 2001; Patel et al. 2007)
Average speed	Mean average speed when drawing the stroke.	Adapted from (Rubine 1991; Patel et al. 2007)
Maximum speed	Maximum speed when drawing the stroke.	Adapted from (Rubine 1991; Patel et al. 2007)
Maximum speed squared	Maximum speed of the stroke squared.	(Rubine 1991)
Minimum speed	Minimum speed when drawing the stroke.	Adapted from (Rubine 1991; Patel et al. 2007)
Total duration	Total duration of the stroke from pen up to pen down.	(Rubine 1991)

Table 15 Time and Speed Features from Literature

4.10 Divider Results

Features in this category provide the results of text-shape divider algorithms on a stroke. Results from previous dividers on a stroke can act as a first parse of recognition, similar to a voting scheme. The feature in Table 16, from our previous feature library (Patel et al. 2007), uses the Tablet PC Operating System text recogniser to find the probability that a given stroke is text. Strokes can have a strong, intermediate or poor probability of being text. Additional features using divider results are described in the next section as they are new features.

Feature	Description	Origin
Tablet OS text probability	Tablet OS text recogniser probability of the stroke being text.	(Microsoft Corporation 2005) (Patel et al. 2007)

Table 16 Divider Features

4.11 New Features

New features were developed as a result of discussions in workshops, knowledge of related work, prior experience in this field and examination of common misclassifications of our text-shape divider from previous work (Patel et al. 2007). Spatial context was identified as an area with great potential. This was supported by observations from participants in our workshops where the previous divider was frequently misclassifying parts at the beginnings and ends of words and parts of shapes as shown in Figure 18. Therefore, the majority of the new features measure characteristics of spatial context. In

addition, features using our divider from previous work as a pre-parse of strokes have been added. All new features are summarised in Table 17 and described in more detail in the remainder of this section.

Feature	Description
Curvature	
Number of direction changes	Number of changes in the direction of a stroke.
Divider Results	
Divider 2007 result	Results of our text/shape divider on the current stroke. (Patel et al. 2007)
Spatial Context	
Spatial Context: Location	
Number of strokes horizontally close	Number of strokes horizontally close to current stroke.
Number of strokes on same horizontal plane	The number of strokes on the same horizontal plane as the current stroke.
Number of strokes contained	Number of strokes contained inside the current stroke.
Is contained	The stroke is contained by another stroke.
Smallest distance between strokes from end point	The smallest distance to another stroke from the current stroke's end point.
Smallest distance between strokes from start point	The smallest distance to another stroke from the current stroke's start point.
Spatial Context: Size	
Number of strokes of similar height	Number of strokes of similar height to current stroke.
Length of closest stroke	The length of the closest stroke to the current stroke where the closest stroke is found by measuring distance between the middle of each strokes' bounding box to the current stroke.
Spatial Context: Intersections	
Number of strokes vertically overlapping	The number of strokes vertically overlapping the current stroke.
Spatial Context: Divider Results	
Divider 2007 closest stroke	Results of our text/shape divider for the closest stroke to the current. (Patel et al. 2007)
Temporal Context	
Temporal Context: Divider Results	
Divider 2007 next stroke	Results of our text/shape divider for the next stroke. (Patel et al. 2007)
Divider 2007 previous stroke	Results of our text/shape divider for the previous stroke. (Patel et al. 2007)

Table 17 Summary of New Features

4.11.1 Curvature

The following feature is a newly formulated way of measuring stroke curvature.

Number of Direction Changes

This feature resulted from discussions in the workshops we conducted. It counts the number of times a stroke changes direction by calculating the gradient between successive points in the stroke and detecting the number of times this gradient changes. The changes are only counted when there is a shift between positive, negative, vertical or horizontal gradients, i.e. if the stroke has a positive gradient of 2 and changes to a positive gradient of 3, this change is not counted as a change in direction.

More changes in direction are expected in text strokes than shape strokes as writing generally has more curves. For example the number of changes in direction for the writing shown in Figure 22 is greater than the number of changes in direction for the rectangle.

Figure 22 Number of Direction Changes Example

4.11.2 Spatial Context

The following features measure characteristics of spatial context with respect to stroke location, size and intersections.

Location: Number of Strokes Horizontally Close

This feature counts the number of strokes that are horizontally close to the current stroke. As letters and words in the English language are typically written horizontally we expect that this feature will measure important information for horizontally close strokes in writing.

To calculate this feature, the horizontal gap between two strokes must first be found, based on their bounding box. This gap must be below a specified threshold for these strokes to be considered horizontally close. Our threshold is 0.6 times the maximum bounding box height of the two strokes that are being tested. This threshold value was obtained by trial and error by observing various cases of strokes that were considered to be horizontally close and not horizontally close from a visual perspective. The maximum bounding box height is used in this threshold to ensure that the threshold is relative to the size of the strokes rather than having a fixed value.

If the gap between the bounding boxes is less than the threshold, the strokes are considered horizontally close. This includes strokes that overlap horizontally. The strokes do not have to sit on the same baseline to be considered horizontally close: for example, the arrow in Figure 23 does not share the same baseline as the two rectangles but they are still considered horizontally close strokes. The letters ‘a’ and ‘b’ are also considered to be horizontally close in Figure 23.

Figure 24 shows another example of horizontally close strokes. The first stroke, “Au” has one horizontally close stroke, “ck”. To find this value, firstly the maximum bounding box height of this pair of strokes is found, which is 1.2 cm. Next, the threshold for the maximum gap between these strokes is calculated, which is 1.2 * 0.6 which equals 0.72cm. The bounding boxes are overlapping, resulting in a gap that is less than the threshold of 0.72cm. Thus they are horizontally close. The same calculation

between “Au” and “la” result in the same threshold of 0.72cm generated, as “Au” has the largest bounding box height. However the gap between the bounding boxes is greater than the threshold, therefore “Au” and “la” are not horizontally close strokes. Note that this calculation is in pixels in the implementation.

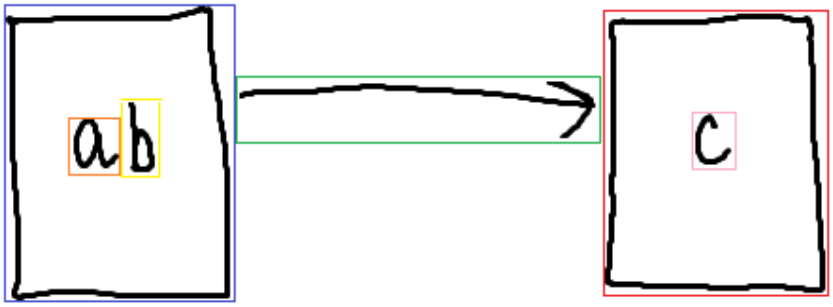


Figure 23 Horizontally Close Strokes.

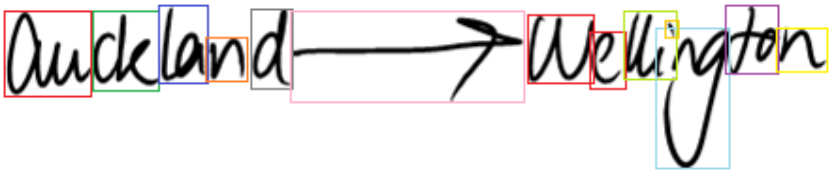


Figure 24 Example 2 of Horizontally Close Strokes and Strokes on the Same Horizontal Plane.

Location: Number of Strokes on Same Horizontal Plane

This feature measures the number of strokes on the same horizontal plane as the current stroke. It is similar to the feature presented above, except that close strokes must sit on the same horizontal baseline as the current stroke to be counted.

This feature is also calculated using the bounding boxes of strokes. A threshold is calculated as 0.3 * the maximum bounding box height of the two strokes that are being tested. This threshold value was obtained in the same way as the previous feature, by observing various cases of strokes that were and were not considered to be on the same horizontal plane as each other from a visual perspective. The maximum bounding box height is used in this threshold to ensure that the threshold is relative to the size of the strokes rather than a fixed value.

Three measures are then compared to this threshold. Firstly, the gap between the bottoms of the bounding boxes, then the gap between the tops of the bounding boxes and finally the gap between the top of one bounding box to the middle of the other stroke’s bounding box (this tests for the difference

between tall and short letters e.g. between ‘a’, ‘b’ or ‘g’). If any of the three comparisons above are less than the threshold, the two strokes are considered to be on the same horizontal plane.

In Figure 24 all strokes share the same horizontal plane, for example “Au” has 12 strokes on the same horizontal plane. In Figure 25 only strokes in the same row share a horizontal plane. The letter ‘e’ in “one” is on the same horizontal plane as three strokes. When compared to all other letters in the same row, ‘e’ has the largest bounding box of 1.0cm. Therefore the threshold for these comparisons is calculated as 0.3 * 1.0, which is 0.3cm. Using this threshold first the gap between the bottom of the bounding boxes can be made. Considering just ‘n’ and ‘e’, the gap between the bottom of their bounding boxes is approximately 0.2cm: this is less than the threshold of 0.3cm so they are considered to be sitting on the same horizontal plane.

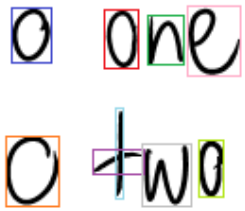


Figure 25 Strokes on the Same Horizontal Plane

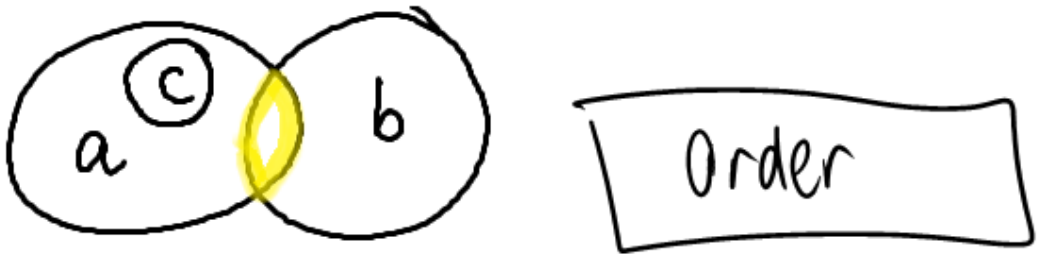
An example of strokes not on the same horizontal plane is ‘e’ and ‘w’ from Figure 25. The same threshold as above is used, as ‘e’ has a larger bounding box height than ‘w’. The gap between the tops, bottoms and middles to tops are approximately 1.4cm, 1.4cm, 1.0 (middle of ‘e’ to top of ‘w’) and 1.9cm (middle of ‘w’ and top of ‘e’) respectively. None of these gaps are less than the threshold of 0.3cm: therefore, they are not on the same horizontal plane.

Location: Number of Strokes Contained

This feature measures the number of strokes contained by the current stroke: i.e. the current stroke is the container. We suspect that if there are strokes contained within a stroke, the current stroke is most likely to be a shape. Writing strokes do not typically contain other strokes.

The feature is calculated using the bounding boxes of two strokes. If the bounding box of the current stroke contains the bounding box of another stroke then the current stroke is considered to contain the other stroke. The circle on the left in Figure 26(a) contains three strokes and the circle on the right contains one stroke. Note that the parts of the two circles highlighted in yellow are not considered to be contained by another stroke as the whole stroke must lie within another stroke.

Figure 26(b) shows a common example of containment, where the rectangle contains five strokes. The letter strokes do not contain any other strokes.



a) b)

Figure 26 Examples of a Strokes Contained by Shape Strokes

Location: Is Contained

This feature establishes if the current stroke is located within another stroke, where another stroke acts as a container. It is more likely that text is contained by another stroke in diagrams as shape strokes often act as containers in node-edge type diagrams, although a shape could also be contained by another shape.

An example of text and shape strokes contained by other shape strokes is shown in Figure 26(a) where the circle labelled 'c' is contained by circle 'a' and all letter strokes are contained by shapes. In Figure 26(b) each of the letter strokes are contained by one stroke (the rectangle stroke), whereas the rectangle is not contained by any other stroke.

This feature is calculated using the same algorithm as the previous feature except that the current stroke and the other stroke swap places in the calculation to determine if the current stroke is contained by the other stroke.

Location: Smallest Distance between Strokes from End Point, and
Location: Smallest Distance between Strokes from Start Point

These features measure the smallest distance to another stroke from the current stroke's end point or start point. It is a variation of the distance to the next stroke and the distance from the previous stroke (in temporal context: location/ distance category) which was found to be useful in our previous writing drawing divider work (Patel et al. 2007).

These features are calculated by measuring the distance between the end point or start point of the current stroke to the end point and start point of all other strokes. The minimum distance found between strokes is returned. The distance between points is calculated using Equation 1 below:

Distance $(point_1, point_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

Where point1 is (x_1, y_1) and $point_2$ is (x_2, y_2)

Equation 1 Distance between Points

For example, in Figure 27, to calculate the smallest distance between strokes from the start point one, the distances from start point one to points three, four, five and six are calculated using Equation 1. The smallest distance found is to point four. To calculate the smallest distance between strokes from end point two the same distances are calculated and the smallest distance found is to point three.

Figure 27 Smallest Distance between Strokes from the Start/End Point of a Stroke Example.

Size: Number of Strokes of Similar Height

This feature measures the number of strokes of similar height to the current stroke. This can be used to detect a line of writing: if there are many strokes with a similar height, they are probably text strokes.

This feature is calculated using the bounding boxes of strokes. To decide if the current stroke is of a similar height to another stroke, the bounding boxes of the two strokes are first obtained. Then the height of the smaller bounding box is divided by the height of the larger bounding box. If this ratio is found to be larger than or equal to 0.5 then the strokes are considered to have a similar height. In other words, the smaller bounding box must be at least half the height of the larger bounding box. This threshold ensures that tall and short letters in the same word are still considered to have a similar height.

The writing strokes in Figure 28 are all considered to be strokes of a similar height. The bounding box of the tallest stroke in Figure 28, “all”, is approximately 1.8cm high. The other strokes, ‘s’ and ‘m’, are

approximately 1.5cm and 1.1cm respectively. These values are over half the height of the "all" stroke, therefore both 's' and 'm' are of a similar height to "all".

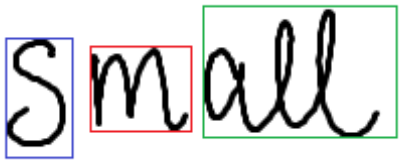


Figure 28 Strokes of a Similar Height Example.

Size: Length of closest stroke

This feature measures the length of the closest stroke to the current stroke. The closest stroke is found by measuring the distance between the middle of the bounding boxes of each stroke, using the formula shown in Equation 1. Then the length of this stroke is calculated as the sum of the distances between points in the stroke, also using Equation 1.

This feature may provide important information for short strokes that are part of words or shapes but are often misclassified because of their size. Taking the length of the closest stroke may help to correct such situations. For example, the arrowhead in Figure 29 may be such a stroke. The crosses drawn in Figure 29 show the approximate location of the middle of each stroke's bounding box. The closest stroke to the arrowhead based on the middle of the bounding boxes is the arrow shaft. The length of the arrow shaft is therefore calculated as the length of the closest stroke to the arrowhead.

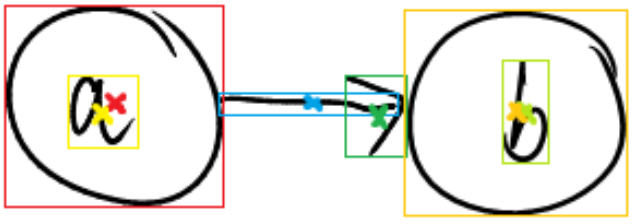


Figure 29 Length of Closest Stroke Example

Intersections: Number of Strokes Vertically Overlapping

This feature counts the number of strokes that vertically overlap the current stroke. We believe that shape strokes more frequently overlap in this way than text strokes.

This feature is calculated using the bounding boxes of two strokes. If the bottom (y coordinate) of bounding box 1 is greater than the top (y coordinate) of bounding box 2 and the bottom of bounding

box 2 is greater than the top of bounding box 1 then the strokes are considered as vertically overlapping.

In Figure 30, each rectangle has one vertically overlapping stroke and the arrow has two vertically overlapping strokes. For the first rectangle, the bottom of its bounding box is greater than the top of the arrow's bounding box and the bottom of the arrow's bounding box is greater than the top of the rectangle's bounding box; therefore they are vertically overlapping. However, when comparing the two rectangles, the bottom of the first rectangle is less than the top of the second rectangle, therefore they are not vertically overlapping.

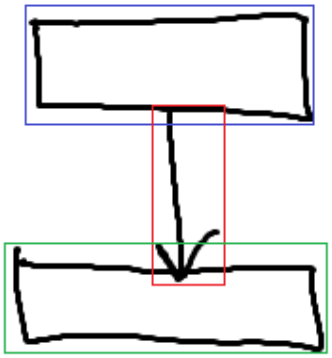


Figure 30 Vertically Overlapping Strokes Example

4.11.3 Divider Results

The following features use our text-shape divider from previous work (Patel et al. 2007) as a pre-parsing step of strokes. A detailed description of this divider can be found in Section 2.1.

Divider 2007 Result

This feature classifies the current strokes as text or shape based on our previous text-shape divider (Patel et al. 2007). Using this classification as a feature is similar to using a voting system where various classifiers' predictions are combined to form the final prediction. Figure 31 shows an example of strokes classified by this divider, where strokes in red are classified as text and strokes in blue are shapes.

Spatial Context: Divider 2007 Closest Stroke

This feature uses Divider 2007 (Patel et al. 2007) to classify the closest stroke to the current stroke. As it based on the closest stroke, this feature is categorised under spatial context: divider results. The closest stroke is found by measuring the distance between the middle of the bounding boxes of each stroke using the formula shown in Equation 1. We believe that strokes close to one another are frequently from the same class, especially for letters in the same word. Therefore, information regarding the classification of strokes close by can be valuable for classifying the current stroke.

For example the letter ‘e’ in Figure 31 is closest to ‘t’. If ‘t’ is correctly classified by Divider 2007 as a text stroke, this can strengthen the prediction accuracy of ‘e’.

Temporal Context: Divider 2007 Next Stroke, and
Temporal Context: Divider 2007 Previous Stroke

These features uses Divider 2007 (Patel et al. 2007) to classify the next and previous strokes to the current stroke. As the previous and next strokes are used here, these features are categorised under temporal context: divider results. The feature is based on the same reasoning as the previous feature: where strokes drawn in succession are more likely to be from the same class, the exception is when objects are interspersed (Sezgin et al. 2007).

For the example shown in Figure 31, if we assume that the rectangle was drawn first then the writing, the first letter ‘t’ has the next stroke as a text stroke and the previous stroke as a shape stroke.

Figure 31 Divider Results Example

4.12 Summary

A comprehensive feature library has been compiled consisting of 114 ink features. These features are from previous work in sketch recognition and include 14 of our own new features. To organise the feature library, we have developed a new taxonomy consisting of ten categories: curvature, density, direction, intersections, pressure, size, spatial context, temporal context, time/speed and divider results. Most of our new features measure aspects of spatial context.

This feature library is central to the further development of ink recognisers. Each feature is implemented in DataManager, our data collection tool, described in the following chapter. Using example sketches collected within DataManager, these features are used to automatically generate datasets composed of feature vectors for each stroke. The datasets can then be analysed using data mining techniques to build recognisers including text-shape dividers.

Chapter 5

Data Collection

There is a critical need for more rigorous analysis of sketch recognition algorithm performance and tuning. The development of high precision recognition techniques requires large amounts of digital ink data to aid the training and evaluation stages. In addition to quantity, the quality of this data is paramount to the success of algorithm development and therefore must be un-biased and representative of a wide range of diagram types and end user input. To accomplish this, we require a corpus of well-authored, labelled sketches to be assembled.

There is little ink data available that meet these criteria and there is also little support for obtaining such data. A tool that provides ease of unbiased data collection and management would allow us to construct a data repository more efficiently and therefore aid the development of recognition techniques; not only for ourselves but also for the sketch recognition community at large. To this end, we have developed a tool that enables efficient collection, labelling and management of ink data as well as automatic generation of datasets for analysis. This tool assists in the effective construction of a large database of sketches to aid the development of recognition techniques.

This chapter describes the development of DataManager, our sketch data management tool, and provides an overview of the training data collected using DataManager for the development of text-shape dividers.

5.1 DataManager Requirements

First, we consider the high level requirements of our data collection tool and expectations of its user interface. The essential requirements for the collection, labelling and dataset generation functions of the tool are then described.

There are two types of users of DataManager: researchers and participants. A researcher's primary goal is to construct a database of sketches for developing sketch recognition algorithms. The fundamental requirement of DataManager is that it minimises the time and effort a researcher must devote to the

tasks of ink data collection and management. In our previous work (Patel 2007) we labelled and constructed a dataset manually. This was a very time consuming and potentially error-prone, process taking more than three days to complete for a set of 1519 strokes. We want to minimise the time spent carrying out these tasks so our tool must provide more efficient support for data collection, labelling and dataset generation. We aim to automate as much of this process as possible.

A more efficient process of collecting and managing data enables researchers to collect larger amounts of data. Previously, due to the time required to complete the collection, labelling and dataset construction process, datasets were limited to a small number of sketches. Ideally, we would like to be able to supply our analysis techniques with as much data as possible to ensure that it is representative of as many styles of sketching as possible. In addition to providing a tool that minimises the time and effort required for data collection tasks, it must have a user interface that gives researchers the ability to manage and navigate large amounts of data with ease. It must also allow management of a large corpus of sketched content that can be shared with other researchers.

The second group of users is participants. These are users who provide data by drawing the sketches that the researcher requires. It is important that the tool is easy and intuitive for the participant to use when sketching. The tool must ensure as much as possible that the data collected from participants is free from bias. “Unbiased” means that the method of collection follows the natural practice of drawing diagrams as closely as possible so that the data obtained provides a true representation of typical diagrams.

In our previous work, when collecting data, participants were given pre-drawn examples of what they had to sketch: these examples were isolated components of diagrams rather than full diagrams. We believe that this data was biased for the following two reasons. The participants were copying from pre-drawn examples rather than constructing sketches on their own. We believe this affects the timing data associated with these sketches. Secondly, participants were asked to draw isolated components as opposed to constructing full diagrams, thus losing the temporal and spatial relationships between components in a diagram. Subsequent results (Field et al. 2009) have confirmed that these factors concerning data collection are important and significantly influence the accuracy of recognisers. To reduce the effect of these factors, it is important that the tool provides a written description of a full diagram for construction rather than a pre-drawn example that will be copied.

Labelled data is required when supervised learning techniques are used, as discussed in Section 2.4. Labelling sketches is a very time consuming task. One reason why researchers have preferred to collect

isolated components is that labelling is easier - the label for a shape can simply be inferred by the diagram name. This is in contrast to full diagrams where each stroke must be grouped and labelled (Field et al. 2009). Our tool must provide good support for labelling full diagrams, and one solution we explored is automatic labelling of diagrams. Automatic labels can be over-ridden by the user if they are incorrect.

The extensibility of the tool is also important so that the tool can be changed as new research emerges. One aspect of this is that each function of the tool should be fully extensible. For example, the labelling subsystem needs to be designed so that it can be easily replaced by a new labeller. Other examples of extensibility include supporting the addition of new ink features that users want to include in their datasets, and the addition of alternative dataset output formats.

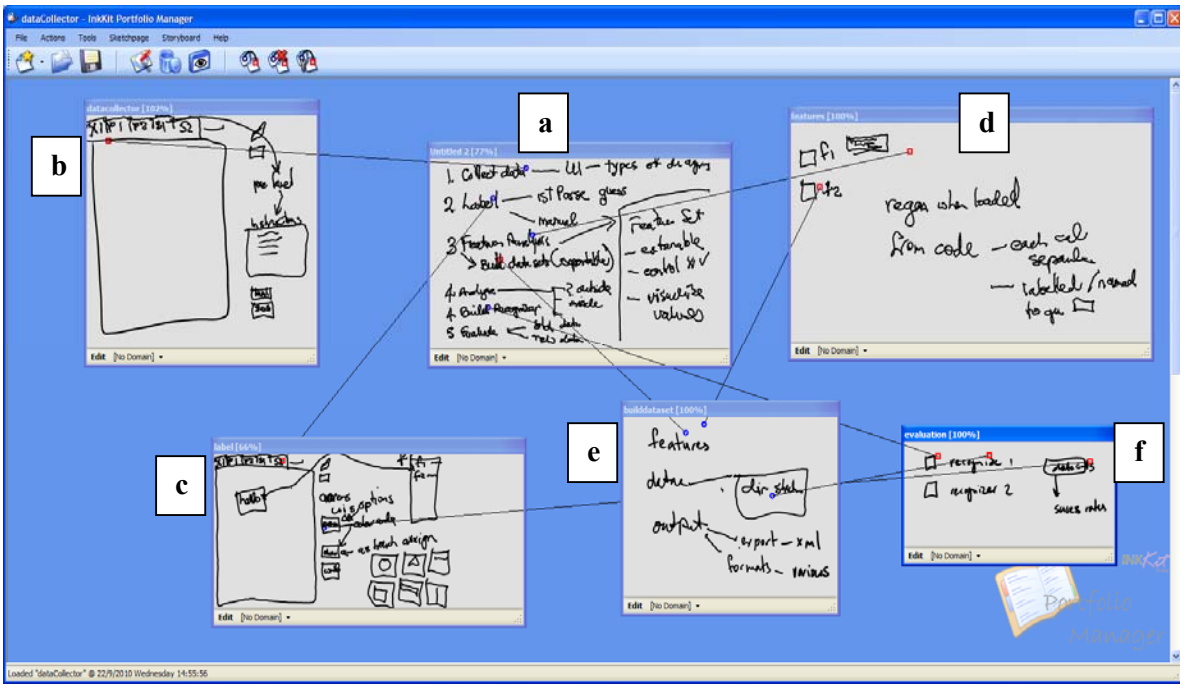


Figure 32 First Lo-fidelity Prototype

The main functional requirements of DataManager are to collect and label data, and to automatically generate datasets with the use of our feature library. In keeping with Human Computer Interaction practices (Dix et al. 2004), we first describe the system in terms of lo-fidelity prototypes and scenarios. Figure 32 shows the first lo-fidelity prototype that was constructed to design DataManager's interface. Figure 32a lists the main parts of DataManager and has links to windows showing more detail on each function. Figure 32b is the design of the data collection interface, Figure 32c the labelling interface, Figure 32d and Figure 32e show plans for integrating the feature library to build datasets automatically. Figure 32f shows plans for an evaluation platform: this is not part of this project but was later built by

Schmieder (2009) as an extension of DataManager. The requirements for each part of DataManager's user interface are described in more detail below.

Before any sketches can be collected, researchers must have some way defining what kind of sketches they want to collect from participants for their particular project. The basic functions of creating new projects as well as opening and saving existing projects (using xml files) are required here. A process diagram, shown in Figure 33, was drawn to design the steps to creating a new project.

The first step is to define templates that specify the types of diagrams that we are interested in collecting. It is important that participants are presented with the same information when asked to draw specific diagrams, to ensure consistency in our collection. Thus, templates for these diagrams must be defined for each diagram type that we want to collect for the particular project. Templates should contain a diagram name and instructions to participants on what to sketch. This ensures that when participants come to sketching a diagram, they are all shown the same information from these templates before they begin. For example; consider Tom, a researcher interested in developing recognisers for organisation diagrams and user interface designs. He would like to collect a large number of diagrams from both domains to use as data in his investigation. He would like each participant to draw one of each type of diagram. To do this, he defines two templates: one for organisation diagrams and the other for user interface designs. He includes instructions to participants on how they should construct each diagram.

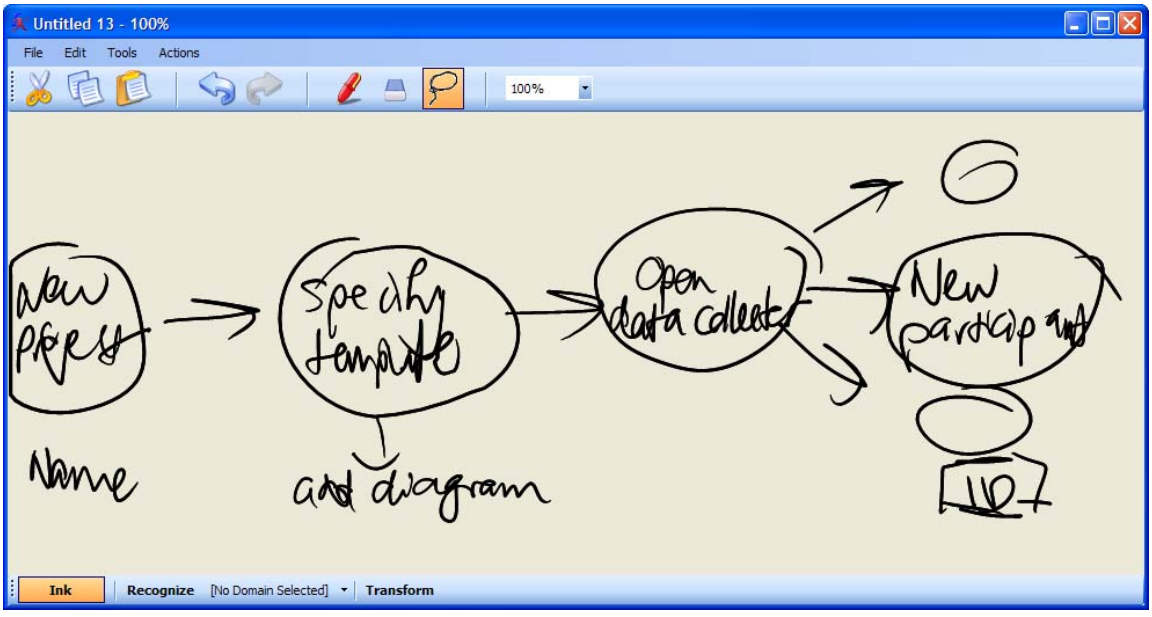


Figure 33 Process Diagram of Project Creation and Data Collection

A lo-fidelity prototype for creating a new project and collecting data is shown in Figure 34. The lines between windows in Figure 34 show the results of clicking the buttons shown on each interface design. Figure 34a is the initial dialog shown when the program is run, giving options to open an existing project or to create a new one. If an existing project is opened, the main window in Figure 34c is shown. If a new project is selected, Figure 34b shows where the templates are defined for each diagram type that will be collected for the particular project.

In addition to a diagram name and instructions on how to construct the diagram, each template should also include the specific labels related to the diagram. These are defined by the researcher, based on the type of diagram that is being collected. For example, Tom adds the labels “text”, “shape”, “rectangle” and “line” for the organisation diagram template, and “text”, “shape”, “textbox”, “button”, “combo box”, “radio button” for the user interface design template. Once Tom has collected organisation diagrams and user interface designs from participants, these labels (defined in each template) can be used to label each sketch.

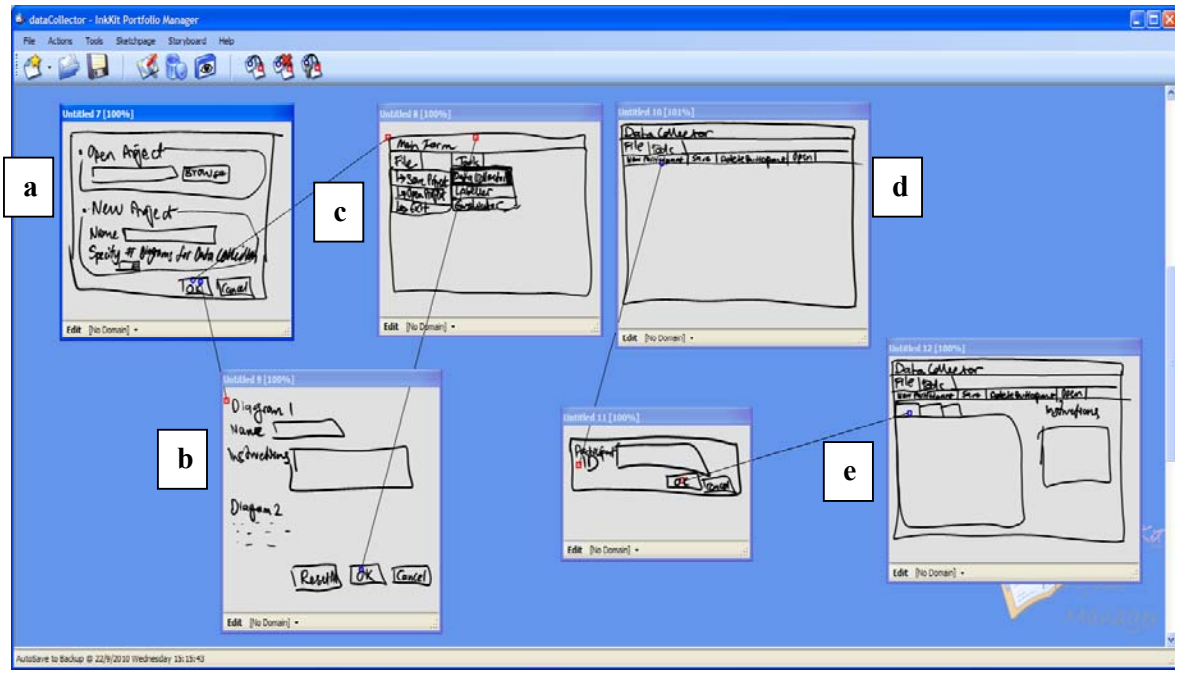


Figure 34 Lo-fidelity Prototype for Project Creation and Data Collection

Once the templates have been defined for the project, the process of data collection begins, as shown in Figure 33. DataManager must support the collection of multiple sketches from many participants. A tab view, with a drawing area for each diagram defined by the project templates and written instructions on how to construct these diagrams, is an ideal way to display what participants are required to sketch. For example, Tom’s participants are presented with two tabs: one for drawing an organisation diagram and

the other for a user interface design. In each tab there is a drawing space and textual instructions describing how to draw each diagram. These instructions are taken from the templates that Tom defined when he created the project. Designs for the data collection interface of DataManager are shown in Figure 32b and Figure 34e. Editing facilities such as select and erase must be available when drawing these diagrams. When a participant has finished, the sketches should be viewable but not editable.

Scenarios were written for the data collection process to help determine the requirements of this part of DataManager: these scenarios are shown in Figure 35. The first scenario describes the steps a participant must take to sketch diagrams. For example, in following the scenario for Tom's case, Tom (or the participant) would run the application, open the project file and the data collection interface. This step is illustrated in the lo-fidelity prototype in Figure 34a, c and d which show the open project dialog, the main interface and the data collection interface. Tom's participant would then click on the "new participant" button which opens up a new tab view showing a tab for an organisation diagram and a user interface design. These steps are illustrated by the transition from Figure 34d to e, where the new participant button is pressed and the interface updates to show the tabbed view for each diagram. The participant can then proceed to sketch each diagram and save the project.

The second scenario explains the steps required for a researcher to view participants' diagrams. The data collection interface is accessed in the same way as the previous scenario. A list box shows all the identification numbers (IDs) of all participants that have contributed sketches to the project. In step 2, the researcher can view the diagrams drawn by a particular participant by clicking on their ID in the list box; this updates the interface to show the tab view with the completed diagrams.

The researcher needs to collect different types of diagrams from participants.

1. The researcher (or participant) runs the application and opens the project file and opens the data collection interface
2. A participant clicks on the "new participant" button.
3. A new tab view opens for each diagram to be drawn.
4. The participant reads instructions on how to construct the diagram.
5. The participant draws the diagram (the participant has the ability to edit using erase and select).
6. The participant then clicks on other tabs and draws the remaining diagrams.
7. The participant presses the save button to "save" the whole project.

The researcher wants to view participants saved diagrams.

1. The researcher runs the application and opens the project file and opens the data collection interface.
2. The researcher clicks on a participant listed in the list box.
3. The corresponding participant's diagrams are shown in a tab view.

Figure 35 Data Collection Scenarios

Once sketches have been collected, each component of the sketch must be labelled. The first design of the labelling interface is shown in Figure 32c; this shows a similar tabbed view as in the data collection

interface and labels on the right hand side to apply to the diagrams. Automatic and manual labelling must be available, to facilitate both ease of use and unbiased dataset collection. We began by supporting automatic labelling of shape and text strokes using our "Divider 2007" (Patel 2007; Patel et al. 2007), which categorises ink as text or a shape. It is possible to extend this later with further recognition and categorisation algorithms. Manual labelling can be used to correct the automatic parser and add further information.

A hierarchy of labels should be pre-defined in the diagram template, as mentioned earlier. A hierarchy can be used so that enough information is available for different recognition problems. For example, one stroke in a diagram may be labelled as a circle which will automatically imply that it can also be labelled as a shape stroke for more general recognition problems. The labelling process is summarised by the scenario written during the requirements gathering process shown in Figure 36. In Tom's case, once he has collected sketches from a participant, he must label them. Continuing with the scenario, he would launch the application, open the project file and the labelling interface. This would show a tabbed view of diagrams and list of labels to apply to the participant's diagrams. To label the organisation diagram he can automatically label it first by pressing the "auto parse" button. Not all strokes in the diagram are correctly classified, so he must correct them manually. To correct text strokes, he selects the "text" label and then selects each misclassified text stroke. He does the same for strokes with other labels until all are labelled correctly.

The researcher needs to label the collected data.

1. The researcher runs the application, opens the project file and opens the labelling interface.
2. The diagrams are displayed with a tab view for each participant.
3. They use the automatic labelling for the diagrams by hitting the "auto parse" button.
4. The researcher corrects misclassified text strokes by hitting the "text" button to indicate that they are now classifying text strokes, and then selecting each misclassified text stroke either by clicking directly on the stroke or by lassoing it.
5. They do the same for misclassified shape strokes.
6. The researcher saves the labelled diagram by hitting the "save" button.

Figure 36 Labelling Data Scenario

The researcher wants to generate a dataset of feature values for the data collected.

1. The researcher runs the application, opens the project file and opens the dataset generation interface.
2. The researcher first picks which features to include in this dataset using check boxes. (This needs to be loaded dynamically from code to include new additions to the feature library.)
3. The researcher then picks which diagrams from all the data that has been collected to include in this dataset using check boxes.
4. Once the features and diagrams have been selected, push the "generate dataset" button. This outputs the dataset file.

Figure 37 Dataset Generation Scenario

The labelled sketches can be used to generate multiple datasets for data analysis. This involves calculating a number of features for each stroke in each sketch and outputting a dataset file. The interface should make it easy to select which participants, diagrams and features should be included in

the dataset. The dataset can then be imported into data mining tools such as R (R Development Core Team 2006) and Weka (Witten et al. 2005) to be analysed for the development of new recognisers. The scenario in Figure 37 summarises this process, which was written while gathering requirements for the dataset generator. For example, if Tom wants to generate a dataset for organisation diagrams, he can follow the steps in the scenario. After opening the project and the data generation interface, he chooses the features he wants to include. He then selects all the organisation diagrams drawn by all participants and finally clicks the "generate dataset" button. The dataset of organisation diagrams is generated and output into a file.

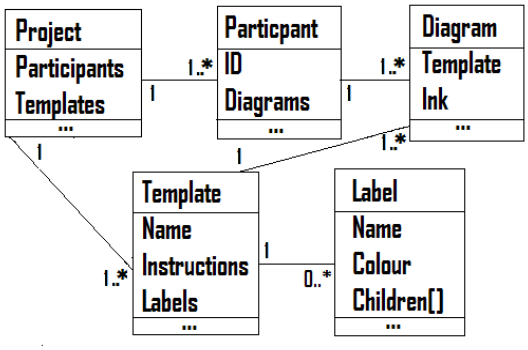


Figure 38 Target UML Class Diagram for the Main Data Structures in DataManager.

Figure 38 shows the target UML Class diagram for the data structures required in DataManager. It contains a Project class which can contain many participants. A Project is defined by one or many Templates where each Template describes the type of diagram to be collected, including the pre-defined Labels. Participants can draw many Diagrams. Each Diagram is based on a pre-defined Template.

5.2 Usage Example

The following gives an example of how a researcher can use the final implementation of DataManager. The example is based on a scenario of collecting sketches representing organisation diagrams. The implementation details follow in Section 5.3.

When the application starts, a dialog is shown which gives the researcher the option of either creating a new project or opening an existing project, as shown in Figure 39. If they choose to begin a new project, they first specify a project name and then construct templates on which the project will be based.

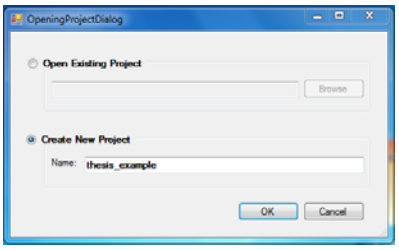


Figure 39 Open Project Dialog

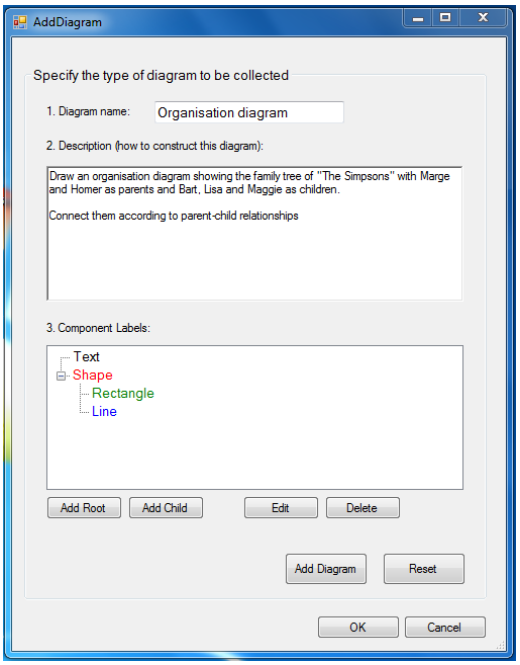


Figure 40 Add Template Form

A template provides information on a diagram type that the researcher wishes to collect. It consists of a name and instructions on how to draw a diagram which will be shown to participants. It also has a set of labels to describe the components of that diagram to be used in the labelling tool after a sketch has been collected. A dialog box (Figure 40) is displayed which asks the researcher to define a template by specifying this information. The example in Figure 40 shows a template for collecting organisation diagrams. The researcher has entered a diagram name, instructions for participants to follow when drawing an organisation diagram and labels. Once these templates have been defined for each diagram type that the researcher wants to collect, the data collection process can begin.

Templates can be added and edited later by selecting properties from the menu bar; the properties window is shown in Figure 41. Participants can also be deleted from this window.

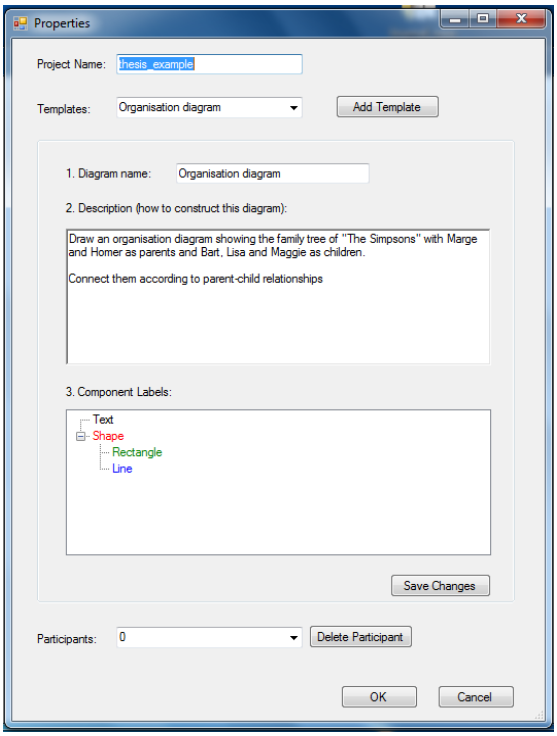


Figure 41 Properties Window

5.2.1 Data Collection

To begin collecting data from participants, the Data Collector must be selected from the tools menu. This will launch a form (example shown in Figure 42) which shows a list box on the left listing the ID numbers of the participants who have already drawn sketches for that particular project. When each ID number is selected, the diagrams drawn by the corresponding participant are shown in the drawing area. This part of the interface allows researchers to navigate large datasets with ease.

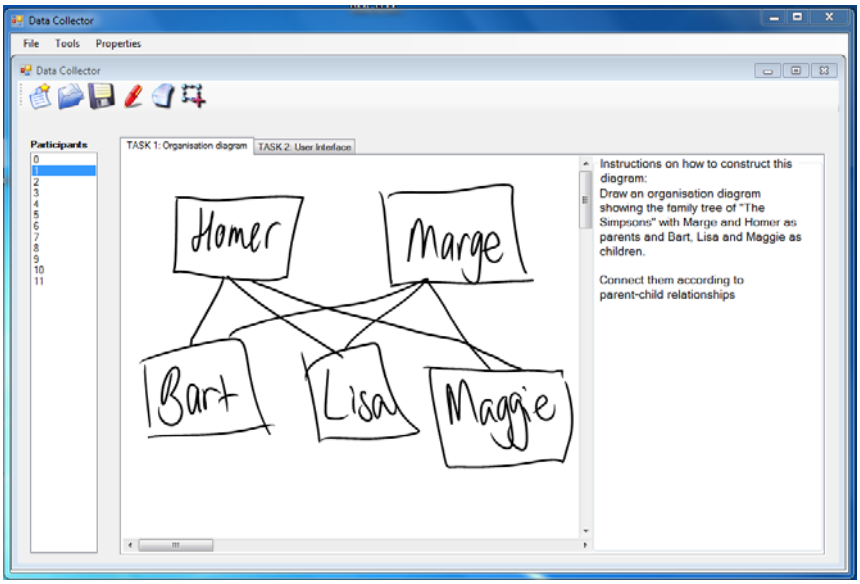


Figure 42 Data Collector Form

In the middle of the screen are the tabs for each diagram. Each tab has a drawing area and instructions on how to construct each diagram (as specified by the researcher when creating the diagram templates shown in Figure 40). All data collected is saved to XML files. This includes project information such as the diagram templates, all the raw stroke data for each participant and the corresponding labels applied to these strokes, as discussed in the next section.

The simple interface ensures that the participant finds the tool easy to use. They simply read the instructions and draw the diagram defined by the instructions in the drawing area. If there is more than one diagram required, each will have a separate tab. When participants are ready to complete a new diagram, they can switch tabs to complete the remaining diagrams in the same manner.

5.2.2 Labelling Data

Once a diagram has been drawn, the strokes can be labelled. Using the tools menu, the researcher can select the Labeller which will take them to the screen shown in Figure 43.

The user interface for the Labeller has the same list box showing the participant IDs and tabs for each diagram as the Data Collector form. The drawing area on each tab is un-editable except for changing the colour of the stroke. Pressing the "auto parse" button (top right button) automatically parses the current diagram using our old divider, Divider 2007 (Patel 2007; Patel et al. 2007) into shape and text strokes. It colours the strokes according to the colour map shown in the tree view box e.g. text strokes are black and shape strokes are red.

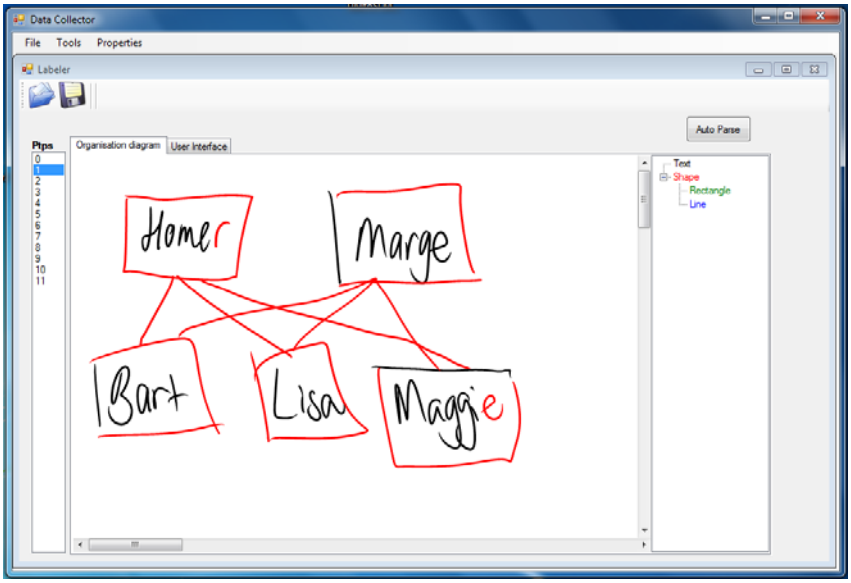


Figure 43 Labeller Form Showing a Diagram Labelled using the Automatic Parser

The user can also manually label strokes by selecting the correct label from the tree view component and selecting the stroke/strokes that require this label. The labels are those specified when defining the template for that diagram type as seen in Figure 40. The stroke is then coloured to match the deepest label in the tree as shown in Figure 44. We have chosen this hierarchical labelling structure to allow more than one label to be applied to a stroke without manually specifying each one. Designing a labelling structure for efficient multi-label and multi-stroke labelling is difficult. The hierarchical structure that has been implemented meets the needs of developing text-shape dividers and provides flexibility for future use.

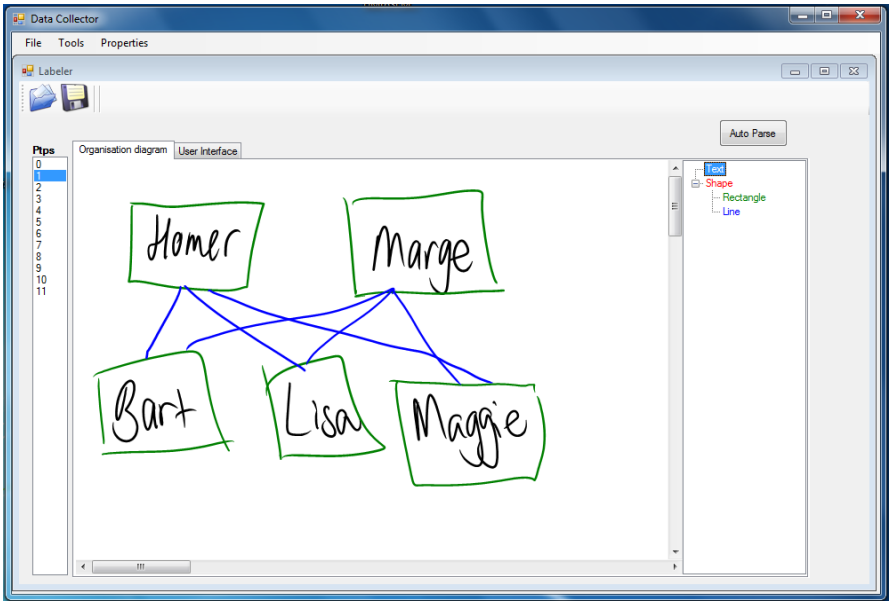


Figure 44 Labeller Form showing a Diagram Labelled Manually

5.2.3 Dataset Generation

The final step to the data collection process is to generate a dataset. To do this, the researcher selects "Dataset Generator" from the tools menu. The screen shown in Figure 45 will appear.

There are two steps to generating a dataset. Firstly, the researcher must choose which digital ink features they want to measure and secondly choose which diagrams they are interested in measuring.

A list of possible features is displayed in a list box (top left in Figure 45) sorted according to the taxonomy described in Chapter 4. This list is dynamically generated to ensure that the feature set is easily extensible. There are check boxes (top right in Figure 45) to enable the user to select or deselect all features or groups of features from each category with ease. Only those features selected are calculated in the dataset. The full feature set is described in Chapter 4.

All the diagrams that are part of the current project are displayed in the lower list box. It has a tree structure showing which diagrams each participant has drawn. A quick select list (lower right in Figure 45) is available to enable the user to select or deselect all the diagrams or to easily choose only certain diagram types. Only the diagrams that have been selected are included in the dataset.

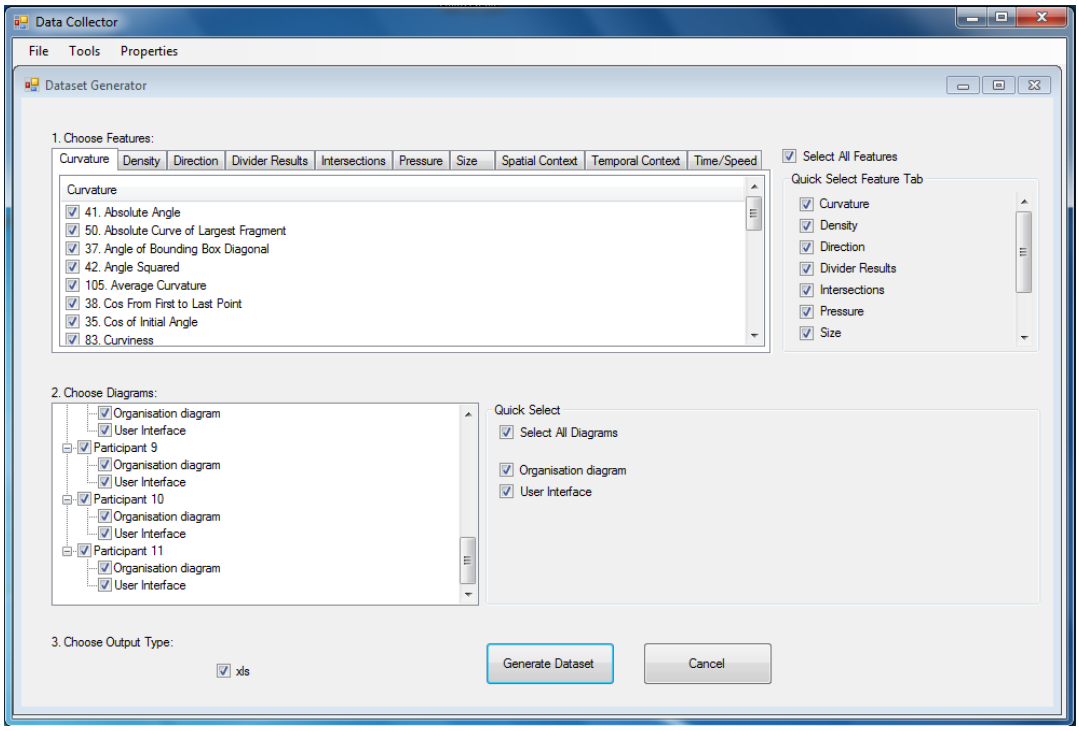
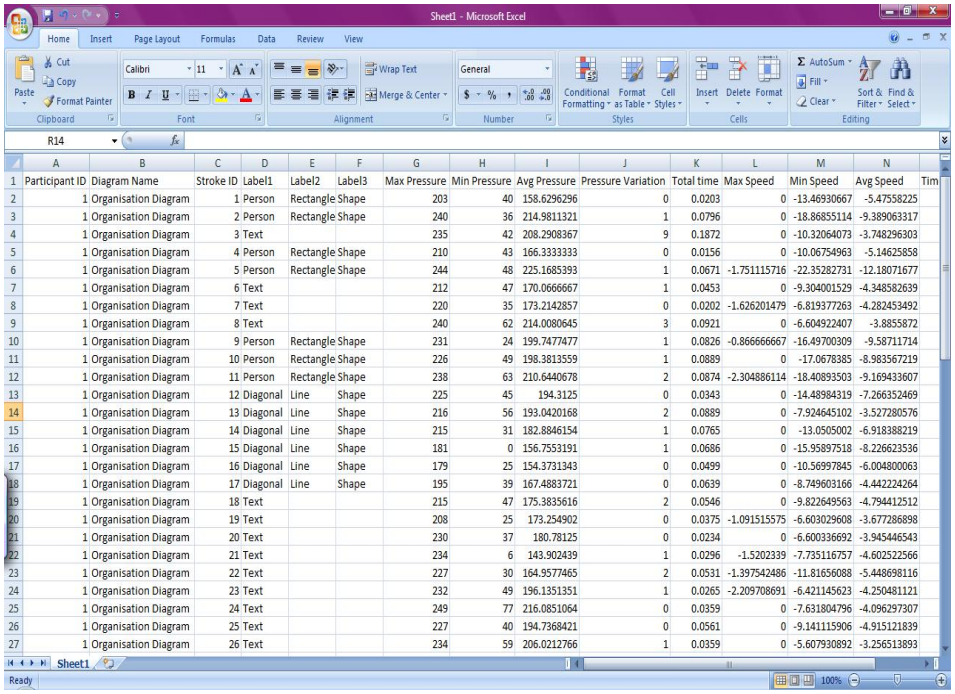


Figure 45 Dataset Generator Form



Participant ID	Diagram Name	Stroke ID	Label1	Label2	Label3	Max Pressure	Min Pressure	Avg Pressure	Pressure Variation	Total time	Max Speed	Min Speed	Avg Speed
1	Organisation Diagram	1	Person	Rectangle	Shape	203	40	158.6296296	0	0.0203	0	-13.46930667	-5.47558225
1	Organisation Diagram	2	Person	Rectangle	Shape	240	36	214.9811321	1	0.0796	0	-18.86855114	-9.389063317
1	Organisation Diagram	3	Text			235	42	208.2908367	9	0.1872	0	-10.32064073	-3.748296303
1	Organisation Diagram	4	Person	Rectangle	Shape	210	43	166.3333333	0	0.0156	0	-10.06754963	-5.14625858
1	Organisation Diagram	5	Person	Rectangle	Shape	244	48	225.1685393	1	0.0671	-1.751115716	-22.35282731	-12.18071677
1	Organisation Diagram	6	Text			212	47	170.0666667	1	0.0453	0	-9.304001529	-4.348582639
1	Organisation Diagram	7	Text			220	35	173.2142857	0	0.0202	-1.626201479	-6.819377263	-4.282453492
1	Organisation Diagram	8	Text			240	62	214.0080645	3	0.0921	0	-6.604922407	-3.8855872
1	Organisation Diagram	9	Person	Rectangle	Shape	231	24	199.7477477	1	0.0826	-0.866666667	-16.49700309	-9.58711714
1	Organisation Diagram	10	Person	Rectangle	Shape	226	49	198.3813559	1	0.0889	0	-17.0678385	-8.983567219
1	Organisation Diagram	11	Person	Rectangle	Shape	238	63	210.6440678	2	0.0874	-2.304886114	-18.40893503	-9.169433607
1	Organisation Diagram	12	Diagonal	Line	Shape	225	45	194.3125	0	0.0343	0	-14.48984319	-7.266352469
1	Organisation Diagram	13	Diagonal	Line	Shape	216	56	193.0420168	2	0.0889	0	-7.924645102	-3.527280576
1	Organisation Diagram	14	Diagonal	Line	Shape	215	31	182.8846154	1	0.0765	0	-13.0505002	-6.918388219
1	Organisation Diagram	15	Diagonal	Line	Shape	181	0	156.7553191	1	0.0686	0	-15.95897518	-8.226623536
1	Organisation Diagram	16	Diagonal	Line	Shape	179	25	154.3731343	0	0.0499	0	-10.56997845	-6.004800063
1	Organisation Diagram	17	Diagonal	Line	Shape	195	39	167.4883721	0	0.0639	0	-8.749603166	-4.442224264
1	Organisation Diagram	18	Text			215	47	175.3835616	2	0.0546	0	-9.822649563	-4.794412512
1	Organisation Diagram	19	Text			208	25	173.254902	0	0.0375	-1.091515575	-6.603029608	-3.677286898
1	Organisation Diagram	20	Text			230	37	180.78125	0	0.0234	0	-6.600336692	-3.945446543
1	Organisation Diagram	21	Text			234	6	143.902439	1	0.0296	-1.5202339	-7.735116757	-4.602522566
1	Organisation Diagram	22	Text			227	30	164.9577465	2	0.0531	-1.397542486	-11.81656088	-5.448698116
1	Organisation Diagram	23	Text			232	49	196.1351351	1	0.0265	-2.209708691	-6.421145623	-4.250481121
1	Organisation Diagram	24	Text			249	77	216.0851064	0	0.0359	0	-7.631804796	-4.096297307
1	Organisation Diagram	25	Text			227	40	194.7368421	0	0.0561	0	-9.141115906	-4.915121839
1	Organisation Diagram	26	Text			234	59	206.0212766	1	0.0359	0	-5.607930892	-3.256513893

Figure 46 Example Dataset

Once all the desired selections have been made, the researcher clicks on the “generate dataset” button. Each selected feature is calculated for each stroke in the selected diagrams with the results written to an Excel spreadsheet. Figure 46 shows an example dataset for the organisation diagram in Figure 44. There are extra pieces of information added to each stroke, including the participant ID, the diagram name, the stroke ID and the labels applied to that stroke.

The resulting dataset can be analysed by data mining tools such as Weka (Witten et al. 2005) to determine the most significant features and algorithms for any given sketch recognition problem.

5.3 Implementation

DataManager has been implemented using C# .NET Framework with extensive use of the Microsoft.Ink library to handle the digital ink in the application. In particular, the Microsoft.Ink.Stroke class is the key data structure used to represent the ink.

Most of the interface is generated dynamically at runtime rather than at design time. This is because the interface is highly dependent on the actual project that has been opened. For example, the number of tabs shown in the data collection and labelling interfaces is dependent on how many templates have been specified for the particular project. Another example is the features available in the dataset generator; these are added at runtime so that additions to the feature library do not require changes to the interface at design time.

The overall architecture of DataManager is shown in Figure 47. The three core interfaces, Data Collector, Labeller and Dataset Generator, run from the main DataManager form. Information held in the Data Structures (described in Figure 48) is used to populate each of the core interfaces. For example, the Data Structures hold information about the labels applied to strokes; this information can be viewed with the Labeller interface. The Data Structures are also updated with information based on users’ interaction with the Data Collector and Labeller. For example, when new labels are applied to strokes, this information is saved using the Data Structures. The Dataset Generator relies on the Feature Library to generate datasets.

A UML class diagram showing the final architecture of the data structures within the tool is shown in Figure 48. The main addition to the UML class diagram, from the target diagram in Figure 38, is the LabeledStroke class. This class represents a stroke in a diagram that has been labelled. It is different from the Label class which holds the definition of a label originating from the diagram template. The

LabeledStroke holds the relationship of a stroke to its assigned Label. In addition, as a Diagram is composed of strokes it can have many LabeledStrokes. The data structures represented in Figure 48 are serialised and de-serialised into XML to save and load the project.

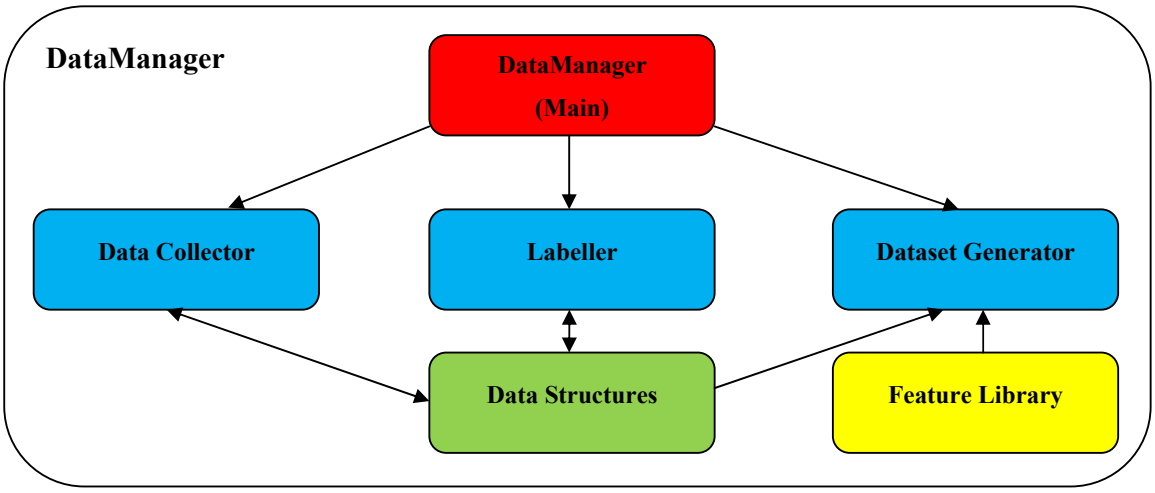


Figure 47 Architecture Diagram of DataManager

DataManager has been designed so that each function is independent; their only link is based on the project information held by the main Data Structures in Figure 48. Therefore, adding functions is a simple task provided that the correct information is updated in the Data Structures. For example if the Labeller was replaced, the Labels from each Template must be passed to the new Labeller to display as possible labels for a particular diagram. In addition, once a stroke in a Diagram is labelled, the corresponding LabeledStroke must be updated. Extensions to DataManager have been made by others, including an evaluation platform for recognition algorithms (Schmieder 2009) and an interface to Weka (Witten et al. 2005) for automatically generating recognisers (Chang 2010).

The main implementation challenges were in designing the application. The design of the data structures was done carefully as they form the basis of DataManager. The use of UML class diagrams here was a very valuable tool. The interface design was also important as it had to be easy for participants to use and to provide flexibility for researchers to define their own projects with minimal limitations. Constructing lo-fidelity prototypes and scenarios were useful tools for this task.

The design of the labelling structure was a particularly difficult task. It was important to provide researchers with the ability to define their own labels, as projects can be very different to one another. We also wanted to allow multiple labels to be applied to the same stroke while still ensuring that the labelling process is as efficient as possible. The hierarchical tree structure for labelling was the best solution we found.

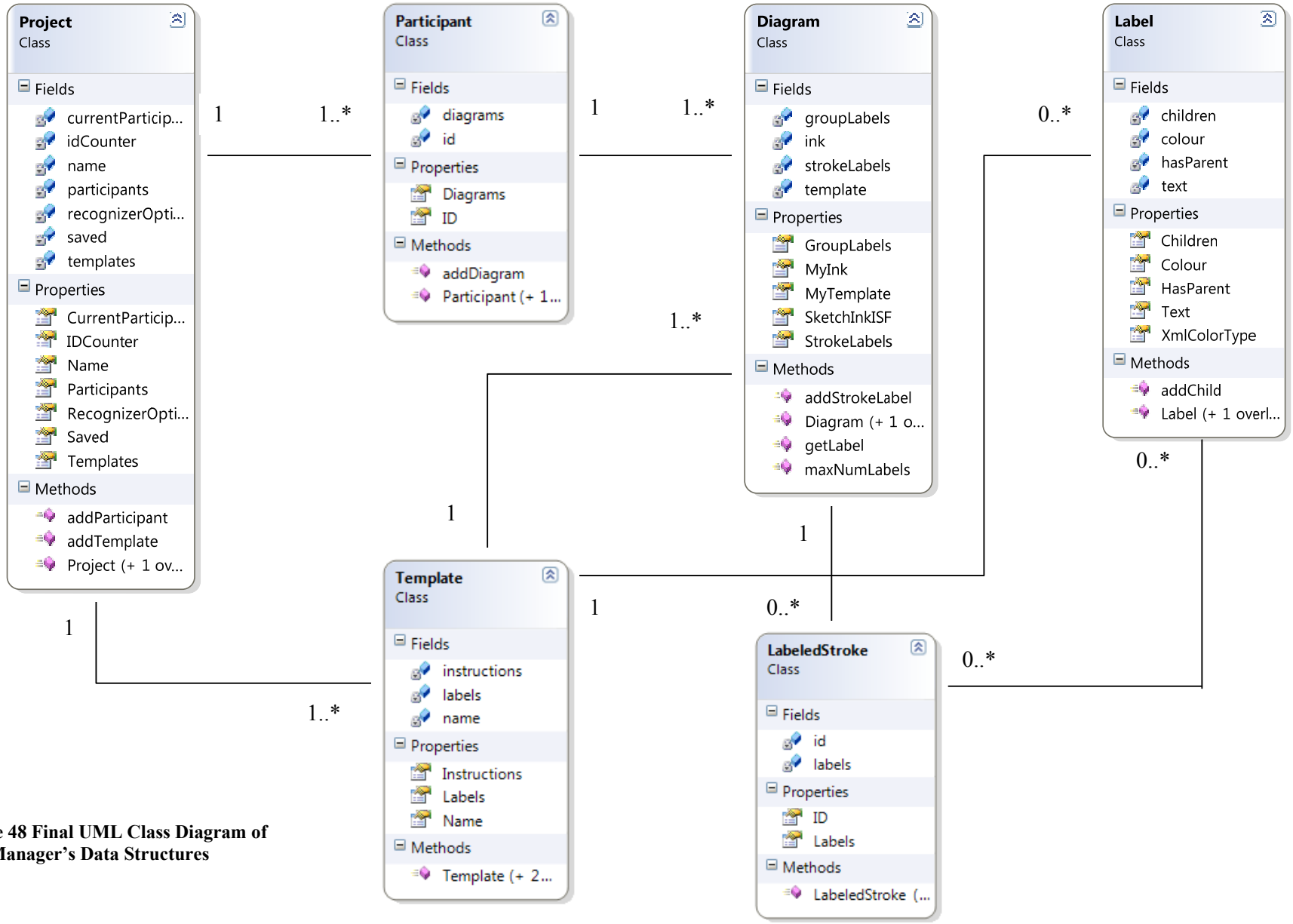


Figure 48 Final UML Class Diagram of DataManager's Data Structures

Some unexpected implementation decisions had to be made. For example, what would happen to data already collected and labelled if researchers later edited information in Templates? We had to decide if the previous template information, diagram name, instructions or labels should still apply to the old data or if everything should be changed in the whole project. We decided that everything in the project should be updated. The diagram name and instructions are not so important once that data has been collected, although it does mean that the researcher will have no record of what instructions were actually given when the diagram was constructed. The labels on the other hand should be updated if modifications are made. However, this means that diagrams already labelled may have to be re-labelled if some labels no longer exist or have been edited.

Once a good design was in place, the implementation was completed with relative ease.

5.4 Usability Study

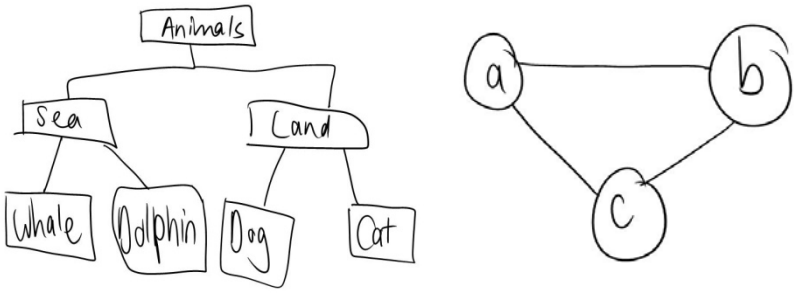
A usability study was undertaken to test how intuitive DataManager is to use; in particular the data collection and labelling interfaces. For the data collection interface, we wanted to learn how easy it is for participants to sketch diagrams using the provided instructions. For the labeller, we were interested in how efficiently we can label collected diagrams with the existing interface.

5.4.1 Data Collection

Six students from a Computer Science and Software Engineering background participated in the study. Half the participants use pen input on a computer frequently and the others had used pen input only occasionally or once before. As we were testing the usability of DataManager from a participant's point of view, rather than that of a researcher, these students were suitable for the study, although students with this background are expected to be more competent with technology. Ethics approval was obtained from the University of Auckland Human Participants Ethics Committee for this and later studies for data collection; details can be found in Appendix E.

The participants were asked to draw two types of diagrams: an organisation diagram and a graph. Examples are shown in Figure 49. The participants were given very specific instructions on how to construct these diagrams. However, at this stage we were not interested in evaluating the way problems are presented to participants to sketch, but were only evaluating the usability of the DataManager interface.

We observed the participants as they completed each task and then asked them to complete a short questionnaire, shown in Appendix F. The questionnaire focused on learning how easy it is for participants to complete the tasks with our software on a Tablet PC. All participants strongly agreed that, given the environment, creating the diagrams was easy. They also agreed that the interaction tools (hardware and software) helped them to complete each task. All participants agreed that they understood the tasks they were to perform. Although we were not evaluating the way that the tasks were presented, this feedback gives us a positive indication that the style used to display instructions to the user on how to complete each task is effective.



a. Organisation Diagram *b.* Graph

Figure 49 Diagrams Collected for the Usability Study

Five of the six participants were neutral when asked if editing and checking the diagrams was easy. This is because most completed the task without the need to edit the diagram as the tasks were easy to understand and they had been presented with clear instructions. The sixth participant strongly agreed that editing and checking the diagrams was easy.

Three participants, after completing the first task, almost used the participant list box by accident to navigate to the next task. However, before clicking in the wrong place, they quickly realised that they needed to use the tabs to switch tasks. We renamed the tabs "Task n" in a larger font before the diagram name to make this selection more obvious.

One participant was also unsure where the instructions were for the second task as they did not realise that the text area would change to display the instructions corresponding to the selected tab. We altered the interface to include the text area within the tab to make it clear which instructions belonged to which task. An additional observation was that the drawing area was too small. Of course, this is restricted by the tablet screen area, but we were able to add scroll bars in case users required more space.

5.4.2 Labelling

We were interested in evaluating how efficient DataManager's Labeller is to use. After collecting all the diagrams from the participants, each sketch was labelled while measuring how long this process took. To label all 12 diagrams (two diagrams per participant) it took approximately seven minutes. An extra minute was used to double check all the diagrams and another 2.5 minutes for the dataset to be generated using the 45 features from our previous work (Patel et al. 2007). This is a total of approximately 10.5 minutes to label all diagrams and generate a dataset of 476 strokes. In comparison, manually labelling the data for our previous work with 26 participants and 1519 strokes took more than three days.

When labelling the diagrams, we found that the automatic parser using our text-shape divider (Patel 2007; Patel et al. 2007) to give preliminary labels to the diagram was especially helpful, given the amount of text that was in the diagrams.

One possible improvement would be to allow all the diagrams of the same type to be labelled together. For example, label all the organisation diagrams first, and then label all the graph diagrams, rather than labelling all the diagrams for each participant. This may make the labelling process more efficient as it may minimise the cognitive load required when switching tasks and allows for familiarity with labelling one type of diagram. Modifying the participant list box, shown in Figure 43 (left hand side), to display by diagram type as an alternative to displaying by participant would allow the user to navigate through each sketch as required. At times there are also strokes that do not fall into the categories of expected labels. In such cases, we recommend always adding an extra label for miscellaneous strokes.

The evaluation results indicate that DataManager provides a good environment for capturing and labelling ink data for further analysis. The usability study showed that participants found the tool intuitive and easy to use and the labelling interface was shown to be an efficient way of performing this task.

DataManager has since been used for data collection in several other projects (Schmieder 2009; Schmieder et al. 2009; Chang 2010; Chang et al. 2010; Delaney et al. 2010). Data collected using DataManager has also been used in a recent study of data collection (Field et al. 2009). DataManager's functionality has also been extended by others to include an evaluation platform for recognition algorithms (Schmieder 2009), a group labelling function (Blagojevic et al. 2009; Schmieder 2009) and an interface to Weka (Witten et al. 2005) for generating basic shape recognisers (Chang 2010). These additions rely on the data collection, labelling, and dataset generation functions developed as part of this thesis.

5.5 Data Collection

Several datasets are required for developing text-shape dividers; in particular, for training and testing our text-shape divider algorithms. DataManager was used to collect and label this data and finally to generate datasets for analysis. The requirements of the data collected are summarised below, followed by a description of the datasets collected. Additional datasets were collected for the evaluation of our dividers and they are described in Section 7.2.

5.5.1 Requirements

We have identified three basic requirements for data we collect. First and foremost it must be of high quality. High quality data should be free from bias as much as possible, as discussed in Section 5.1. To ensure that our data fits this requirement, our participants were asked to draw complete diagrams, rather than isolated components. They were also given written instructions, rather than pre-drawn examples to copy. Secondly, we need a large quantity of data to ensure our analysis is fair. Finally, we require a good variety of diagrams to be included in the dataset. Variety in the data will help us to develop a text-shape divider that can distinguish components in a wide range of situations. We have chosen three types of diagrams to collect which display various shapes and components, relationships between components and differing diagrammatic structures.

5.5.2 Data

There are two independent groups of data described here: one is the training dataset used for training the algorithms; the other is the verification dataset which is used to obtain preliminary test results to inform further training. Summary statistics for the data are shown in Table 18. A more detailed description of each diagram type follows.

	# Participants	# Text Strokes	# Shape Strokes	Total # Strokes	% Text:% Shape Strokes
Training					
User Interface	20	4354	671	5025	87:13
Directed graph	20	164	354	518	32:68
Organisation chart	20	1098	607	1705	64:36
Total Training Set		5616	1632	7248	77:23
Verification					
ER diagrams	33	2143	1050	3193	67:33
Process diagrams	33	2674	1195	3869	69:31
Total Verification Set		4817	2245	7062	68:32

Table 18 Summary Statistics for each Dataset

Training Dataset

For the training set, diagrams were collected from 20 participants using DataManager. Each participant drew three diagrams; a directed graph, organisation chart and a user interface design. Examples of these diagrams are shown in Figure 50. There are a total of 7248 strokes in the training set, with 5616 text strokes and 1632 shape strokes; further statistics for each diagram type are shown in Table 18. The instructions given to participants when sketching each diagram are shown in Table 19. A small pilot test of the instructions was conducted to ensure that participants could easily complete the tasks. The pilot test showed that the tasks could be easily completed by participants when following the instructions given. Participants came from a variety of backgrounds and ages ranging from 18 years and upwards.

User interface design diagrams feature in the training dataset for several reasons. This domain of diagrams typically contains a range of shapes including squares, rectangles, circles and triangles as well as a large amount of text. The size of shapes in a user interface can be similar to text such as in the case of radio buttons and check boxes. There are interesting relationships between shapes, such as in combo boxes where a triangle is contained by a rectangle, and between shapes and text, such as text labels of radio buttons and check boxes that may be difficult to distinguish in terms of a text-shape divider. Additionally, there can be lines of text that differ in length.

Directed graphs have a different set of characteristics to user interface design diagrams. These diagrams are composed of nodes, arrows and connectors. They have proportionately less text than shapes. The text is always contained within a node and in this case is a single character. We specifically wanted to include arrows in our training set as these are a known problem for sketch recognition (Kara et al. 2004; Freeman et al. 2007). The connectors are also interesting as they differ in length and curvature. The overall layout of directed graphs is less structured than the other diagrams in the training dataset.

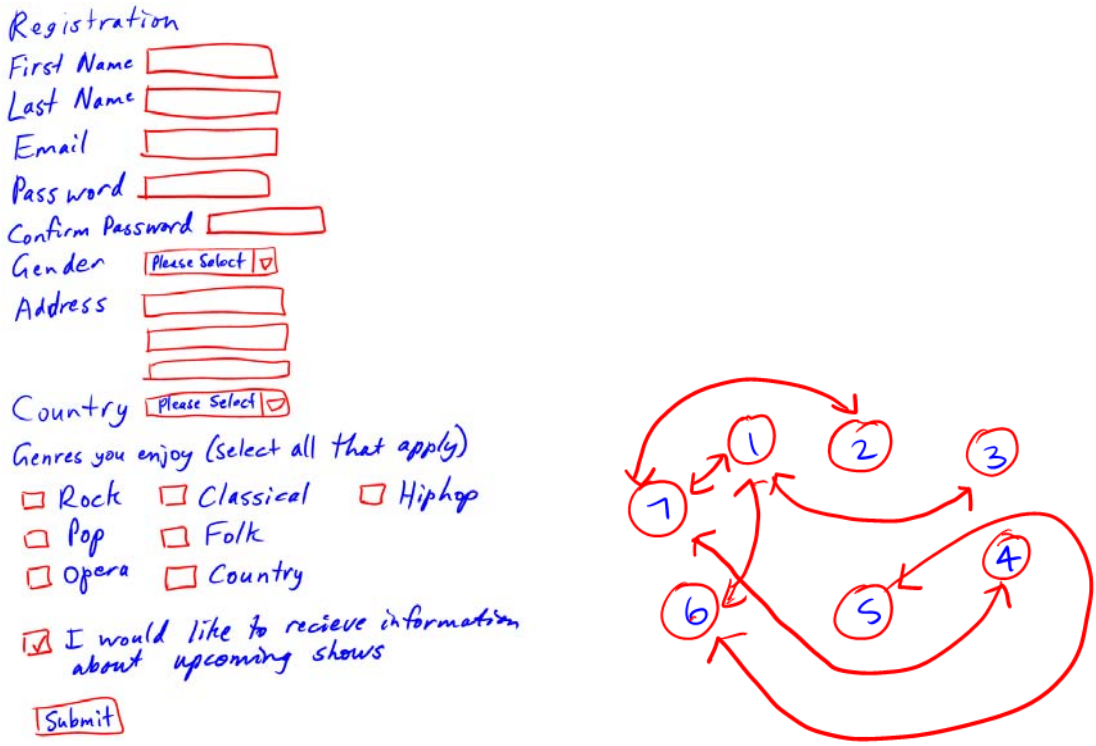
Organisation charts are a hierarchically structured node and connector diagram. They are typically composed of rectangles and straight line connectors. Text labels are contained within each rectangle and are typically more than a single character, which is in contrast to the directed graph. This is reflected in the ratio of text to shape strokes in Table 18 for these diagrams.

All of these characteristics are used to train text-shape dividers to handle a variety of elements in a given diagram.

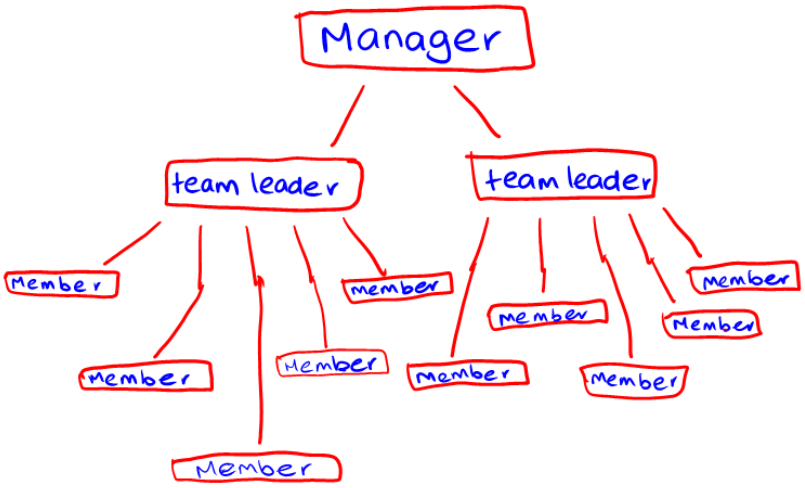
Diagrams	Instructions
User interface	Design a web form for user registration to a website for buying concert tickets online First Name Last Name Address Country Email Gender Password Confirm Password Ask the user to pick genres of music they are interested in from the list below Rock Pop Classical Folk Country Opera Hiphop And ask them to tick a box if they would like to receive information about upcoming shows.
Directed graph	Draw a directed graph with 7 nodes labeled 1-7. Add the following arrows to connect the nodes 1 to 3 2 to 7 4 to 7 5 to 6 6 to 1 7 to 1
Organisation chart	Draw an organisation diagram showing the hierarchical structure a company with a manager, 2 team leaders and 10 team members (5 members under each team leader)

Table 19 Instructions to Participants for Drawing Diagrams for Training Data

Using this collection of diagrams, we generated a dataset of feature vectors for each stroke using DataManager. DataManager's dataset generator function is able to take the diagrams collected and calculate feature vectors based on the implementation of our feature library. (See Chapter 4 for a description of the feature library.) The dataset was cleaned before analysis. This involved removing elements such as "+/-Infinity", "NaN", "#Name" which are generated for some feature calculations when written to an Excel file format.



a) User Interface b) Directed Graph



c) Organisation Chart

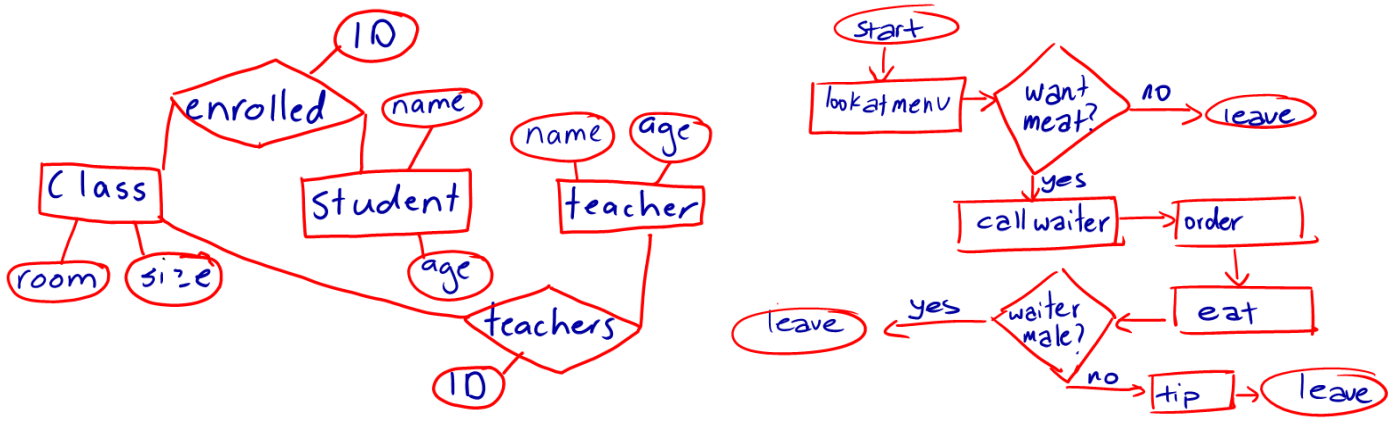
Figure 50 Example Sketched Diagrams for Training Set

Verification Dataset

This dataset was collected by Schmieder (2009) using DataManager. It is used for preliminary testing of text-shape dividers to further improve their development. This test set is composed of entity relationship (ER) and process diagrams (shown in Figure 51) collected from 33 participants who drew one diagram from each domain. The participants were asked to construct the diagrams from textual descriptions composed by Schmieder (2009). There are a total of 7062 strokes in this test set which is similar in size to

the training set. There are 4817 text strokes and 2245 shape strokes; further statistics can be found in Table 18.

These diagrams are composed of a wide range of shapes including rectangles, diamonds, lines, circles, ellipses, arrows and triangles. In addition, there is text contained within shapes and outside shapes: for example, labelled connectors in the process diagrams. Although this dataset does not display the full range of characteristics of the training dataset, it can be used as a preliminary test bed for text-shape dividers that is completely independent from data used for training.



a) ER Diagram b) Process Diagram

Figure 51 ER and Process Diagram Test Set Examples

The development of sketch recognition algorithms requires large amounts of unbiased data. We have developed a tool, DataManager, for collecting sketches, labelling those sketches and automatically generating datasets for use in further analysis. The main requirement of this tool is that it minimises the time and effort required by researchers for ink data collection and management.

Our usability study shows that DataManager provides a good environment for collecting and labelling data and that labelling and dataset generation time is significantly reduced when using this tool over manual methods used previously. Two datasets have been generated using DataManager: one for training and another for preliminary testing of classifiers. The systematic analysis of data mining techniques described in the next chapter uses these datasets to develop more accurate recognisers.

Chapter 6

Data Analysis

This chapter describes the steps that have been used to build new text-shape dividers as illustrated in Figure 52. Firstly, as an input to the data mining algorithms used to build classifiers, a training dataset was generated. It was generated by calculating feature vectors, using the feature library from Chapter 4, on each stroke collected in the training set of diagrams described in Section 5.5.2. The next step involved running a preliminary investigation into appropriate data mining algorithms. The results of this preliminary analysis were used to narrow the study to seven algorithms. Subsequently, these seven algorithms were tuned to their optimal parameters for the training dataset. Following this, the use of feature selection and ensembles were investigated, but proved not to improve results significantly. Finally, a second round of analyses based on the preliminary evaluations were performed. The culmination of this process is a group of general text-shape dividers.

Weka (developer version 3.7) (Witten et al. 2005), an open source data mining tool described in Section 2.4, was used to perform the data analysis and to build, tune and test classifier models for our dividers. Weka has a large number of machine learning algorithms, ranging from Naïve Bayes, Decision Trees and Rules to Support Vector Machines and Neural Networks. These algorithms can be used to perform data mining on supplied training data and consequently build classifiers based on this analysis. Due to Weka's enormous number of variables in terms of training parameters and algorithms, we sought advice from Frank (2009 - 2010) in order to optimise our search for effective algorithms. In particular, he provided advice on algorithm selection, tuning of parameters, methods of feature selection and the use of ensembles.

This chapter describes each of the steps shown in Figure 52 in more detail and outlines the computational requirements of such an analysis.

Figure 52 Process Diagram of Analysis

6.1 Preliminary Analysis of Classifiers

The goal of the preliminary phase was to explore a large range of data mining algorithms and narrow them down based on their performance and suitability to the divider problem. From the many algorithms within Weka, 39 were considered as possible candidates. Selection of these candidates was based on their ability to classify data as nominal classes, in this case text and shape, rather than numeric output. The analysis began with a preliminary investigation of all these algorithms. This involved building classifier models for each algorithm using the training data. The training data has a total of 7248 strokes; 5616 are text strokes and 1632 are shapes. Further details on the training data used for the analysis can be found in Section 5.5.2.

The results of the preliminary analysis showed that several classifiers clearly performed well on the training dataset. Others needed tuning of their specific parameters to optimise their results. A summary of these results is presented in Appendix A. The top three classifiers, based on accuracy, were RandomForest (Breiman 2001), Bagging (Breiman 1996) and Multilayer Perceptron (Minsky et al. 1969; Rumelhart et al. 1986). Five of the top ten classifiers, based on accuracy, use Logistic Regression in some form. The lowest performing classifiers are variations of decision trees, ensembles, rules and boosting procedures. The list of algorithms was narrowed down to seven that were likely to gain the best classification accuracy for a divider.

The chosen classifiers are listed below and are described in more detail in the next section.

1. Bagging (Breiman 1996)
2. RandomForest (Breiman 2001)
3. LogitBoost (Friedman et al. 2000)
4. LADTree (Holmes et al. 2002)
5. LMT (Landwehr et al. 2005; Sumner et al. 2005)
6. Multilayer Perceptron (Minsky et al. 1969; Rumelhart et al. 1986)
7. SMO (Hastie et al. 1998; Platt 1999; Keerthi et al. 2001)

6.2 Classifier Tuning

Using the same training dataset of feature vectors, as used for the initial investigation of algorithms, text-shape dividers were built by training each of the chosen seven classifiers. While Weka provides sensible default parameters for most algorithms, some classifiers required tuning to optimise their results.

Ten-fold cross validation is used for training in all our experiments. This is a standard technique used in data mining to make the most of the data that is available (Witten et al. 2005). It involves splitting the dataset into ten parts (or folds) and running the training and testing procedure ten times, each time using a different part of the data to test the classifier and the remaining nine parts to train. An average classification rate is obtained using the results of the ten tests.

In this chapter, algorithms are compared using paired t-tests. Paired t-tests show if the mean of one method is significantly different to the mean of another. The same folds for cross validation of the dataset are used in all experiments and thus the results for each fold can be paired up between methods to perform the paired t-tests. When using cross validation, samples are not independent. This can cause significant differences to be found that don't really exist (Witten et al. 2005). Weka's Experimenter interface includes a corrected paired t-test that solves this problem by modifying the t-statistic used (Witten et al. 2005). The correct paired t-test is used in this chapter. For all paired t-tests, the degrees of freedom (df) is 9, the significance level (α) is 0.05 and the standard deviation (sd) is shown in brackets next to the average percentage correct for a particular experiment.

6.2.1 Bagging

Bagging (Breiman 1996) is short for "bootstrap aggregating". It involves taking k bootstrap samples of the dataset and using these samples to train k classifiers. To take a bootstrap sample, the dataset is randomly sampled, with replacement, to generate a new dataset that is the same size as the original (Efron et al. 1993). This new dataset is used to train the base learner. This process is repeated k times. The final classification results are obtained by voting with equal weighting given to each classifier's prediction. The trained classifiers are used to predict the class of an instance and the most frequent prediction is considered to be the final classification.

The base learner for Bagging used in our experiments is the REPTree, a fast decision tree (Witten et al. 2005). The REPTree is the default setting for Bagging in Weka. Standard decision trees are made up of

nodes and leaves; an example decision tree is shown in Figure 53. Nodes are responsible for testing features against pre-determined values, other features or use functions involving features. For nominal features, the number of child nodes is usually equal to the number of possible values. For example, for the nominal feature curviness, there are three possible values: low, intermediate and high. Therefore, using this feature in a decision tree will look something like that shown in Figure 53. For numeric features, the number of branches can vary between two or three children, determining if the feature value is less than, greater than or within a specific range of values as shown in Figure 53. The leaves deliver the classification of an instance. To classify an instance, it is sent down the tree, where its features are tested at each node in its path until a leaf node is reached and it is classified (Witten et al. 2005). For the decision tree shown in Figure 53, if a new instance has a curviness of "low" and a width of 2.5 then it is classified as text.

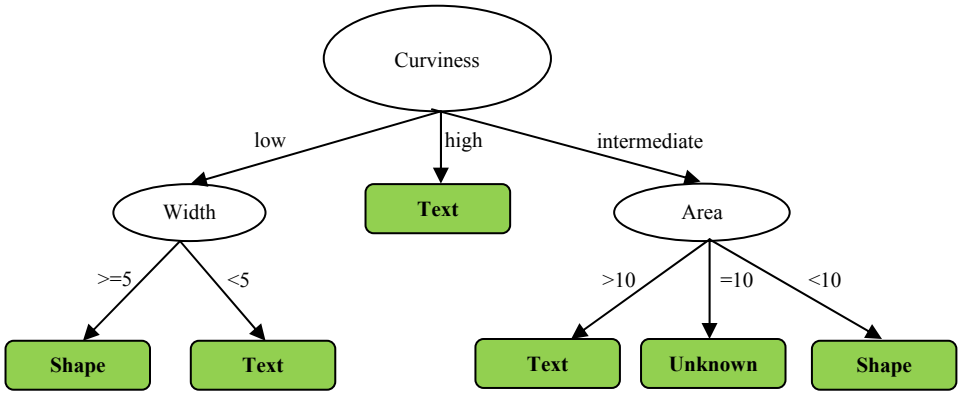


Figure 53 Standard Decision Tree Example

A REPTree for classification is built using information gain. Information gain measures the value of adding a certain feature as a node in the decision tree based on the training data. Features with high values of information gain are added to the tree. The REPTree is pruned using reduced-error pruning. This method of pruning splits the training data into two sets: a growing set for building the tree and a pruning set for testing how valuable each node is to the accuracy of the tree, where nodes that are not found to be valuable are removed.

Using an ensemble classifier such as Bagging has the advantage of drawing on multiple classifiers rather than a single one. Breiman (1996) states that Bagging is best for unstable procedures where small changes in training set result in large changes in the classifier's predictions.

For our experiments, the Bagging algorithm was tuned by varying the number of iterations that are performed. This parameter is indicative of the number of trees that can be produced. The default value for

this in Weka is 10 iterations. An experiment was run using ten-fold cross validation for Bagging with a REPTree base classifier at 10, 100, 500, 1000 and 5000 iterations. Paired t-tests show no significant difference in the results at each level of iterations, so no further tuning was applied. All the results are shown in Table 20. The highest results are produced at 500 and 1000 iterations, where on average (over ten folds of cross validation), 95.67% of the training dataset is correctly classified into text and shape.

Number of iterations	10	100	**500**	**1000**	5000
Average % Correct (sd)	95.31 (0.71)	95.64 (0.49)	**95.67** (0.58)	**95.67** (0.62)	95.57 (0.58)

Table 20 Bagging Results

6.2.2 RandomForest

RandomForest (Breiman 2001) essentially involves Bagging with Random Trees as the base learner. Random Trees are a type of decision tree; an example of a decision tree is shown in Figure 53 and a description can be found in Section 6.2.1. A Random Tree is built for each iteration of the algorithm using a bootstrap sample. A Random Tree constructs its nodes by choosing from a randomly chosen set of features and splitting the tree into branches using these features. The tree is not pruned. The final classification is based on a vote of all random trees built.

As with Bagging, using the RandomForest ensemble classifier is typically more successful than using single classifiers.

RandomForest was tuned by varying the number of iterations of the algorithm to 10, 100, 500, and 1000 iterations. It was observed that a higher number of iterations produced higher accuracies; therefore additional tests were added with 1500, 2000, 2500, and 3000 iterations. Paired t-tests show that there is no significant difference between any of the models. All results are shown in Table 21: the highest result is produced at 500 iterations where, on average (over ten folds of cross validation), 96.45% of the training dataset is correctly classified into text and shape.

Number of iterations	10	100	**500**	1000	1500	2000	2500	3000
Average % Correct (sd)	95.99 (0.82)	96.43 (0.54)	**96.45** (0.57)	96.44 (0.59)	96.40 (0.59)	96.44 (0.54)	96.39 (0.54)	96.36 (0.58)

Table 21 RandomForest Results

6.2.3 LogitBoost

LogitBoost (Friedman et al. 2000) is an implementation of additive logistic regression. A forward stage-wise process is used to build prediction models iteratively that complement each other. It is additive because it bases its predictions on an ensemble of all models generated.

The LogitBoost algorithm for a user specified number of iterations, *j*, uses the following steps.

1. Regression is used for numerical prediction; therefore the class of the training instances must first be translated from nominal (in our case text and shape) to numerical. The translated classes are used as the response variable, z_i, of the regression function, $f(x_i)$, where x_i is the i^{th} instance. The transformation used to do this is shown in Equation 2. For a two class problem, as in the text-shape division case, one class can be represented as $y_i^* = 1$ and the other as 0.

   The calculation of the probability of a class, $p(x_i)$, is shown in Equation 3 where $F(x)$ is the sum of the regression models' prediction, i.e. the sum of the $f_j(x)$, where j indexes the iterations. $F(x) = 0$ and $p(x_i) = ½$ for the first iteration.

$$z_i = \frac{y_i^* - p(x_i)}{p(x_i)\,(1 - p(x_i))}$$

Equation 2 Logit Transform

$$p(x) = \frac{e^{F(x)}}{e^{F(x)} + e^{-F(x)}}$$

Equation 3 Probability of a Class

2. The weights for each instance are calculated using Equation 4.

$$w_i = p(x_i)\,(1 - p(x_i))$$

Equation 4 Weighting of an Instance

3. A weighted least-squares regression model, $f(x_i)$ is fitted using the class values z_i, the weighting w_i, the feature values and a base learner. (This base learner must be able to handle numeric classes as it performs the regression). $P(x_i)$ is calculated using the regression models built in the previous iterations, F(x). Before any models have been built, it is ½.

   A shrinkage parameter can also be used as a multiplier for each classifier's prediction to avoid over-fitting.

4. To classify a new instance, the sum of the predictions of the all the generated regression models, $F(x)$, is calculated. If $F(x)$ is positive, the class is 1, and if it is negative, the class is 0.

Two base learners were investigated for LogitBoost: the Decision Stump (a one node decision tree) and the REPTree (described in Section 6.2.1). To begin, a preliminary ten-fold cross validation experiment was run to see if there were any significant differences between these base learners for LogitBoost. A paired t-test showed no significant difference between the two at 120 iterations of the algorithm. Based on these results, both trees were further investigated as base learners.

To further tune LogitBoost, various values for the number of iterations the algorithm performs and the shrinkage parameter were tested. As mentioned earlier, shrinkage is a parameter that can be tuned to avoid over-fitting the LogitBoost model to the training dataset. When a classifier is over-fitted, it reduces the likelihood of the model retaining the same level of accuracy on a new test dataset that had been originally achieved with training data. Small values for shrinkage reduce over-fitting.

The first run of experiments used LogitBoost with ten-fold cross validation and the following options:

- Base learner: Decision Stump or REPTree
- Number of iterations: 10, 100, 500, 1000, 5000
- Shrinkage: 1.0 (Weka default value), 0.1

It was observed from the results of paired t-tests that classifiers with a shrinkage value of 0.1 performed better than 1.0, particularly when using REPTree as the base learner. A higher number of iterations also resulted in more accurate models in general. Based on these observations, a second round of experiments was run with the following additional options:

- Number of iterations: 5500, 6000, 7000
- Shrinkage: 0.01

Using all combinations of the above options (for rounds one and two) resulted in 48 models for LogitBoost, 24 for each base classifier. The average results for these models are shown in Table 22. The results in black cells are those that are significantly more accurate, according to paired t-tests, than those in white cells: these models are able to correctly divide approximately 96.6% of the training dataset (using ten-fold cross validation) into text and shape. There is no significant difference between the results in black cells.

		Number of Iterations							
Base classifier	**Shrinkage**	10	100	500	1000	5000	5500	6000	7000
Decision Stump	0.1	87.89 (1.00)	92.71 (0.77)	95.38 (0.79)	96.10 (0.82)	96.70 (0.84)	96.69 (0.89)	96.62 (0.95)	96.56 (0.91)
Decision Stump	1.0	91.87 (1.04)	95.74 (0.58)	96.37 (1.00)	96.51 (0.79)	96.48 (0.59)	96.43 (0.56)	96.43 (0.59)	96.34 (0.66)
Decision Stump	0.01	86.48 (1.04)	87.73 (0.95)	90.74 (0.85)	92.66 (0.76)	95.31 (0.70)	95.45 (0.64)	95.54 (0.67)	95.82 (0.69)
REP Tree	0.1	94.84 (0.54)	96.67 (0.48)	96.69 (0.48)	96.69 (0.48)	96.69 (0.48)	96.69 (0.48)	96.69 (0.48)	96.69 (0.48)
REP Tree	1.0	96.01 (0.82)	96.03 (0.83)	96.03 (0.83)	96.03 (0.83)	96.03 (0.83)	96.03 (0.83)	96.03 (0.83)	96.03 (0.83)
REP Tree	0.01	93.23 (0.69)	95.06 (0.42)	96.56 (0.72)	96.62 (0.63)	96.62 (0.63)	96.62 (0.63)	96.62 (0.63)	96.62 (0.63)

Table 22 LogitBoost Results. The results in black cells are significantly better than all other results according to paired t-tests. There is no significant difference between those results in black cells.

The model with the highest level of accuracy uses a Decision Stump base learner, 5000 iterations, and a shrinkage value of 0.1. Paired t-tests show that it is significantly better than all other Decision Stump models, except that the models with shrinkage of 0.1 at 5500 and 6000 iterations are not significantly different. Comparing this model to the REPTree classifiers shows that there is no significant difference found for 13 of the REPTree models as shown in Table 22 (black cells). Further, smaller values of shrinkage produce good results, whereas those models with a shrinkage value of 1.0 are all significantly worse than the other LogitBoost models. In terms of the number of iterations, models with 10 iterations are clearly not optimal. One issue with this algorithm is a long training time. The fastest configuration that still produces significantly good results is LogitBoost using REPTree as the base learner, shrinkage of 0.1, and 100 iterations. Due to its advantage of fast training time while retaining high accuracy, this model was used for later stages of analysis.

6.2.4 LADTree

LADTree (Holmes et al. 2002) is an alternating decision tree using the LogitBoost strategy described in the previous section. An alternating decision tree has nodes that are added incrementally in each iteration of the algorithm. LogitBoost is used to grow the alternating decision tree by generating rules at each iteration. These rules translate to splitter and prediction nodes that get added to the tree. Prediction nodes are leaves unless a splitter node is added beneath it. Splitter nodes are where a decision must be made. For a two class problem, such as text-shape division, prediction nodes will contain a vector with two predictions – one for each class.

The classification of an instance is obtained by filtering the instance through the tree; it may lead to multiple leaves. The sum of the prediction values for each class is calculated and the instance is classified as belonging to the class with the highest sum.

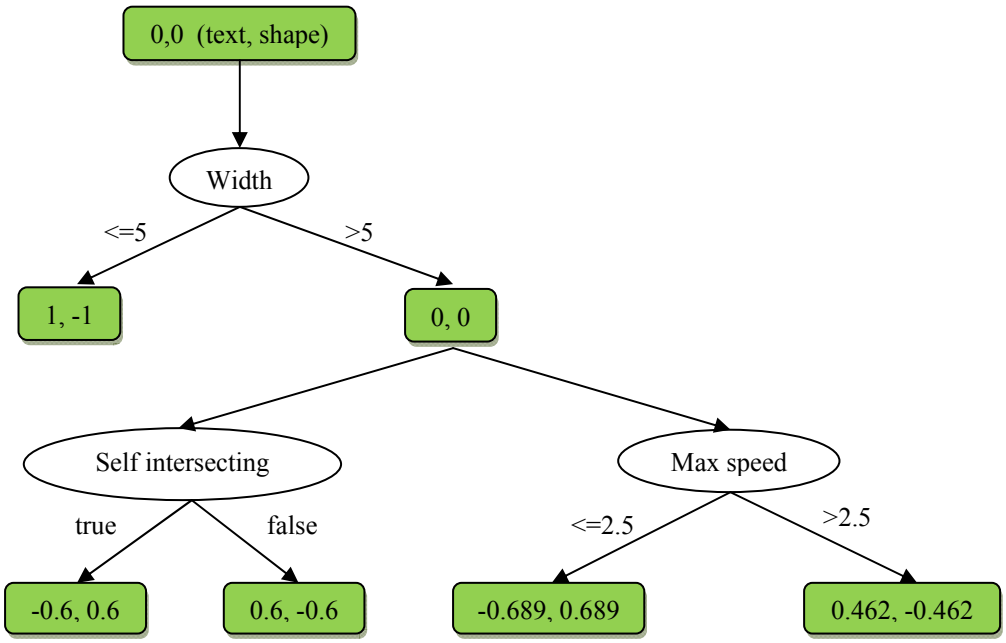


Figure 54 Logistic Alternating Decision Tree (LADTree) Example

A simple example of an LADTree is shown in Figure 54 where the classes are stroke = (text, shape). If a new instance has a width > 5, is self intersecting and has a max speed <= 2.5, we add the values for each class as follows:

stroke = text, 0 + (-0.6) + (-0.689) = -1.289

stroke = shape, 0 + 0.6 + 0.689 = 1.289

The highest prediction value is for stroke = shape: therefore this is the final classification of this instance.

We tuned the LADTree by varying the number of iterations of the algorithm to 10, 100, 500, 1000 and 1500 iterations. The number of iterations could not be increased any further due to computational time and memory constraints. Paired t-tests show that the LADTree with 1500 and 1000 iterations is significantly more accurate than the others, except for the model with 500 iterations which has no significant difference to 1000 iterations, but is significantly different to the 1500 iteration model. In summary, there is no significant difference between the LADTrees with 1500 and 1000 iterations and no significant difference between LADTrees with 1000 and 500 iterations. All results are shown in Table 23: the highest result is produced at 1500 iterations where on average 97.48% of the training dataset (using ten-fold cross validation) is correctly divided into text and shapes.

Number of iterations	10	100	500	1000	**1500**
Average % Correct (sd)	91.94 (0.92)	96.07 (0.93)	97.12 (0.54)	97.35 (0.50)	**97.48** (0.52)

Table 23 LADTree Results

6.2.5 LMT

LMT (Landwehr et al. 2005; Sumner et al. 2005) is a logistic model tree. It is a classification tree built with LogitBoost (described in Section 6.2.3) that has linear logistic regression models at the leaves of the tree. To grow the tree, LogitBoost is run on all the data for the first node. The data is then split according to class labels and the tree continues growing at child nodes by building on the LogitBoost model from its parent node in an iterative fashion. The tree stops growing if fewer than 15 instances are present at that node or if there are fewer than 2 examples of each class at the node. The tree is then pruned to avoid over-fitting. Classification is completed by sending the instance down the tree: when it reaches a leaf, the linear logistic regression model at that leaf is applied to obtain the class prediction.

The default parameters in Weka for LMT are sensible for this problem and do not require tuning. The result of ten-fold cross validation using the training dataset on LMT with default parameters is 94.85% (sd = 0.68). This means that on average 94.85% of the training dataset (using ten-fold cross validation) is correctly classified by LMT.

6.2.6 Multilayer Perceptron

The Multilayer Perceptron (Minsky et al. 1969; Rumelhart et al. 1986) is a neural network. There are generally three layers in the network: an input layer (one node for each feature), a hidden layer and an output layer (classification nodes) as illustrated in Figure 55. There are connections between the nodes in

each layer, going from the input layer to the hidden layer and finally to the output layer. Each connection has a weight applied to it. Each node has an activation function (sigmoid function in this case) which is used to transform the weighted sum of the inputs into a node to determine the output of that node.

To train the Multilayer Perceptron, the best value for the weights must be found; this is achieved by back propagation. First, each instance in the training set is sent through the network, weights are initially assigned randomly. The weights are applied to the feature values for that instance and are transformed using the activation functions to produce a value at the output node. The value produced at the output node is compared to the expected class value of the instance. Using this information, the error can be calculated as the squared difference between the actual and expected values. Changes in weight are obtained by minimising these errors: in other words, the squared error of the output is minimised. This process is continued by moving backwards through the network. Once this process is completed for all training instances, all the changes in a particular weight are summed and multiplied by a specified learning rate; this is then subtracted from the current weight to find the final weight value.

Classifications for new instances are then based on the Multilayer Perceptron with these final weights.

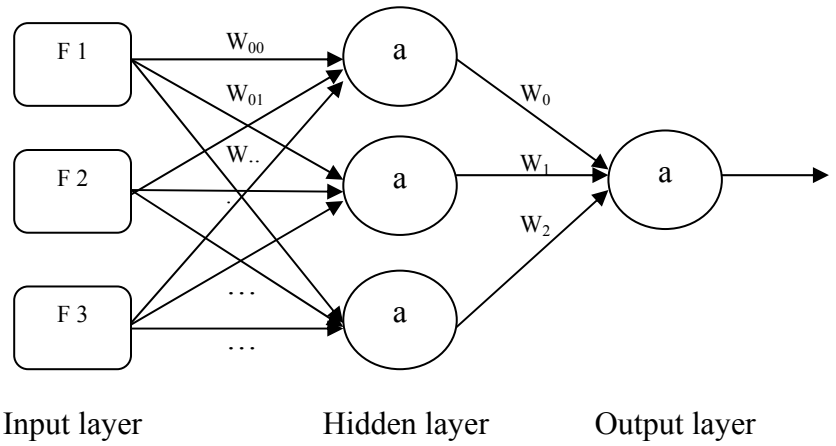


Figure 55 Example of a Multilayer Perceptron. 'F 1-3' represent the features used, 'w' represents the weights for each branch and 'a' represents the activation function.

The default parameters in Weka for Multilayer Perceptron are sensible for this problem and do not require tuning. The result of ten-fold cross validation using the training dataset on Multilayer Perceptron with default parameters is 95.02% (sd = 0.78). This shows that on average, (over ten folds of cross validation) 95.02% of the training dataset is correctly classified by Multilayer Perceptron.

6.2.7 SMO

SMO (Hastie et al. 1998; Platt 1999; Keerthi et al. 2001) trains a Support Vector Machine using the Sequential Minimal Optimisation algorithm. For a two class linear problem, a Support Vector Machine identifies a hyper-plane that separates the classes. The hyper-plane is situated such that the distance between the nearest instances of each class is maximised: this is illustrated in Figure 56. The nearest instances to this hyper-plane are support vectors; all other training instances are not important as they do not affect the position of the hyper-plane.

This problem becomes more complex when estimating non-linear boundaries. Kernel functions are used to create non-linear mappings. Also a complexity parameter, C, is used for non-linear problems. This parameter allows a trade-off between misclassifying some instances by allowing for a wider distance between classes. The solution is implemented using numerical quadratic programming.

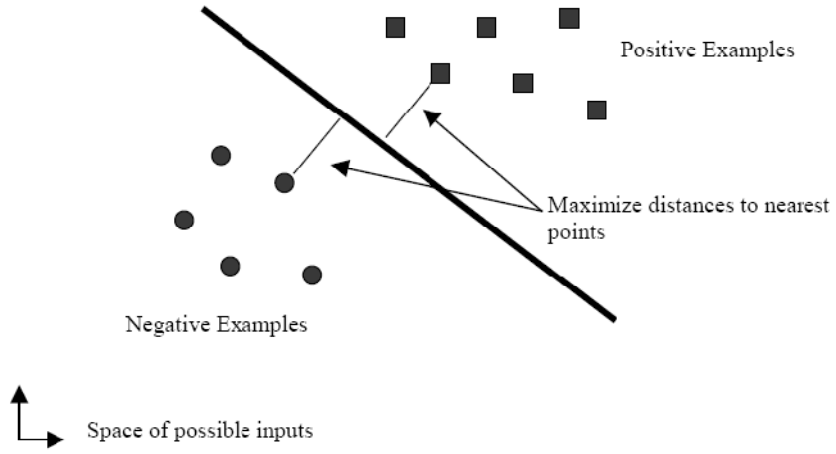


Figure 56 Illustration of a Linear Support Vector Machine from (Platt 1999)[11].

The Sequential Minimal Optimisation (SMO) algorithm is a faster version of the traditional Support Vector Machine that uses a more complex programming step. We use an RBF Kernel as a parameter of SMO whereby a support vector machine can be viewed as a type of neural network similar to a Radial Basis Function (RBF) network (Witten et al. 2005). RBF networks are similar to Multilayer Perceptrons (described earlier) except that the hidden layer uses the RBF activation function (bell-shaped Gaussian activation function) instead of the sigmoid function. This computation is based on the distance of the instance from the point represented by the node in the hidden layer. The gamma parameter of the RBF

[11] Image obtained from Platt, J. (1999). Fast training of support vector machines using sequential minimal optimization. Advances in Kernel Methods - Support Vector Learning.185-208. Reproduced with permission from the author.

kernel can be tuned to control the linearity of the mapping done by the RBFKernel. High values of gamma result in almost linear mappings.

SMO is a more complicated classifier to tune. There are two parameters that can be tuned: the complexity value, C, of SMO and the gamma value of the RBF kernel used by SMO. To find the best model, the GridSearch function (Witten et al. 2005) in Weka was used. This allows two parameters of an algorithm to be optimised by setting a maximum, minimum, base value, and step value for how much a parameter can increase by for each test. One of the main advantages of GridSearch is that the parameters of interest do not have to be first level parameters. For example, gamma is not a first level parameter as it is a value used by the RBF kernel, where the RBF kernel is a parameter of SMO.

The optimal value found for complexity is 100, with a gamma value of 0.1. The result of ten-fold cross validation using this model on the training data is 96.41% (sd = 0.73). This result shows that on average (over ten folds of cross validation) SMO correctly classifies 96.41% of the training dataset.

6.2.8 Summary of the Best Results

The best results of ten-fold cross validation for each classifier on the training dataset are shown in Table 24. Paired t-tests show that LogitBoost and LADTree are significantly better than the other classifiers. There is no significant difference between LogitBoost and LADTree. This is not surprising as LADTree uses the LogitBoost strategy.

Classifier	Average % Correct (sd)	Configuration
LADTree	97.48 (0.52) *	Iterations: 1500
LogitBoost	96.70 (0.84) *	Base classifier: Decision Stump Shrinkage: 0.1 Iterations: 5000
RandomForest	96.45 (0.57)	Iterations: 500
SMO	96.41 (0.73)	Complexity: 100 Kernel: RBF kernel Gamma: 0.1
Bagging	95.67 (0.58, 0.62)	Iterations: 500 and 1000
MultilayerPerceptron	95.02 (0.78)	Default
LMT	94.85 (0.68)	Default

Table 24 Best Results Obtained from the Selected Classifiers. Those with a * are significantly more accurate than the others according to paired t-tests.

In terms of the features that are used by the top classifiers, LADTree has a total of 107 distinct features while LogitBoost has 100 features of the 114 available in the feature library. Ninety eight features are

common to both models - which we anticipated, given that they use a similar strategy and produce results that are not significantly different from each other.

Closer inspection of the LADTree shows that the features in Table 25 are present at the top levels of the tree. This position indicates that they are more discriminating than those at lower levels. Feature 1 is the result obtained using our previous divider (Patel et al. 2007). This is considered as a first parse of the stroke to find its probable class. Feature 2 breaks the stroke into fragments, identifies the longest fragment, constructs an enclosing rectangle around this fragment (that is not necessarily axis aligned like a traditional bounding box), and measures the length of the longest side of this rectangle. Features 3, 4, and 5 are related to the spatial context of the current stroke by measuring characteristics of strokes that are nearby. Feature 3 measures the average length of strokes close to the current stroke's end point. Feature 4 measures the average density of close strokes. Feature 5 measures the smallest distance to another stroke from the current stroke's start point. Finally, there are two features measuring curvature, with feature 6 measuring the sum of the squared value of the angle at each point of the stroke and feature 7 the total angle traversed by the stroke divided by the stroke length.

Feature	**Category**
1. Divider result (Patel et al. 2007)	Divider Results
2. Length of the longest side of the enclosing rectangle of the largest fragment (Bishop et al. 2004)	Size
3. Average length of close end point strokes (Ao et al. 2007)	Spatial context -> Size
4. Average density of close strokes (Ao et al. 2007)	Spatial context ->Density
5. Smallest distance between strokes from start point (new)	Spatial context -> Location
6. Squared angle (Rubine 1991)	Curvature
7. Total angle and length ratio (Long et al. 2000)	Curvature

Table 25. Features at the Top Levels of the LADTree

The entropy feature proposed by Bhat et al (2009) for dividing first appears on the third level of the LADTree.

We are unable to identify similar features in the LogitBoost models as this is composed of a collection of Decision Stumps (one node trees) rather than one large tree.

6.3 Feature Selection

The use of feature selection was investigated to refine the feature set to those that contribute the most to building an accurate divider model. With a large feature library, it is possible that many features do not add any value to the recogniser or may indeed be detrimental to the recognition accuracy. Filtering these features may result in higher accuracy and faster algorithm training and testing time, as a smaller number of feature calculations have to be made.

The process of building a divider with feature selection is illustrated in Figure 57 where feature selection acts as a filter, eliminating non-contributing features from the library to produce a subset of good features to use in training a classifier. Three methods of feature selection were used: an Attribute Selected Classifier with Wrapper, Attribute Selected Classifier with Relief F and a hybrid method. The exploration of these methods is described in the remainder of this section.

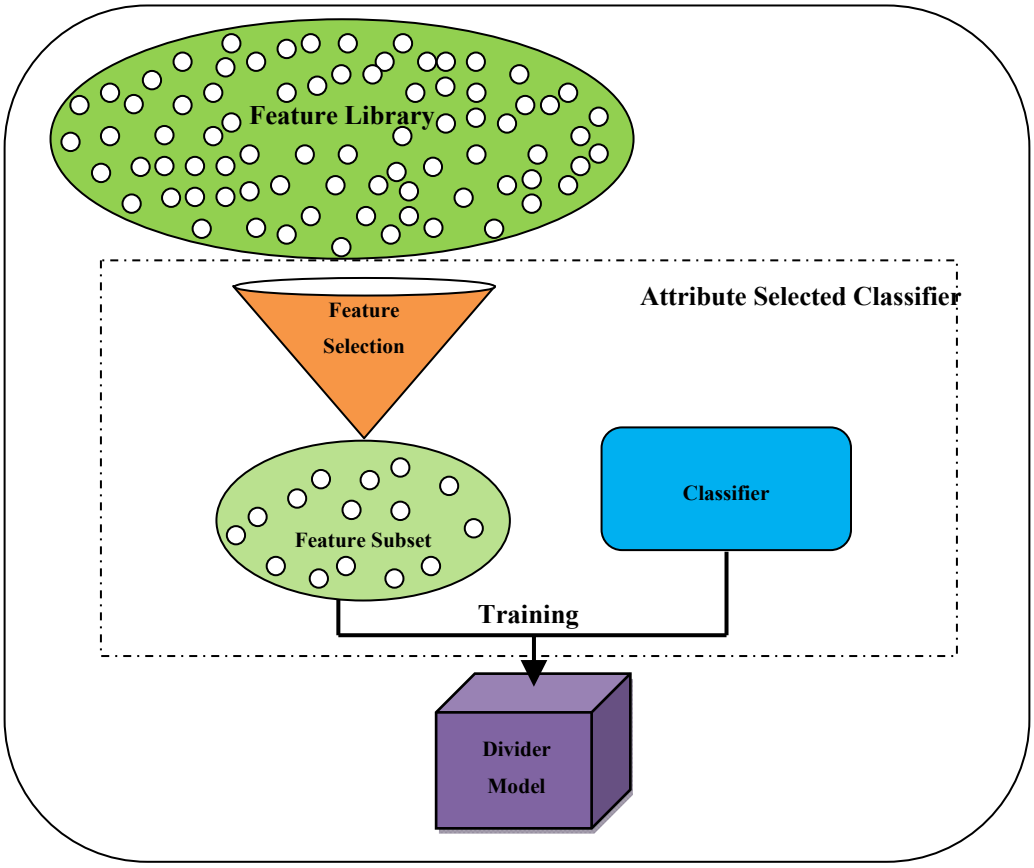


Figure 57 The Feature Selection Process

6.3.1 Attribute Selected Classifier with Wrapper

The Attribute Selected Classifier with Wrapper (Kohavi et al. 1997) in Weka was used as a method of feature selection. The benefit of using the Attribute Selected Classifier is that it allows feature selection and classifier training to be completed in one process, as illustrated in Figure 57, which simplifies the process of applying feature selection to classifiers. It works by finding a subset of features using the chosen feature selection method. It then uses this feature subset to train the specified classifier and output a classifier model.

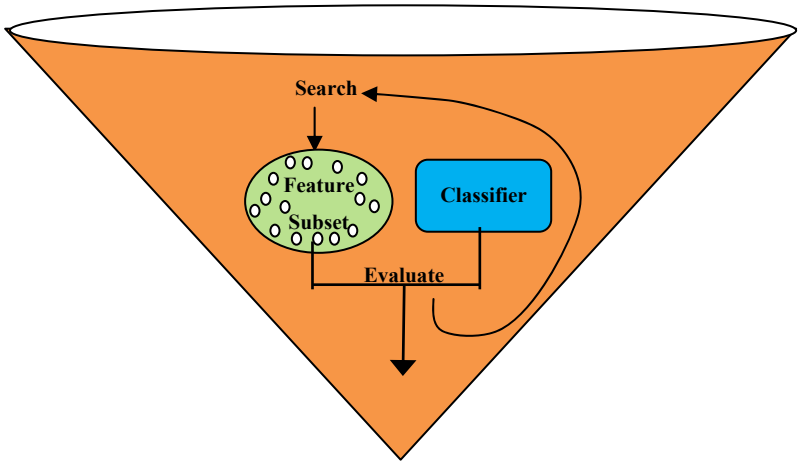


Figure 58 The Process of Feature Selection using a Wrapper

In addition, a Wrapper (Kohavi et al. 1997) can be used within the Attribute Selected Classifier as the feature selection method. The process of feature selection using a Wrapper is shown in Figure 58. First, a search of the feature library is performed to select a feature subset. The selected feature subset is used to train a classifier and the results of this training are used to evaluate the success of the feature subset. The process is repeated with a new feature subset. This process continues until the search of the feature library is complete (which varies according to the chosen search method) and the best feature subset has been found.

The Wrapper can take any classifier and use it to evaluate the feature selection. The main advantage of using the Wrapper here is that the same classifier can be used to evaluate the feature subset (shown in Figure 58) and also train the final classifier (shown in Figure 57) so we can expect good results. For example, within the Attribute Selected Classifier, the LADTree can be used via the Wrapper to find the best feature subset and then this chosen feature subset is used to train an LADTree for the final classifier model. A disadvantage of this method is that it has a very long training time.

The Attribute Selected Classifier with Wrapper was used for all seven classifiers with ten-fold cross validation. The classifier used within the Wrapper always matched the base classifier. The parameters used for each classifier are shown in Table 26. The parameters were chosen as optimal from the tests described in Section 6.2.

For algorithms where multiple models are not significantly different from each other, the model with faster training time was chosen. The exception was the LADTree; the model with 500 iterations was chosen, despite it being significantly lower in accuracy to the 1500 iteration model. This is because the time taken to train the 1500 iteration model is significantly longer than all others and a trade-off between time and accuracy had to be made.

The search method used is linear forward selection as this is known to be faster than other search methods without loss of accuracy (Gütlein et al. 2009). The parameter k, the number of top ranked attributes that are taken into account by the search process, for linear forward selection was set to 10. Gütlein & Frank et al (2009) report that for problems with few classes, as in our case with two classes, high accuracy is achieved when k<=10, using higher values of k can lead to over-fitting the model to the training data.

Classifier	Parameter settings
Bagging	Number of iterations = 500
LADTree	Number of iterations = 500
LMT	*Default*
LogitBoost	Number of iterations = 100 Base classifier = REPTree shrinkage = 0.1
Multilayerperceptron	*Default*
RandomForest	Number of iterations = 500
SMO	Kernel = RBFKernel complexity = 100 gamma = 0.1

Table 26 Parameter Settings used for Feature Selection and Ensembles

The results of ten-fold cross validation using the Attribute Selected Classifier with Wrapper on each classifier is shown in Table 27. The first results column shows the accuracy of the original tuned models with the parameter settings shown in Table 26, without feature selection. These results are compared with each method of feature selection to ascertain if any significant improvements in accuracy have been made. The remaining columns show results for each method of feature selection. Results for the Attribute Selected Classifier with Wrapper are shown in the second results column.

The overall result was that **all** of the results are significantly **worse** than the accuracy obtained by using the tuned classifier without feature selection, according to paired t-tests. This suggests that in fact all of the ink

features captured do add some value to each classifier. Therefore, reducing the feature set by using this feature selection method decreases the accuracy of all of these classification models. Unfortunately, the LADTree experiment was not able to be completed due to an unknown error in the software. However, judging from the poor results for all other methods, we believe LADTree would have performed no better than the tuned model without feature selection.

Average % correct (sd)	Original tuned models	Attribute selected (Wrapper)	Attribute selected (Relief F)	Hybrid feature sets			
				Top 20		Top 50	
				1	2	1	2
LADTree	97.12 (0.54)	-	97.12 (0.54)	94.58 (0.62)	92.78 (1.01)	96.40 (0.63)	95.85 (0.82)
LogitBoost	96.67 (0.48)	90.34 (0.76)	96.51 (0.58)	94.83 (0.56)	92.99 (1.05)	96.04 (0.52)	95.99 (0.75)
RandomForest	96.45 (0.57)	93.58 (0.45)	96.37 (0.68)	94.80 (1.02)	93.40 (0.88)	96.27 (0.50)	96.12 (0.68)
SMO	96.41 (0.73)	89.24 (1.13)	96.52 (0.73)	93.21 (1.13)	89.94 (1.04)	95.76 (0.62)	94.80 (0.74)
Bagging	95.67 (0.58)	90.12 (0.99)	95.61 (0.60)	93.72 (1.00)	92.83 (0.96)	95.32 (0.73)	95.01 (0.84)
MLP	95.02 (0.78)	89.38 (1.20)	95.06 (0.51)	92.96 (1.10)	91.02 (0.83)	94.55 (1.10)	94.54 (1.11)
LMT	94.85 (0.68)	89.51 (0.81)	94.85 (0.68)	93.38 (0.96)	91.74 (0.92)	94.77 (0.80)	93.90 (0.90)

Table 27 Results of Feature Selection. Black cells denote results that are not significantly different to the tuned model of that particular algorithm according to paired t-tests. White cells denote results with significantly worse results than the tuned model for that particular algorithm according to paired t-tests.

6.3.2 Attribute Selected Classifier with Relief F

Other feature selection methods were also explored. In a related project (Blagojevic et al. 2010) we found Relief F (Kira et al. 1992; Kononenko 1994; Robnik-Sikonja et al. 1997) to be one of the best of the eight ranking methods tested from Weka's feature selectors.

Relief F originates from instance based learning (Kira et al. 1992; Kononenko 1994; Robnik-Sikonja et al. 1997). It judges the value of a feature by considering that relevant features should be able to distinguish among instances from different classes while still grouping instances belonging to the same class together. An instance from the dataset is picked at random; the k nearest neighbours of this instance from the same category are then found (nearest hit instances) as well as the k nearest neighbours of this instance from another category (nearest miss instances). A weighting for that feature is then determined based on the difference between the feature values for each of the nearest hit instances and the chosen instance minus the same difference for the nearest miss instances. This is repeated for all instances and an average weighting for each feature is found. Relevant features must have a weight higher than a given threshold.

The Attribute Selected Classifier was used once again on all seven classifiers, but this time using Relief F as the feature selection method, rather than the Wrapper. Relief F requires the Ranker search method (Witten et al. 2005) to be used as it is a single attribute evaluator rather than a subset evaluator. The Ranker simply takes each individual evaluation from Relief F and ranks the features using this information. This contrasts the linear forward selection search method used earlier, which requires a subset evaluator to rank individual features initially and the same subset evaluator to judge feature subsets (Gütlein et al. 2009). Since Relief F is not a subset evaluator it cannot be used with linear forward selection.

The results for each classifier using this method of feature selection are shown in the third results column of Table 27. Paired t-tests show for all classifiers that the original tuned models are not significantly different from the models built with Relief F attribute selection. In fact for two of the seven models the average accuracy over ten folds is exactly the same. Therefore, the Attribute Selected Classifier with Relief F does not make any significant improvement to the accuracy of the dividers when compared with the classifiers without feature selection.

6.3.3 Hybrid Method

A subset of features was also compiled by merging the results of as many feature selection algorithms as possible. This method was chosen because we believe choosing features using only one feature selection method may run the risk of ignoring features that may be very important. We hypothesised that this problem might be avoided by merging the results of many feature selectors and building a subset of features based on this merger.

Eighteen feature selection algorithms were run from Weka on the training dataset: they are listed in Appendix B. These algorithms were found to rank features in one of two ways. One way to rank the features is based on how many folds the feature is chosen for. For example, when using ten-fold cross validation, ten subsets are chosen and an aggregate of those subsets forms the final result. If the feature is chosen for eight of the ten subsets then it will have a ranking of eight (or 80%). The other method of ranking is based on average merit, which indicates the average importance of that feature over all ten folds of cross validation based on the results of the particular evaluator used.

Due to the difference in feature ranking, two feature sets were formed. Feature set 1 is based on the number of folds the feature is present in while feature set 2 is based on the sum of the average merit.

Eleven of the 18 feature selectors rank according to fold number; the results from these selectors were merged by calculating the sum of the number of folds for each feature and ranking these. This is similar to the method of feature selection used by (Paulson et al. 2008b). The remaining seven feature selection algorithms rank according to average merit: the results from these were merged by calculating the sum of the average merit for each feature and ranking these. A full list of these feature sets can be found in Appendix B.

Chang conducted a study of numerous feature selection algorithms for basic shape recognition, using my feature library, and found that an optimal number of features for his problem is 20. Adding more features does not change the accuracy of an algorithm by much (Blagojevic et al. 2010). Note that this study was an extension of Chang's Master's project and is not part of this thesis although the feature library published here is. Further details on this study are found in (Blagojevic et al. 2010).

The top 20 from both feature sets were selected and also the top 50 features to test against the chosen seven data mining algorithms described previously. There are three common features in feature sets 1 and 2's top 20. 27 of 50 features are common to both feature sets 1 and 2's top 50 subsets. The results of each subset for each classifier are shown in Table 27. The results of the top 20 for both feature sets 1 and 2 are significantly worse than the tuned classifiers with no feature selection according to paired t-tests. For feature set 1's top 50, only LogitBoost and SMO are significantly worse than the tuned models; there are no significant differences found for the other classifiers using this feature set. For feature set 2's top 50, all are significantly worse than the tuned model except for Multilayer Perceptron (MLP) where there is no significant difference.

In summary, considering all the feature selection methods that were investigated, none are significantly more accurate than the original tuned classification models using the full ink feature set. Based on these results, we believe that we need to use all features in the library; otherwise we risk decreasing the accuracy of the classifiers.

6.4 Ensembles

The use of ensembles was also investigated to enhance the results we had already obtained. An ensemble uses more than one classifier to predict the class of an unknown instance by aggregating the results of each classifier in some way. One option for building ensembles is to use the Vote function (Kittler et al. 1998; Kuncheva 2004) provided by Weka. Two or more classifiers can be specified within the Vote function; such classifiers are trained using the training dataset and then used to classify test data. Classification is done by taking a vote of all classifiers' predictions. There are numerous ways of combining the votes. We chose to use the average of the probability estimates from the classifiers to obtain the overall classification as this was the most suitable setting available for this case. The higher the accuracy of the individual classifiers, the better the voting combination will be. Also, with some knowledge of the strengths and weaknesses of each classifier, a more robust voting combination can be chosen.

To begin composing classifier combinations, the list of tuned algorithms were ranked according to the results of paired t-tests on the ten-fold cross validation experiments described in Section 6.2. That ranking is shown in Table 24. Combinations are composed of the top 7 algorithms (all algorithms), the top 6, top 5, top 4, top 3 and top 2. The algorithms that made up each combination are shown in Table 29 (see combinations 1 to 6) along with the results of ten-fold cross validation using the training dataset (shown in the right hand side column). Also, the parameters for each algorithm, obtained from the tuning steps in Section 6.2, have been used here, as described in Table 26.

To further inform the choice of classifier combinations, each individual classifier was tested on the ER/Process diagram dataset, described in Section 5.5.2, with special attention drawn to their performance on each shape class. The ER/Process diagram dataset contains a total of 7062 strokes, with 4817 text strokes and 2245 shapes strokes. The results in Table 28 show clearly that a lower proportion of shapes, between 80-89%, are correctly classified than text, which has a correct classification rate between 97-99%. Obtaining the classifiers' performance on each shape class allows us to identify where the strengths and weaknesses lie and try to maximise the potential of a voting system by using combinations with strong results for each shape class. For example, we hypothesised that if classifier A performs well on rectangles but badly on arrows, then it would be beneficial to pair this with classifiers that perform well on arrows to obtain a good voting combination. The results of these experiments are presented in Table 28.

Classifier	Arrow	Rectangle	Diamond	Ellipse	Lines	% Shapes Correct	% Text Correct	% Total Correct
LADTree	55.56%	98.24%	99.34%	99.42%	86.87%	88.51%	98.28%	95.17%
LogitBoost	64.67%	97.71%	99.34%	99.42%	82.09%	88.37%	98.11%	95.02%
SMO	64.96%	97.80%	98.60%	98.83%	84.48%	89.00%	97.01%	94.46%
RandomForest	55.16%	96.12%	96.26%	98.60%	92.08%	84.28%	99.17%	94.37%
LMT	51.57%	95.70%	97.70%	97.95%	87.01%	86.86%	97.34%	94.01%
MLP	57.10%	95.59%	95.33%	97.47%	92.82%	84.50%	97.66%	93.41%
Bagging	40.17%	95.20%	94.40%	98.25%	72.99%	80.36%	98.51%	92.74%

Table 28 Summary of Testing Tuned Models with ER/Process Diagrams

	LADTree	LogitBoost	Random Forest	SMO	Bagging	MLP	LMT	% Correct (sd)
1. top 7	x	x	x	x	x	x	x	96.94 (0.57)
2. top 6	x	x	x	x	x	x		96.95 (0.45)
3. top 5	x	x	x	x	x			97.14 (0.53)
4. top 4	x	x	x	x				**97.23 (0.51)**
5. top 3	x	x	x					96.76 (0.56)
6. top 2	x	x						96.83 (0.54)
7. LLS	x	x		x				**97.25 (0.43)**
8. LLSL	x	x		x			x	**97.30 (0.42)**
9. LLSRL	x	x	x	x			x	97.17 (0.47)
10. LogitSRL		x	x	x			x	96.87 (0.51)
11. LadSRL	x		x	x			x	97.30 (0.40)
12. LogitSR		x	x	x				96.81 (0.46)
13. LadSR	x		x	x				**97.36 (0.49)**
14. LadSL	x			x			x	97.14 (0.47)
15. LogitSL		x		x			x	96.73 (0.63)
16. LadRL	x		x				x	97.02 (0.46)
17. LogitRL		x	x				x	96.58 (0.58)
18. LLL	x	x					x	96.73 (0.52)
19. LLSRM	x	x	x	x		x		96.94 (0.38)

Table 29 Voting Combinations Chosen to Test. The results of the top four ensembles are highlighted in green.

Table 28 shows the results for each shape class. All algorithms are able to classify Rectangles, Diamonds, Ellipses and Text very well for the ER/Process diagrams dataset, where correct classification rates range from 94-99%. The areas of weakness are in classifying arrows and lines. For arrows, correct classification rates range from a very low 40% to 64%. LogitBoost and SMO perform the best on Arrows and Bagging gives the worst results for this shape class. The correct classification rates for lines range between 72% and 92%. RandomForest and MLP have the highest accuracy rate for this shape class and Bagging the lowest.

Bagging has the lowest correct classification rates for every class except Ellipse and Text, and the lowest overall percentage correct at 92%. It is possible that this algorithm is over-fitted to the training data and therefore does not perform as well on test data. Overall, LADTree and LogitBoost still produce the most accurate results with 95% of strokes correctly classified.

Using the above information, additional combinations were formed for testing with the voting algorithm; these combinations are shown in Table 29 (see combinations 7 to 19) with results of ten-fold cross validation (shown in the right hand side column). Table 30 gives the reasoning behind each combination choice.

<table>
<tr><th>Combination</th><th colspan="2">Reason for combination choice</th></tr>
<tr><td>1.</td><td colspan="2">Using all algorithms</td></tr>
<tr><td>2.</td><td colspan="2">Using the top six algorithms based on the ranking of the algorithms on the training dataset as shown in Table 24.</td></tr>
<tr><td>3.</td><td colspan="2">Using the top five algorithms based on the ranking of the algorithms on the training dataset as shown in Table 24</td></tr>
<tr><td>4.</td><td colspan="2">Using the top four algorithms based on the ranking of the algorithms on the training dataset as shown in Table 24</td></tr>
<tr><td>5.</td><td colspan="2">Using the top three algorithms based on the ranking of the algorithms on the training dataset as shown in Table 24</td></tr>
<tr><td>6.</td><td colspan="2">Using the top two algorithms based on the ranking of the algorithms on the training dataset as shown in Table 24</td></tr>
<tr><td>7.</td><td colspan="2">Using the top three algorithms ranked according to their overall accuracy on the ER/Process diagrams dataset, as shown in Table 28.</td></tr>
<tr><td>8.</td><td colspan="2">Using the top 3 algorithms ranked according to their overall accuracy on the ER/Process diagrams dataset plus LMT.</td></tr>
<tr><td>9.</td><td colspan="2">Using the top five algorithms ranked according to their overall accuracy on the ER/Process diagrams dataset, as shown in Table 28.</td></tr>
<tr><td>10.</td><td colspan="2">Separate LADTree and LogitBoost (as they produce similar results) and combine each separately with other classifiers (based on test data ER & Process results). This combination is using the top four classifiers excluding LADTree.</td></tr>
<tr><td>11.</td><td colspan="2">Separate LADTree and LogitBoost (as they produce similar results) and combine each separately with other classifiers (based on test data ER & Process results). This combination is using the top four classifiers excluding LogitBoost.</td></tr>
<tr><td>12.</td><td>Using LogitBoost, SMO and RandomForest.</td><td rowspan="6">Separate LADTree and LogitBoost (as they produce similar results) and combine each separately with other classifiers (based on test data ER & Process results).
These combinations are made up of triples, using either LADTree or LogitBoost and every possible pairing of SMO, RandomForest and LMT.</td></tr>
<tr><td>13.</td><td>Using LADTree, SMO and RandomForest.</td></tr>
<tr><td>14.</td><td>Using LADTree, SMO and LMT.</td></tr>
<tr><td>15.</td><td>Using LogitBoost, SMO and LMT.</td></tr>
<tr><td>16.</td><td>Using LADTree, RandomForest and LMT.</td></tr>
<tr><td>17.</td><td>Using LogitBoost, RandomForest and LMT.</td></tr>
<tr><td>18.</td><td colspan="2">Using LADTree, LogitBoost and LMT</td></tr>
<tr><td>19.</td><td colspan="2">This combination includes the top performing classifiers for each shape class with the ER/Process diagrams dataset.</td></tr>
</table>

Table 30 Reasoning behind Voting Combinations.

Paired t-tests were used to compare the different voting combinations; in particular the ranking function provided by Weka Experimenter. The highest ranked combination is (8), which is composed of LogitBoost, LADTree, SMO and LMT. The next highest are (7) and (13), followed by combination (4). The correct classification rates of these combinations range from 97.23% to 97.36%. These combinations all include LADTree and SMO classifiers. A possible reason why these classifiers perform well in combination is because SMO provides strength in classifying arrows, which is an area of weakness for the LADTree according to the results in Table 28. However, paired t-tests show that these results are not significantly different from using the tuned LADTree or LogitBoost classifiers alone.

6.5 Second Round Analysis

The results in Table 28 show that the greatest area of weakness for all classifiers trialled is with arrows, where recognition rates range from 40% to 64%. An example of a process diagram with many misclassified arrows is shown in Figure 59. The results for lines are also very low, ranging from 72% to 92% for the ER/Process diagrams dataset. These observations motivated a second round of analysis, beginning with a search for features that can specifically identify these connectors.

More recent features were found in related work and added to the feature set. These were found after the initial set had been compiled. These features are described in the next section with the results of analysis using these features. Following this, a strategy for a second parse of the results is described: the goal of this second parse is to correct misclassifications of arrows that may occur in the initial classification.

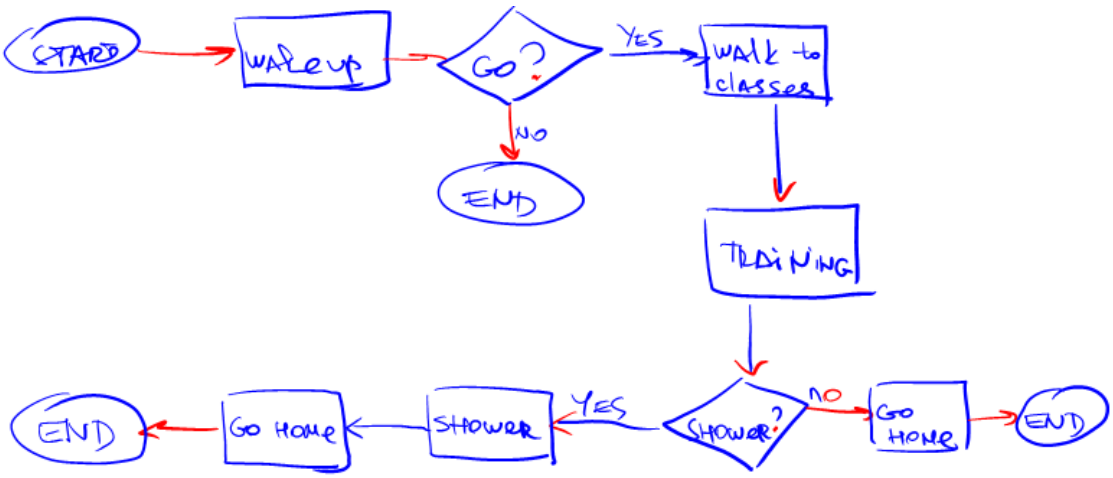


Figure 59 Misclassified Connectors. The strokes in red show those that are misclassified by the recogniser and strokes in blue are correctly classified.

6.5.1 Additional Features

The following features have been added to the feature library based on research of previous work on identifying connectors.

1) Is Straight Line

This feature identifying straight lines was found in Willems et al's work (2008). A line segment is made from the point 1/3 of the way into the stroke to the point 2/3s into the stroke, this corresponds to k_1 and k_2 in Figure 60. The distance of all points in the stroke to this line segment must be smaller than a chosen threshold for the stroke to be identified as a straight line. A threshold was determined based on empirical observations. Informal tests show that this feature recognises straight lines well. In terms of the feature taxonomy, this feature falls into the curvature category.

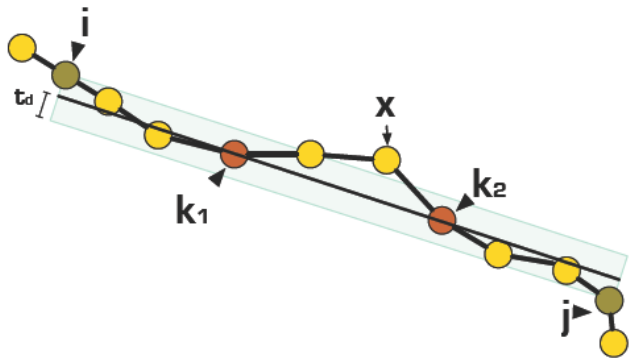


Figure 60 Straight Line Identification (Willems et al. 2008)[12].

2) Number of Cups

Willems et al (2008) also have a feature to identify U shapes in strokes, known as cups. Cups are known as a fundamental characteristic of handwriting. Strokes are examined using a sliding window: where if the angle between the first and last segments of the window is above a certain threshold, the window is thought to contain a cup, as illustrated in Figure 61.

This feature has been implemented with a threshold based on empirical observations. This has been included because any stroke that does not contain cups could be considered close to a straight line; but its limitation may be with arrowheads as these may contain a cup. This feature is part of the curvature category in the feature taxonomy.

[12] Image obtained from Willems, D. and R. Niels (2008). Definitions for features used in online pen gesture recognition, NICI, Radboud University Nijmegen. Reproduced with permission from the author.

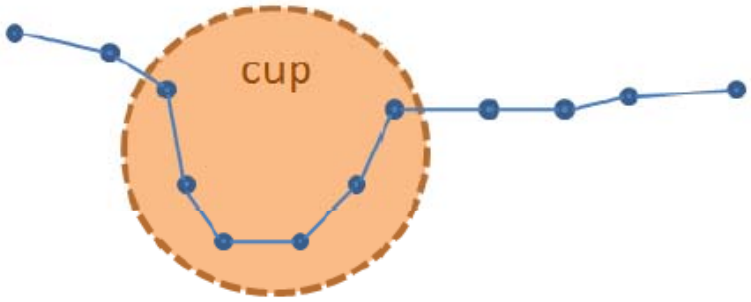


Figure 61 Example of a Cup (Willems et al. 2008)[13].

3) Bounding Box Maximum

This feature calculates the maximum of the stroke's bounding box width and height. This was found to improve the accuracy of our previous divider (Patel 2007; Patel et al. 2007) by Garcia (2010). The previous divider uses bounding box width as a measure of stroke size where, in addition to other feature conditions, small strokes are more likely to be text than shapes. This is fine for horizontal strokes, but vertical lines have a very small bounding box width and therefore are more likely to be classified as text. For example, in the diagram shown in Figure 62, the approximate bounding boxes are shown for the two arrow shafts. The horizontal arrow shaft has a large bounding box width and therefore should be correctly classified as a shape. The vertical arrow shaft on the other hand has a very small bounding box width and so may be misclassified as text. Garcia avoided this problem by taking the maximum of the height and width of the stroke's bounding box. This feature may help solve the problem of identifying vertical connectors such as the one shown in Figure 62. This feature is part of the size category with regards to the feature taxonomy.

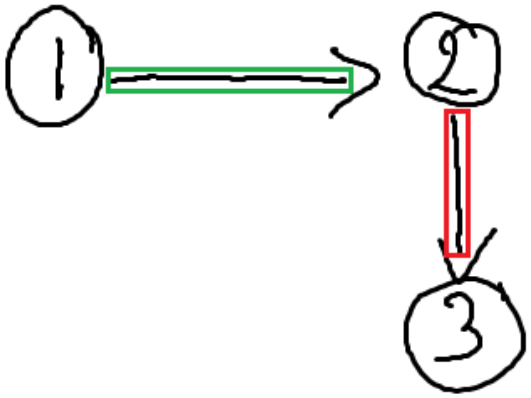


Figure 62 Bounding Box Maximum Example for Horizontal and Vertical lines.

[13] Image obtained from Willems, D. and R. Niels (2008). Definitions for features used in online pen gesture recognition, NICI, Radboud University Nijmegen. Reproduced with permission from the author.

4) Second Parse Classification: Is Arrowhead

Another way of influencing classification results is by adding a second parse to the classification process. A second parse uses information from a classifier's initial classification results as further contextual information to try to correct misclassifications.

A feature was implemented to determine if a stroke is an arrow, based on Freeman et al (2007) and Kara et al's (2004) work. This feature can only be used in a second parse of the diagram to correct any misclassifications, as it draws on information from the initial classification of a stroke.

This feature assumes the arrow is drawn in two strokes: one stroke for the arrowhead and the other for its shaft, as we have observed that these arrows are misclassified frequently. It also assumes that the arrow shaft has already been correctly classified because we have observed that the current divider misclassifies arrowheads more frequently than the shaft for two stroke arrows. Only the strokes that have been classified as text in the first parse of the divider are considered so that arrowhead strokes that have been misclassified as text can be identified.

Firstly, the arrowhead is found by looking at the curvature of the stroke. If the stroke is in two fragments (using the Number of Fragments feature described in Table 7), or has three polyline cusps (based on the Number of Polyline Cusps feature described in Table 7), then it is considered as a possible arrowhead. For example, the three red dots on the arrowhead in Figure 63 denote the polyline cusps and also show how the arrowhead is broken into two fragments. We can therefore consider this to be an arrowhead.

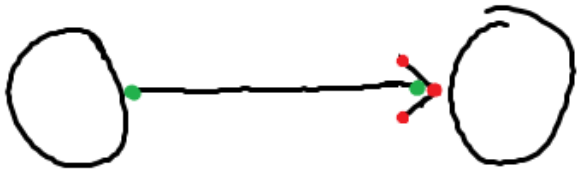


Figure 63 Is Arrowhead Example.

Next, the arrow shaft is found with a search for the closest shape stroke to the possible arrowhead. In Figure 63 the closest shape stroke is the arrow shaft. The shaft must satisfy a line test, where the ratio of the distance between the first and last point of a stroke and the cumulative length of the stroke is under a certain threshold. For the example in Figure 63, the distance between the two green end points of the shaft is calculated and divided by the cumulative length of the stroke. Lastly, the ratio of the length of arrowhead to the whole arrow length must be less than 40%. The length of the arrowhead in Figure 63 is

calculated and compared to the total length of the shaft and arrowhead. This is less than 40%: therefore this arrowhead is re-classified as a shape stroke.

The second parse can be run after the use of any classifier model to refine the results.

6.5.2 Results

The extended feature set, with additional features 1-3 described above, were used to train the seven classifiers; note that feature 4 is not included as this is used in a second parse of classification. The training dataset was generated again to include these new features. The optimal parameters shown in Table 26 were used for each classifier. The results of ten-fold cross validation are shown in Table 31.

In addition, a LADTree at 1500 iterations was trained, as this produced a higher result for our original feature set (as shown in Table 23). The results of ten-fold cross validation for this LADTree are also shown in Table 31.

Paired t-tests show that the LADTree with 1500 iterations is significantly more accurate than all other models. Paired t-tests also show that there is no significant difference among models with the same parameters when trained with the second round feature set and the original feature set. (The results for the original feature set are included in Table 31.) For example, LADTree with 1500 iterations correctly classifies 97.76% of strokes on average (over ten folds of cross validation) in the training dataset when using the extended feature set; the classification rate with the original feature set is 97.48%. The difference between these results is not statistically significant. This is the same for all classifiers when compared with the original feature set results. This indicates that the additional features do not have a significant influence on classification.

To test feature 4, the second parse feature, a different approach was required. The purpose of this feature is to correct misclassifications made in the initial classification results. A pre-trained classifier and a new dataset, independent of the training dataset, are required here. This classifier is used to classify the new dataset and then the results of this initial classification are used by feature 4 to correct any misclassifications that may have been made – thus running a second parse of classification. The ER/Process diagram dataset, described in Section 5.5.2, was used as the test dataset and the LADTree with 1500 iterations was chosen as the model, given that it was found to be significantly more accurate than all other models with the second round feature set.

Classifier	Average % Correct (sd)	
	Extended feature set	Original feature set
LADTree (1500 iterations)	97.76 (0.45)	97.48 (0.52)
LADTree	97.31 (0.67)	97.12 (0.54)
SMO	96.56 (0.54)	96.41 (0.73)
RandomForest	96.44 (0.61)	96.45 (0.57)
LogitBoost	96.39 (0.76)	96.67 (0.48)
Bagging	95.64 (0.58)	95.67 (0.58)
Multilayerperceptron	95.12 (0.80)	95.02 (0.78)
LMT	94.96 (0.35)	94.85 (0.68)

Table 31 Results of using the Second Round Feature Set.

Four conditions are included in the test to highlight the effect of using the extended feature set and second parse feature. The conditions are combinations of using the original and extended feature sets with and without feature 4. The results of classifying the ER/Process diagram dataset under these conditions are shown in Table 32.

Z-tests were conducted between the highest and the lowest results in each shape class. Paired t-tests were not suitable for use here as we were comparing single values rather than paired groups (resulting from 10-fold cross validation) as before. Z-tests can be used to test the difference between two proportions when the sample size is large (LeBlanc 2004). The tests show that there is no significant difference between the highest and lowest results for arrows (p-value: 0.08, sd: 0.026), rectangles (p-value: 0.5042, sd: 0.0052), diamonds (p-value: 0.5354, sd: 0.0052), ellipses (p-value: 0.415, sd: 0.0036), lines (p-value: 0.1091, sd: 0.013) and shapes overall (p-value: 0.1928, sd: 0.0065). As there is no significant difference among these values, we can infer that there is no significant difference in any of the results for each shape class.

	% Arrow	% Rectangle	% Diamond	% Ellipse	% Lines	% Total Shape	% Total Text	% Total Correct
Extended feature set	59.26	98.59	99.02	99.71	85.82	88.86	98.42	95.38
Extended feature set + feature 4	62.96	98.59	99.02	99.71	86.72	89.71	96.70	94.48
Original feature set	58.40	98.24	99.34	99.42	87.16	89.04	98.36	95.40
Original feature set + feature 4	60.97	98.24	99.34	99.42	87.91	89.67	96.66	94.44

Table 32 Results of Second Round Features to the Original Feature Set on ER/Process Diagrams Dataset with the LADTree (1500 Iterations)

For text, z-tests consistently show that the conditions without feature 4 are significantly more accurate than those with feature 4. In addition, the conditions without feature 4 are not significantly different from each

other and the conditions with feature 4 are also not significantly different from each other. The p-values for these tests are shown in Table 33.

A similar pattern is evident in the results for the total percentage of strokes classified correctly, as shown by the p-values in Table 34. This parallel may be because 68% of the total ER/Process diagrams dataset are text strokes, as shown in Table 18.

The results indicate that the extended feature set produces results similar to the original feature set. However, the addition of feature 4 often leads to higher rates of misclassification for text. Feature 4 was designed to detect arrows. The results in Table 32 show that a higher proportion of arrows are correctly classified when using feature 4, but as the z-test showed, these differences are not statistically significant. Only 5% of the ER/Process diagram dataset are arrows. Correct classification of arrows comes at the high cost of text misclassification since text makes up 68% of the dataset.

In general, these results suggest that the original feature set covers the problem of distinguishing text and shapes well. Additions to this set, although they do not hinder the results, may not provide a significant difference in classification. A second parse of results, however, can come at a cost of higher misclassifications of other classes.

	Original	Extended	Original + feature 4	Extended + feature 4
Original		0.7407	**3.375×10^{-14}**	**1.057×10^{-13}**
Extended			**2.887×10^{-15}**	**9.326×10^{-15}**
Original + feature 4				0.8769
Extended + feature 4				

Table 33. P-values for the % of Text Correctly Classified under Four Conditions (sd: 0.0022)

	Original	Extended	Original + feature 4	Extended + feature 4
Original		0.9361	**0.0002**	**0.0004**
Extended			**0.0003**	**0.0006**
Original + feature 4				0.8832
Extended + feature 4				

Table 34 P-values for the Total % of Strokes Correctly Classified under Four Conditions (sd: 0.0026)

6.6 Computational Requirements

Our aim was to build a set of classifiers that, given some ink stroke data, would classify the strokes into the classes of shape or text. Weka was used to perform our data analysis and consequently build accurate text-shape dividers.

The computational requirements of this analysis proved to be demanding due to the complexity of some algorithms used and the large number of features and instances included in the training dataset. During the preliminary investigation and tuning steps, a standard desktop Dell Optiplex 745 Intel® Core™2 CPU 6400 @ 2.13GHz, 3 GB RAM running Microsoft Windows XP was used. Many experiments took several days or even weeks to complete using this PC, so three additional servers were used: one running Linux with 2 Intel(R) Core(TM)2 Duo CPU E6750 @ 2.66GHz and 3 GB RAM and two running Windows Server 2008 with 5 GB RAM each. Even with these resources, in some cases one fold of a ten-fold cross validation experiment took several days to complete.

Finally, the experiments became so computationally expensive that the Auckland Cluster was set up to run the analysis. The Auckland Cluster has 20 nodes, where 10 nodes have 16 GB RAM and 10 nodes have 64 GB RAM; all nodes have two quad core CPUs. The number of cores in use depends on the number of jobs submitted and, as there are many users of this cluster, jobs are queued until resources become available. In order to use the resources of the cluster, the experiments were distributed using the advanced functions of Weka Experimenter. Experiments were distributed by fold. For example for a ten-fold cross validation trial, each fold was run in parallel by separate hosts on the cluster. Parallelising the folds of each experiment using the resources of the cluster greatly decreased the time required for the analysis and allowed us to run more complex algorithm configurations than before. Computation time using the cluster took several days for one run rather than one fold as before.

Use of the Auckland Cluster required some additional configuration steps. Weka was first installed on the cluster. Weka experiments were configured on the local PC with the required parameters and saved to a configuration file. The advanced functions of Weka allow an experiment to be configured so that each fold in an experiment is distributed to different hosts.

Interaction with the cluster was via a command line interface. To run the experiments, two scripts were run by that interface. The first script, shown in Figure 64, starts a remote engine for a worker host; a worker

host is where the work for one fold is carried out. If there are ten folds for an experiment, there must be ten worker hosts; one for each fold. The second script, shown in Figure 65, runs the experiment using the Weka configuration file in Weka Experimenter. This is the main host that passes on tasks to the worker hosts by connecting to each remote engine using Java Remote Method Invocation (RMI).

```
#!/bin/bash
export DUMMY_NODE_NAME=$1
export HOME_DIR=/home

# replace the node name
cd $HOME_DIR/weka
export HOSTNAME=$(hostname -s)
java -cp .:weka.jar InsertHostName $DUMMY_NODE_NAME $HOSTNAME
$HOME_DIR/experiment_configuration_file.exp

# start worker thread
cd $HOME_DIR/remote_engine
bash startRemoteEngine
```

Figure 64 Worker Script to Insert the Correct Host Names and Start the Remote Engine.

```
#!/bin/bash
export HOME_DIR=/home

# start main process
cd $HOME_DIR
java -Xmx1g -cp $HOME_DIR/jars/mysql.jar:remote_engine/remoteEngine.jar:weka/weka.jar
-Djava.rmi.server.codebase=file:$HOME_DIR/weka/weka.jar weka.experiment.Experiment -r
-l $HOME_DIR/experiment_configuration_file.exp
```

Figure 65 Main Script to Start the Experiment.

The experiment configuration file specifies a list of worker host names that is used by the main host for establishing remote connections. However, when using the cluster, hosts are allocated dynamically. Therefore, it is impossible to specify the hosts before an experiment is run. To solve this problem, a small Java application, called InsertHostName, was written to run on the cluster and replace “dummy” host names in the configuration file with the correct host names. This application was run within the first script before starting the remote engine. Java was used, as the configuration file had to be de-serialised, edited (to include the correct host names) and serialised again using Weka code libraries; these code libraries are written in Java.

For an experiment with ten folds, the first script is run ten times and then the main script is run to start the experiment. The results of each fold are written to a database.

Weka was an invaluable tool for performing the data analysis. It has a large range of data mining techniques that are easy to tune, train, test and extract the resulting models. In addition, the ability to distribute each fold of ten-fold cross validation experiments to remote machines made the process much more efficient. Ideally we would have preferred to set up our experiments with ten runs of ten-fold cross validation: this is a standard approach to ensure that the results are accurate and reliable (Witten et al. 2005). However, this was not feasible due to the computational time required. Even with the use of a computational cluster, one run with each fold running in parallel took several days to complete. If we were to run each fold in parallel, this set up would require 101 jobs to be running on the cluster simultaneously, 100 worker jobs (running the worker script in Figure 64) for 10 runs of 10-fold cross validation and one additional job to run the main script (shown in Figure 65) which starts the experiment. Due to high usage of this cluster by many other users, we were only able to run no more than 20 jobs simultaneously, making this set up unfeasible.

Overall, excellent results were obtained using the computational resources available. In addition, the use of large and varied datasets with 10-fold cross validation ensures that the levels of accuracy found were reliable and the algorithms were well trained.

6.7 Summary

Our results showed that the most accurate classifiers produced from our experiments use the LADTree and LogitBoost machine learning algorithms. The use of feature selection and ensembles were investigated to try to improve on these results. However, the original tuned classifiers perform on the same level or better than those using these additional techniques. Additional features were also implemented to identify connectors, as these have the highest rate of misclassified strokes. The results using these features are ~97% for LADTree, but these are not significantly different from the original tuned results.

Based on this analysis, we chose to implement and evaluate the original tuned LADTree and LogitBoost classifiers as the basis for new text-shape dividers for sketch recognition applications. The top two ensembles, LLSL and LLS, were also chosen along with the second round LADTree (1500 iterations) with feature 4 to further evaluate their performance on independent test sets. Models produced by feature

selection techniques are not included in the evaluation. Although some models with feature selection were not significantly different from the original tuned models, they did not produce high levels of accuracy compared with the ensembles and second round models. The following chapter presents the results of our evaluation of these text-shape dividers.

Chapter 7

Evaluation

The goal of our research is to improve recognition of hand-drawn diagrams through the development of more accurate recognisers using data mining. Our focus has been on developing text-shape dividers. A systematic investigation of data mining techniques, using a comprehensive ink feature library, has resulted in the construction of several dividers. To evaluate whether or not these new dividers are an improvement over existing divider techniques, we have run several experiments to compare their accuracy to three existing text-shape dividers.

In order to run a comparative evaluation of our new dividers against existing ones, we integrated all of the existing and proposed dividers into DataManager's Evaluator (Schmieder 2009; Schmieder et al. 2009). The Evaluator is a part of Schmieder's Master's project and is an extension of the work presented on DataManager in Chapter 5. The Evaluator is a platform designed specifically for comparative evaluations of different sketch recognition algorithms. Algorithms are easily integrated into the platform and compared to each other by testing their performance using the same datasets and testing parameters. Evaluating algorithms with the same functionality under the same environment ensures that fair comparisons are made.

This chapter begins by describing the implementation and integration of each of the dividers included in the evaluation. Fresh data was used for testing these dividers; each dataset is outlined in Section 7.2. The evaluation results of each divider tested against each dataset are reported in Section 7.3. Following this is a section that describes an investigation into the use of domain-specific dividers for selected diagram types. The chapter concludes with a summary of the evaluation results.

7.1 Divider Implementation

To run the evaluation within DataManager's Evaluator (Schmieder 2009; Schmieder et al. 2009) the dividers were implemented and integrated into the platform. In addition to the new divider models, three existing dividers were integrated: Divider 2007 (from our previous work) (Patel et al. 2007), the Microsoft Ink Analysis divider (Microsoft Corporation 2008), and the Entropy divider (Bhat et al. 2009). The details of their implementation are described below.

Integrating each divider into the Evaluator is a simple task. A manager class must be written for each recogniser. This class receives data from the evaluation platform, classifies the data (as text or shape strokes in this case) and outputs the resulting classifications. In addition, a settings file must be edited to include the new recogniser into the evaluation platform and specify any particular parameters unique to each recogniser. The Evaluator is written in C# .NET, but is designed so that recognisers written in other programming languages, such as Java, can be integrated easily.

The Evaluator requires labels to be applied to each stroke, which will be used as the ground truth information for generating statistics after classification. There are options that can be specified for each evaluation. Under stroke options, single and/or multi stroke shapes can be selected for evaluation. For this evaluation, both single and multi stroke shapes are chosen, so all the data is classified. Under shape options, diagrams can be classified shape by shape or as a whole diagram. The dividers classify on a stroke basis but use information about the whole diagram for spatial and temporal context features. Therefore the whole diagram option is selected here.

7.1.1 Entropy

The Entropy divider (Bhat et al. 2009) uses a measure of stroke density to distinguish between text and shapes; a detailed description is found in Section 2.1. We have re-implemented this divider in C# .NET which makes it simple to integrate into the Evaluator.

When implementing the Entropy divider, it had to be trained as no thresholds were provided by Bhat et al (2009). It was trained on the same training dataset as the new dividers; this data is described in Section 5.5.2. The Decision Stump algorithm from Weka using ten-fold cross validation was chosen to find optimal thresholds. This algorithm was chosen as it generates a decision tree with one node, essentially producing one decision based on the Entropy feature. The ten-fold cross validation results report that

85.76% of the training data are correctly classified; other algorithms such as OneR (Holte 1993), a rule based method, and a J48 tree (C4.5 decision tree) (Quinlan 1993) gave similar results.

Bhat et al's (2009) original implementation includes grouping strokes using spatial and temporal proximity as a pre-processing step before classification. In this implementation, strokes are not grouped as there are many cases where their proposal could fail. For example, in Figure 66 the left hand stroke of the rectangle would be grouped with the word, even with the use of temporal context if the strokes in the symbols were interspersed. Sezgin et al (2007) describe interspersed drawing as when a new object is drawn before another object has been finished. They have found that this is very common in naturally drawn diagrams.

Figure 66 Example of where Stroke Grouping could Fail

7.1.2 Divider 2007

Our divider developed from previous work (Patel 2007) was not re-trained; it was implemented with the same thresholds as the original decision tree shown in Figure 67. A detailed description of how this divider was developed can be found in Section 2.1. The implementation is in C# .NET so the integration into the Evaluator was straightforward.

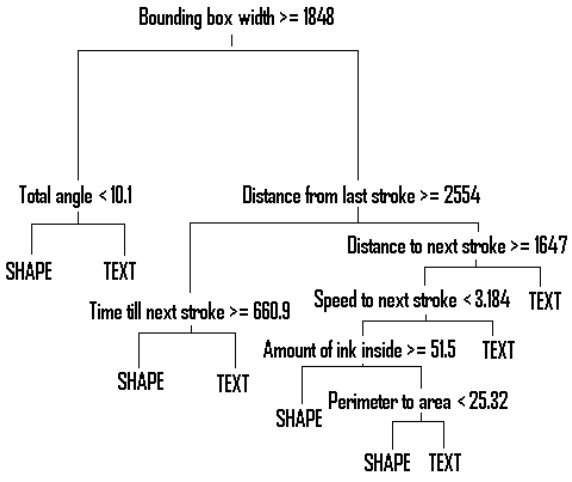


Figure 67. Decision Tree Divider Produced by Patel et al (2007)

7.1.3 Microsoft Divider

Microsoft Ink Analysis (Microsoft Corporation 2008) is able to distinguish between shapes and handwriting; more details are in Section 2.1. This divider is easily implemented using the .NET Microsoft.Ink and Microsoft.Ink.Analysis libraries. C# .NET is used here to ensure simple integration into DataManager's Evaluator.

7.1.4 New Dividers

Many new text-shape dividers were built during our systematic investigation of features and algorithms in Chapter 6. Five particular dividers were selected for the evaluation; they are listed in Table 35 with simplified names for easy reference. LADTree 1 and LogitBoost are the top performing classifiers resulting from the original tuning step of the algorithm parameters in Section 6.2. Vote 1 and Vote 2 are the top classifiers from the investigation of ensembles; see Section 6.4 for more details. LADTree 2 is the best performing classifier with the second round features and second parsing; more details for this classifier can be found in Section 6.5.

Divider	Configuration
LADTree 1	LADTree with 1500 iterations.
LogitBoost	LogitBoost with DecisionStump, shrinkage = 0.1 and 5000 iterations.
Vote 1	Ensemble classifier LLSL; refer to Table 29 for more details.
Vote 2	Ensemble classifier LLS; refer to Table 29 for more details.
LADTree 2	LADTree with 1500 trained with 2nd round features including second parse

Table 35 New Dividers chosen for the Evaluation

Each divider was generated and output into .model files using Weka's Explorer interface. These model files contain all the information necessary for a recogniser to classify a given stroke. We integrated these models into the Evaluator by reading in the .model files and use this information for classification.

DataManager uses the C# .NET Framework, whereas Weka uses Java. However, Weka is open source so integrating Weka libraries into DataManager was done with ease. The IKVM (2009), an implementation of Java for .NET, was used to connect the two and thus allowed us to import Weka models into DataManager's Evaluator. An IKVM DLL and Weka DLL were added to the project to facilitate this step. Documentation is available on this process of connecting Weka to .NET projects[14].

[14] http://weka.wikispaces.com/Use+WEKA+with+the+Microsoft+.NET+Framework

Once the Weka models were read, they were used to classify ink data into text or shape strokes. To classify a stroke, the appropriate feature vector is calculated, based on the feature library presented in Chapter 4, and passed to the classifier which returns its prediction. To summarise, there are three basic steps to integrating these dividers into DataManager's Evaluator:

1. Obtain the trained classifier by reading in the .model file.
2. Calculate a feature vector for the stroke that must be classified as text or shape.
3. Pass the feature vector to the classifier obtained to predict the class of the stroke.

Steps 1 and 3 were implemented using the existing Weka API (Witten et al. 2005). Step 2 was implemented using DataManager's feature library. This was a straightforward procedure which re-used methods already implemented in DataManager's Dataset Generator code.

7.2 Test Data

Several datasets were gathered for testing the set of text-shape dividers introduced above. The datasets represent diagrams from various domains. These datasets were intentionally chosen as they represent a large range of diagram types with different characteristics and relationships between text and shapes that provide a challenge to the dividers. Datasets have also been added to stretch the dividers' capabilities beyond simple diagrams such as to-do lists (which are documents rather than diagrams) and Euler and logic diagrams which have unique content and layout.

We collected some datasets using DataManager and others were collected by different research groups using their own data collection methods. The advantage of including data collected by others is to test our text-shape dividers for any bias relating to our method of data collection. One of the problems with using data sourced from elsewhere is that we must convert the data formats to be compatible with DataManager. Unfortunately, there is no standard sketch data format as yet; every research group uses their own schema and sometimes even multiple schema within the same group.

A summary of each dataset used for testing the text-shape dividers is shown in Table 36 with full descriptions in the following sections.

	# Participants	# Text Strokes	# Shape Strokes	Total # Strokes	% Text : % Shape Strokes	Origin
Mind-map	20	1815	364	2179	83:17	Our own
To-do list	20	1710	201	1911	89:11	Our own
UML Class diagram	20	1481	383	1864	79:21	Our own
COA	6	516	214	730	71:29	(Bhat et al. 2009)
Logic	13	2296	6320	8616	27:73	(Alvarado et al. 2007)
Euler	10	413	334	747	55:45	(Delaney et al. 2010)
Total		8230	7817	16047	51:49	

Table 36 Summary Statistics for each Test Dataset

7.2.1 Mind-map, To-do List and UML Class Diagrams

Three types of diagrams were collected using DataManager as test data: mind-maps, to-do lists and UML class diagrams. These diagrams were collected from 20 participants; each drawing three diagrams, one for each diagram type. The participants were given written instructions, shown in Table 37, to follow when drawing their diagrams. A pilot test of the instructions was conducted to ensure that participants could easily complete the tasks.

Mind-maps were chosen for testing the text-shape dividers as they provide a challenge for these algorithms. The text in mind-maps is not always on a horizontal baseline or contained within shapes and there are lots of connectors; see Figure 68 for an example of a mind-map drawn by a participant. The results of the pilot test of instructions for drawing a mind-map showed that participants were unsure how to construct the diagram. To assist participants an example mind-map, shown in Figure 69, was provided to give them an idea of the expected style of mind-map. This example is different to the mind-map that participants were asked to draw, to prevent them from copying. In addition, the list of words to include in the mind-map was shortened from 15 to 10 items to reduce the drawing time.

UML class diagrams are part of the test set as they include text contained by shapes as well as text outside shapes, such as the specification of relationships between classes; see Figure 70 for an example of a UML class diagram. The pilot test of instructions for UML class diagrams showed that participants needed reminding of the syntax for such diagrams. A dictionary sheet showing participants each component of a UML class diagram was used to remind them of the syntax of this particular domain; the dictionary sheet is shown in Figure 71. As UML class diagrams are generally only known by computer scientists and software engineers, the data collection has been targeted to participants with this background.

Diagrams	Instructions
Mind map	Draw a mind map to classify the following list into categories of your choice. The central idea is Food. Apples Grapes Bread Hot chocolate Chicken Lamb Chocolate Pizza Coffee Tomato
UML Class diagram	There are four classes listed below with their associated attributes: Student (name, id, enrolment status) Course (name, code, semester) Lecturer (name, id) Person (name, id) Draw at least 3 possible relationships between classes e.g. a student may be enrolled in 0 to 4 courses.
To-do list	a) It is Monday morning and Jane has many things to do this week. For university, she needs to finish her maths 101 assignment by Wednesday and start her computer science 101 assignment that is due next week. She has to take her dog to the vet on Thursday and pick her Mum up from the airport on Tuesday – which also means she should make sure she cleans the house before Mum gets home! On top of all this, she is doing some extra shifts at work on Wednesday from 4pm - 9pm and Friday from 6pm – 10pm. Help Jane get organised by writing her a To-do list. You do not have to copy the task descriptions word for word, just summarise them using 1-3 words each. b) Imagine it is now Thursday and Jane has completed all the tasks she had to do before Thursday. Update the list you have just created to show that these tasks have been completed.

Table 37 Instructions to Participants for Drawing Diagrams for Test Data

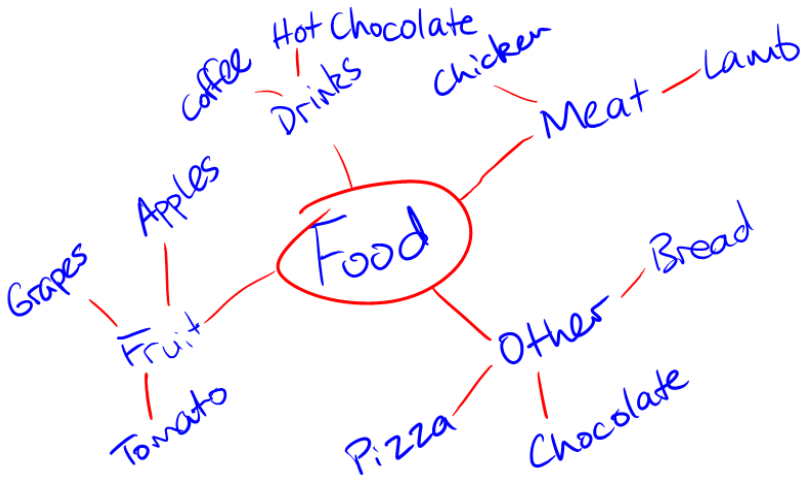


Figure 68 Example of a Mind-map Drawn by Participants. Shape strokes are shown in red and text strokes in blue

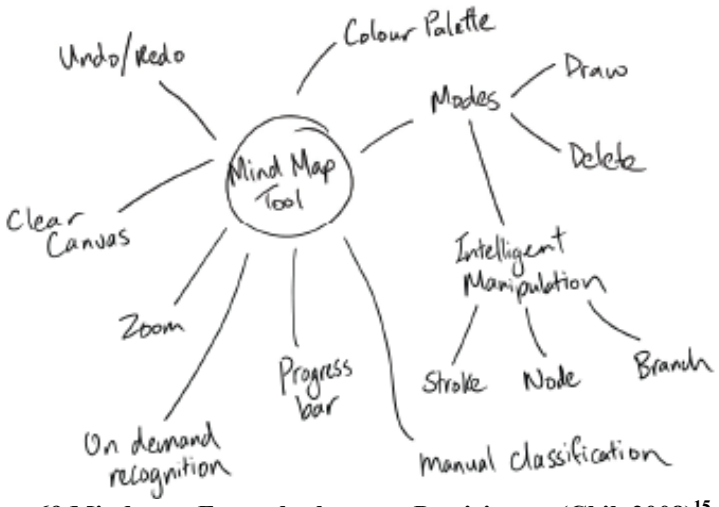


Figure 69 Mind-map Example shown to Participants (Chik 2008)[15].

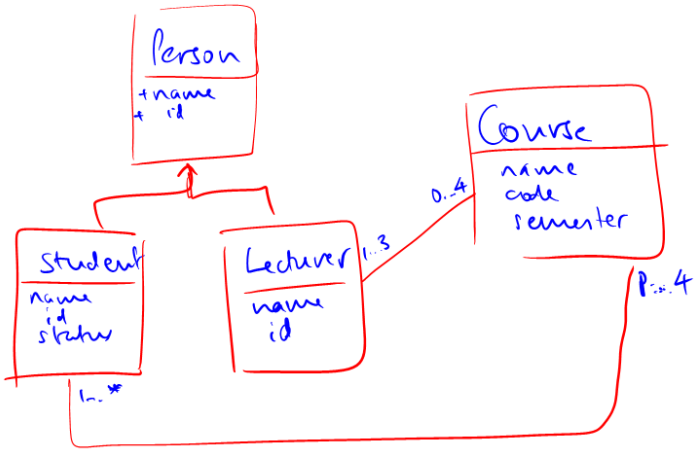


Figure 70 Example UML Class Diagram drawn by a Participant. Shape strokes are coloured in red and text strokes in blue.

The to-do list is a document rather than a diagram, where there is a large amount of text commonly written in lines as well as symbols such as bullet points, ticks and crosses; see Figure 72 for an example of a to-do list. This was used for testing to gauge the dividers' performance on such documents. When piloting the to-do list instructions, it became clear that participants were copying the task descriptions word for word from the instructions and needed help understanding the second part of the task. The instructions were re-written to make it clear that they could summarise the tasks when adding them to the to-do list and clarify the second part of the task.

The number of strokes in each dataset is shown in Table 36 including a breakdown of the number of text and shape strokes.

[15]Image obtained from Chik, V. C. H. (2008). Intelligent Mind-mapping. Computer Science. Auckland, University of Auckland. **MEng:** 119. Reproduced with permission from the author.

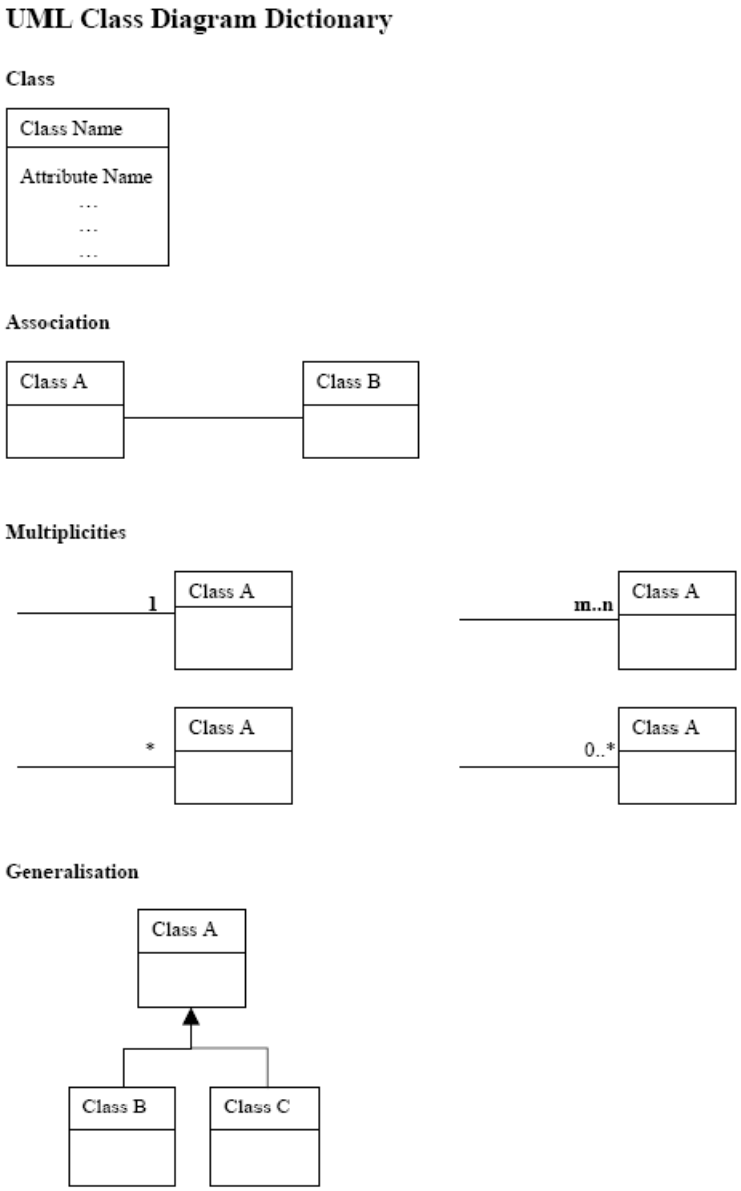


Figure 71 UML Class Diagram Dictionary shown to Participants. Based on (Fowler et al. 1999)

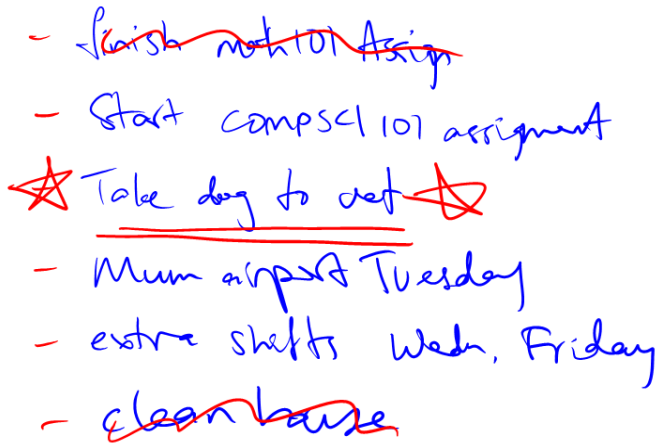
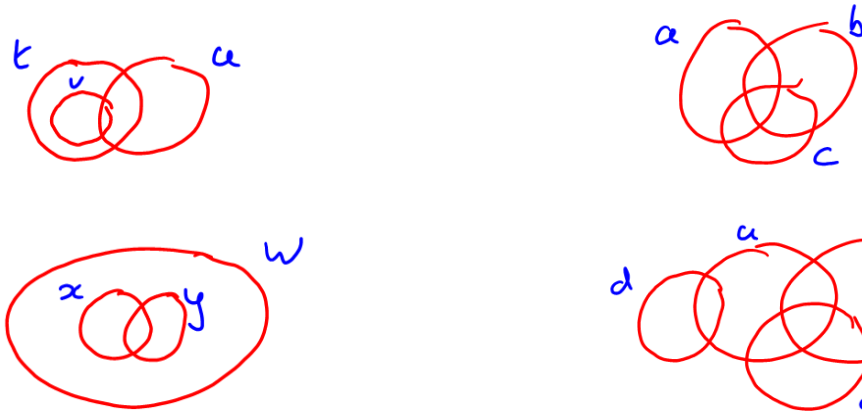


Figure 72 Example To-do List drawn by a Participant. Shape strokes are coloured in red and text strokes in blue.

7.2.2 Euler Diagrams

A set of Euler diagrams was collected by Delaney et al (2010) using DataManager for their own study. The dataset is re-used here for testing text-shape dividers as Euler diagrams exhibit interesting spatial relationships between shapes such as intersections and containment not seen in any other datasets. In addition, the text for this particular dataset is single letter labels that may provide more of a challenge to the dividers. The data comes from ten participants where each one drew five Euler diagrams, ranging from simple to complex tasks. The breakdown of text to shape strokes and the total number of strokes in this dataset are shown in Table 36. Examples of the Euler diagrams drawn for each task are shown in Figure 73.

a) Simple Euler **b) Euler 2** **c) Euler 3**



d) Complex Euler **e) Final Task**

Figure 73 Euler Diagram Examples. Shape strokes are shown in red and text strokes in blue.

7.2.3 Military Course of Action Symbols (COA)

A dataset of military course of actions symbols was used to test the text-shape dividers. This set was collected by Bhat et al (2009) as a training set for their Entropy based text-shape divider. It was obtained directly from the researchers. It is composed of military course of action (COA) symbols collected using SOUSA (Paulson et al. 2008c; Kaster et al. 2009). Six participants drew two examples of 16 COA symbols each; some data was removed by Bhat et al (2009) as they were drawn with a mouse as opposed to a digital pen. Examples of COA symbols from this dataset are shown in Figure 74. Overall, this dataset is very simple; it is composed of isolated symbols, not full diagrams, and most symbols include a rectangle and text.

This dataset was chosen for testing the text-shape dividers as it provides an interesting comparison of our new dividers against Bhat et al's Entropy divider (2009). As SOUSA uses a different data format to DataManager, the data had to be converted to DataManager's format. The number of text and shape strokes in the COA dataset is shown in Table 36.

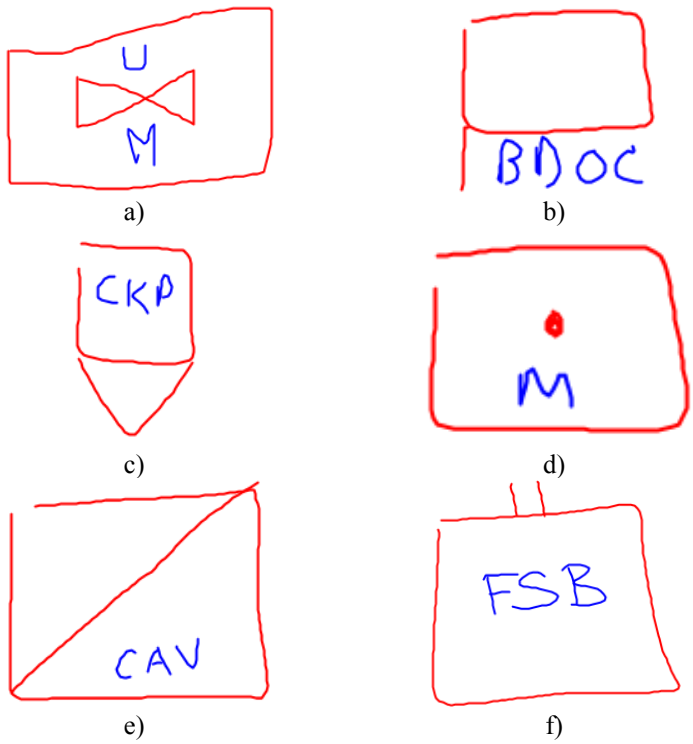


Figure 74 Examples of COA Symbols. Shapes strokes are shown in red and text strokes in blue.

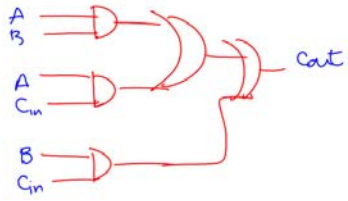
7.2.4 Logic Diagrams

A dataset of logic diagrams is also part of the test set for evaluating the text-shape dividers. This dataset was collected by Alvarado et al (2007) and can be downloaded from their webpage[16]. It is composed of logic diagrams collected from 13 students in an Electronics and Computer Architechture course at Harvey Mudd College. Students used Microsoft Journal to draw circuit diagrams for their assignments, class notes and lab reports. These diagrams were later collated into a dataset containing 98 sketches. The diagrams range from simple to complex; this is illustrated in Figure 75.

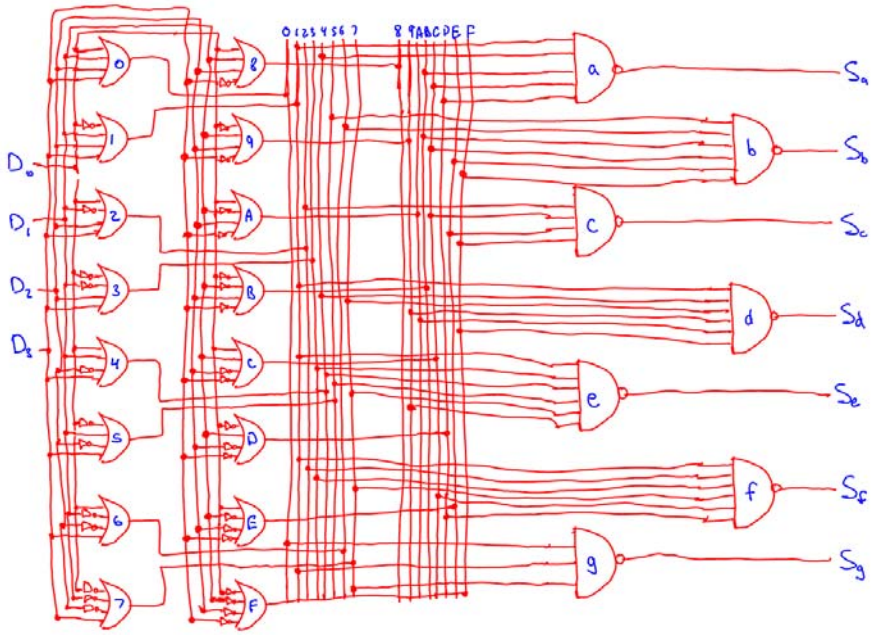
This dataset differs from all the others used in the test set as they were drawn in real world situations. This property makes it a very good choice for testing how well the dividers perform on real world data. The characteristics of logic diagrams are also different from other datasets; there can be many intersecting lines and shapes such as semi-circles that are not found elsewhere.

The dataset was obtained in the MIT XML Format and therefore had to be converted to DataManager's format. The number of text and shape strokes in the logic dataset is shown in Table 36.

a) Simple 1



b) Simple 2



c) Complex

Figure 75 Examples of Logic Diagrams. Shape strokes are shown in red and text strokes in blue.

[16] http://www.cs.hmc.edu/~alvarado/research/download.html

7.3 Results

The goal of this evaluation is to determine whether the new dividers are more accurate than existing dividers. Using DataManager's Evaluator, each divider has been used to classify the datasets described in the previous section, into text and shape strokes. All evaluations were performed on a Hewlett Packard Tablet PC Intel® Core™2 Duo CPU @ 1.86GHz, 2 GB RAM running Microsoft Windows 7. The results of these tests are presented in this section, beginning with the accuracy of each divider overall followed by the results of the dividers on each independent dataset.

Table 38 shows the full classification results of each divider on each dataset. Analysis of these results requires a comparison of classification rates for multiple methods of division and multiple datasets, so a multiple comparison statistical procedure is used. Simpler statistics, such as an F-test, can perform a comparison between method A and all others, but this test can only conclude that method A is significantly different from at least one of the other methods or that there is no significant difference between method A and all others (Ott et al. 2000). It cannot identify exactly which methods are or are not significantly different. Paired t-tests can be used to narrow down where the significant differences lie but this can only be done using one pair at a time. Further, using this test may result in significant differences being found by chance where no significant differences really exist, due to widths of confidence intervals being underestimated.

Tukey's confidence intervals (Ott et al. 2000) are used here as a method of performing multiple comparisons. All comparisons between methods can be performed at the same time rather than pair-wise. Tukey's confidence intervals are wider than for paired t-tests, and this ensures that the significant differences found using this method are accurate.

A graph displaying Tukey's confidence intervals for each dataset and divider is shown in Figure 76. The R code for calculating these confidence intervals can be found in Appendix C. There is one line for each dataset and a confidence interval (vertical line) showing the performance of a particular divider on the dataset. The widths of the confidence intervals are influenced by the size of the dataset. Small datasets have wider confidence intervals than large datasets because there is not as much information available for a finer grained prediction of where the population mean sits. If the confidence intervals for two dividers or datasets do not overlap, then there is a significant difference between them.

	% Text Correct	% Shape Correct	Total % Correct
LogitBoost			
Mind-map	95.81	86.26	94.22
To-do	95.32	63.18	91.94
UML	96.02	97.39	96.30
Euler	72.88	96.71	83.53
COA	91.86	84.58	89.73
Logic	51.31	93.07	81.94
LADTree 1			
Mind-map	97.41	81.59	94.77
To-do	95.32	62.19	91.84
UML	96.15	97.65	96.46
Euler	82.32	91.32	86.35
COA	93.80	79.91	89.73
Logic	76.79	79.56	78.82
LADTree 2			
Mind-map	96.75	82.97	94.45
To-do	94.74	64.18	91.52
UML	95.68	99.22	96.41
Euler	72.88	91.92	81.39
COA	95.54	78.97	90.68
Logic	76.87	76.16	76.35
Vote 2			
Mind-map	97.80	81.04	95.00
To-do	95.32	59.70	91.58
UML	97.43	97.91	97.53
Euler	83.78	92.81	87.82
COA	98.45	75.23	91.64
Logic	93.25	53.10	63.80
Microsoft			
Mind-map	99.67	63.46	93.62
To-do	99.94	34.83	93.09
UML	98.92	90.34	97.16
Euler	77.72	69.76	74.16
COA	98.64	40.19	81.51
Logic	80.75	67.23	70.83
Vote 1			
Mind-map	97.69	81.87	95.04
To-do	95.44	60.70	91.78
UML	97.37	98.69	97.64
Euler	83.29	92.51	87.42
COA	96.90	83.64	93.01
Logic	97.08	33.37	50.35
Entropy			
Mind-map	97.52	54.12	90.27
To-do	97.25	38.81	91.10
UML	98.72	60.57	90.88
Euler	92.98	93.41	93.17
COA	94.96	80.37	90.68
Logic	98.43	22.93	43.05
Divider 2007			
Mind-map	94.21	72.25	90.55
To-do	94.21	71.14	91.78
UML	86.09	85.12	85.89
Euler	43.58	92.81	65.60
COA	93.99	81.78	90.41
Logic	90.51	33.86	48.96

Table 38 Classification Rates for All Dividers on each Dataset

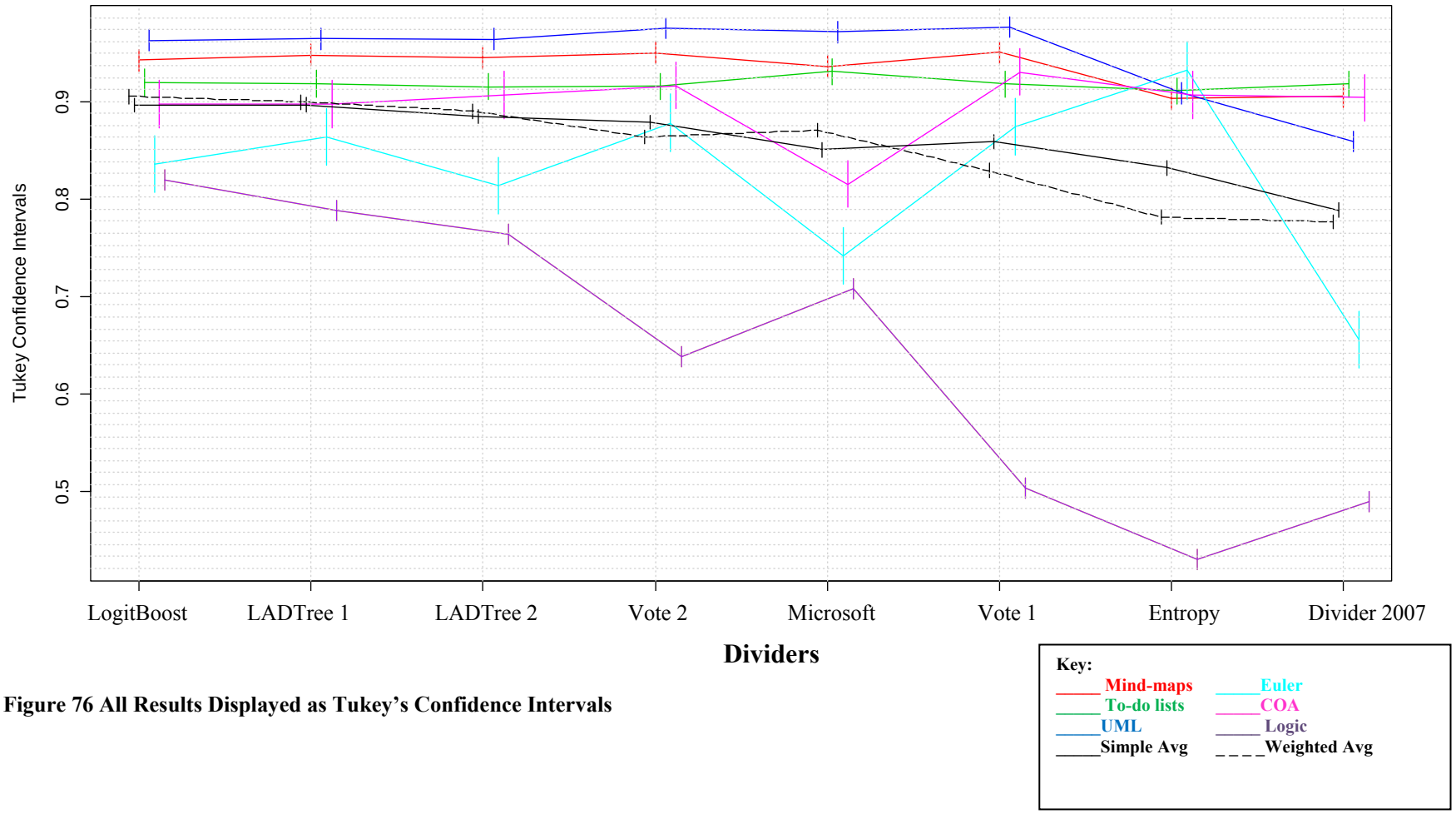


Figure 76 All Results Displayed as Tukey's Confidence Intervals

In terms of the dividers performance on the datasets overall, the dividers excel at classifying the UML class diagram dataset, with the exception of Divider 2007, and have difficulty dividing the logic diagram dataset. Tukey's confidence intervals in Figure 76 show that LogitBoost, Microsoft and Vote 1 dividers are significantly more accurate on the UML dataset than all other datasets. LADTree 1, LADTree 2 and Vote 2 are significantly more accurate when dividing the UML and mind-map datasets. Entropy is unique in that there is no significant difference in its accuracy on all datasets except for the logic diagram dataset; Entropy is significantly less accurate on the logic diagram dataset than all other datasets. Divider 2007 is significantly more accurate at dividing the COA symbol, to-do list, and mind-map datasets. All dividers are significantly less accurate on the logic diagram dataset than other datasets. LogitBoost and the Microsoft divider's accuracy on Euler diagrams are not significantly different to their performance on logic diagrams. Further examination of the results on these datasets appear later in this section.

The overall accuracy of each divider is calculated in two ways: using a simple average and a weighted average. They are shown in Table 39. These averages were used rather than calculating the straight percentage of strokes in all datasets that were correctly classified as the size of the datasets would cause a bias in the overall results. For example, dividers performing well on the logic diagram dataset, the largest dataset of all, would have an inflated classification rate.

The simple average for each divider is calculated using the formula in Equation 5. All datasets are given the same weight using this average, regardless of their size.

Divider	Simple Average	Weighted Average
LogitBoost	89.61	90.50
LADTree 1	89.66	89.93
LADTree 2	88.47	89.01
Vote 2	87.89	86.37
Microsoft	85.06	87.07
Vote 1	85.87	82.93
Entropy	83.19	78.16
Divider 2007	78.86	77.67

Table 39 Simple and Weighted Averages of Classification Rates for each Divider

$$\text{simple average} = \sum_{i=1}^{m} p_i / m$$

where: p is the proportion of strokes correct
i denotes the dataset
m is the number of datasets (there are six datasets included in the evaluation)

Equation 5 Simple Average Calculation for a Divider's Accuracy

The weighted average, on the other hand, takes into account the size of the dataset by calculating weights for each proportion of strokes correct for each dataset. The weighted average is calculated using the formula in Equation 6. The weight for a dataset should be proportional to the inverse variance of the mean (Snedecor et al. 1989). The variance is s^2/n: therefore the inverse variance is used as the numerator of the weight equation shown in Equation 6. The inverse variance is then divided by the sum of all inverse variances of the divider for each dataset. The weighted average for a divider is calculated as the sum of the proportion of strokes correct multiplied by the weight value for each dataset.

$$\text{weight}_i = \frac{n_i/s_i^2}{\sum_{i=1}^{m} (n_i/s_i^2)}$$

$$\text{weighted average} = \sum_{i=1}^{m} (p_i * \text{weight}_i)$$

where: i denotes the dataset
n is number of instances
m is the number of datasets (there are six datasets included in the evaluation)
s is the standard deviation
p is the proportion of strokes correct

Equation 6 Weighted Average Calculation for a Divider's Accuracy

Using the weighted average, datasets with a large number of instances have a smaller variance (as the variance is s^2/n) and are therefore weighted higher according to Equation 6. The reasoning behind this is that results obtained from datasets with smaller variance are more valuable as they are closer to the mean.

Both averages are included here as there is no pre-defined way of presenting the overall accuracy of each divider given the information we have for this multiple comparison problem. Including both averages ensures that the best information possible is presented.

Tukey's confidence intervals for the simple and weighted averages of each divider are shown in Figure 76 to assist us in analysing the overall significant differences between dividers. Table 40 summarises the significant differences observed using these confidence intervals. Two symbols are shown in each cell; the first for the relationship according to the simple average and the second for the weighted average. The relationships in the table should be read by row. For example in the first row, Divider 2007 is significantly less accurate (symbolised by an 'X') than entropy by measure of the simple mean and is not significantly different (symbolised by a '-') to entropy according to the weighted mean.

The only dividers that are always significantly more accurate or not significantly different to all others are LogitBoost and LADTree 1, regardless of which average is observed. LogitBoost is significantly more accurate than all other models except for LADTree 1, where there is no significant difference and except for LADTree 2 where there is no significant difference according to the simple mean. LADTree 1 is also not significantly different to LADTree 2 (based on both averages).

	Divider 2007	Entropy	LADTree 1	LADTree 2	LogitBoost	Microsoft	Vote 1	Vote 2
Divider 2007		X -	X X	X X	X X	X X	X X	X X
Entropy	√ -		X X	X X	X X	X X	X X	X X
LADTree 1	√√	√√		-, -	-, -	√√	√√	√√
LADTree 2	√√	√√	-, -		- X	√√	√√	- √
LogitBoost	√√	√√	-, -	-√		√√	√√	√√
Microsoft	√√	√√	X X	X X	X X		- √	X -
Vote 1	√√	√√	X X	X X	X X	- X		X X
Vote 2	√√	√√	X X	- X	X X	√ -	√√	

Table 40 Summary of the Significant Differences shown by Tukey's Confidence Intervals in Figure 76
√ significantly more accurate, X significantly less accurate, - not significantly different

In terms of the new divider models' performance in comparison to the existing dividers, all the new models are significantly more accurate than Divider 2007 and Entropy, regardless of which average is observed. For the Microsoft divider, LogitBoost, LADTree 1 and LADTree 2 are significantly more accurate regardless of which average is observed, Vote 2 is significantly more accurate according to the simple average and not significantly different based on the weighted average, and Vote 1 is not significantly different according to the simple mean but is significantly worse based on the weighted mean.

Overall, according to the Tukey confidence intervals of simple and weighted means, LogitBoost and LADTree 1 are the most accurate divider models for the example datasets tested. Most importantly, these dividers are significantly more accurate than the three existing dividers tested. A general ranking of dividers based on the information in Table 40 is as follows:

Ranking

1. LogitBoost
2. LADTree 1
3. LADTree 2
4. Vote 2
5. Microsoft

6. Vote 1
7. Entropy
8. Divider 2007

Within each dataset there is variation on the above ranking. The remainder of this section describes the results of the dividers for each dataset in more detail.

7.3.1 Mind-maps

The results for each divider tested against the mind-map dataset are shown in Table 41. Tukey's confidence intervals representing these results are shown in Figure 77. The confidence intervals for the total percentage of strokes correct show that there is no significant difference between any of the dividers when classifying the mind-maps, where classification rates range from 93.62% to 95.04%, except for Entropy and Divider 2007. These last dividers are significantly less accurate than the other models, with correct classification rates of 90.27% and 90.55% respectively.

The overall ranking of dividers based on Tukey's confidence intervals for the mind-map dataset is shown below.

Ranking for mind-map dataset:

1. LogitBoost, LADTree 1, LADTree 2, Vote 1, Vote 2, Microsoft
2. Entropy, Divider 2007

The results for the classification of text show that the Microsoft divider classifies text significantly better than all other dividers with 99.67% of text strokes correctly classified. However, the Microsoft divider's performance on shape strokes is the opposite at 63.46%, making it significantly lower than all of the new dividers which correctly classify between 81.04% and 86.26% of shapes. This result from the Microsoft divider is highly biased as most strokes are classified as text. A similar bias appears for Entropy, where only 54.12% of shapes are correctly classified; the lowest result for shapes of all dividers. In fact the new dividers perform significantly better when classifying shapes than all existing dividers. There is no significant difference between the new dividers classification rate for shape strokes. In general, for all dividers, the classification of text is more accurate than the classification of shapes. Again, however, 83% of the mind-map dataset is composed of text strokes.

Table 42 provides further shape classification information, giving the performance of each divider on containers and connectors in mind-maps. Containers are any strokes that enclose writing strokes, such as the middle circle in mind-maps. Connectors are the lines used to link ideas in these diagrams. There are 87 containers and 277 connectors in the mind-map dataset. All dividers classify containers more accurately than connectors. LADTree 2 correctly classifies all containers. LogitBoost is the most accurate at classifying connectors with a correct classification rate of 82.31%. Entropy performs the worst on both shapes, classifying 88.51% of containers correctly and only 43.32% of connectors. LADTree 2 uses extra features for detecting arrows and connectors, but its performance on connectors is only slightly higher than LADTree 1. Overall, it is clear that the misclassifications in mind-maps are mainly due to connectors.

	Number of strokes correct			**% Correct**		
Existing Dividers	**Text**	**Shapes**	**Total**	**Text**	**Shapes**	**Total**
Entropy	1770	197	1967	97.52	54.12	90.27
Microsoft	1809	231	2040	99.67	63.46	93.62
Divider 2007	1710	263	1973	94.21	72.25	90.55
New Dividers						
LADTree 1	1768	297	2065	97.41	81.59	94.77
LogitBoost	1739	314	2053	95.81	86.26	94.22
Vote 1	1773	298	2071	97.69	81.87	95.04
Vote 2	1775	295	2070	97.80	81.04	95.00
LADTree 2	1756	302	2058	96.75	82.97	94.45
Total	1815	364	2179			

Table 41 Results of All Dividers on Mind-map Dataset

Existing Dividers	**% Containers Correct (87)**	**% Connectors Correct (277)**
Entropy	88.51	43.32
Microsoft	95.40	53.43
Divider 2007	93.10	65.70
New Dividers		
LADTree 1	98.85	76.17
LogitBoost	98.85	82.31
Vote 1	98.85	76.53
Vote 2	98.85	75.45
LADTree 2	100.00	77.62

Table 42 Classification Rates for each Shape Class in the Mind-map Dataset

An example of misclassifications made by each divider for one participant's mind-map is shown in Figure 78. This particular mind-map was chosen as it had one of the lowest average correct classification rates overall. These snapshots were generated by DataManager's Evaluator (Schmieder 2009; Schmieder et al. 2009). Misclassified strokes are coloured in red and correctly classified strokes are in blue. The text shown in green gives more detail for each misclassified stroke; each misclassified stroke is numbered corresponding to the text.

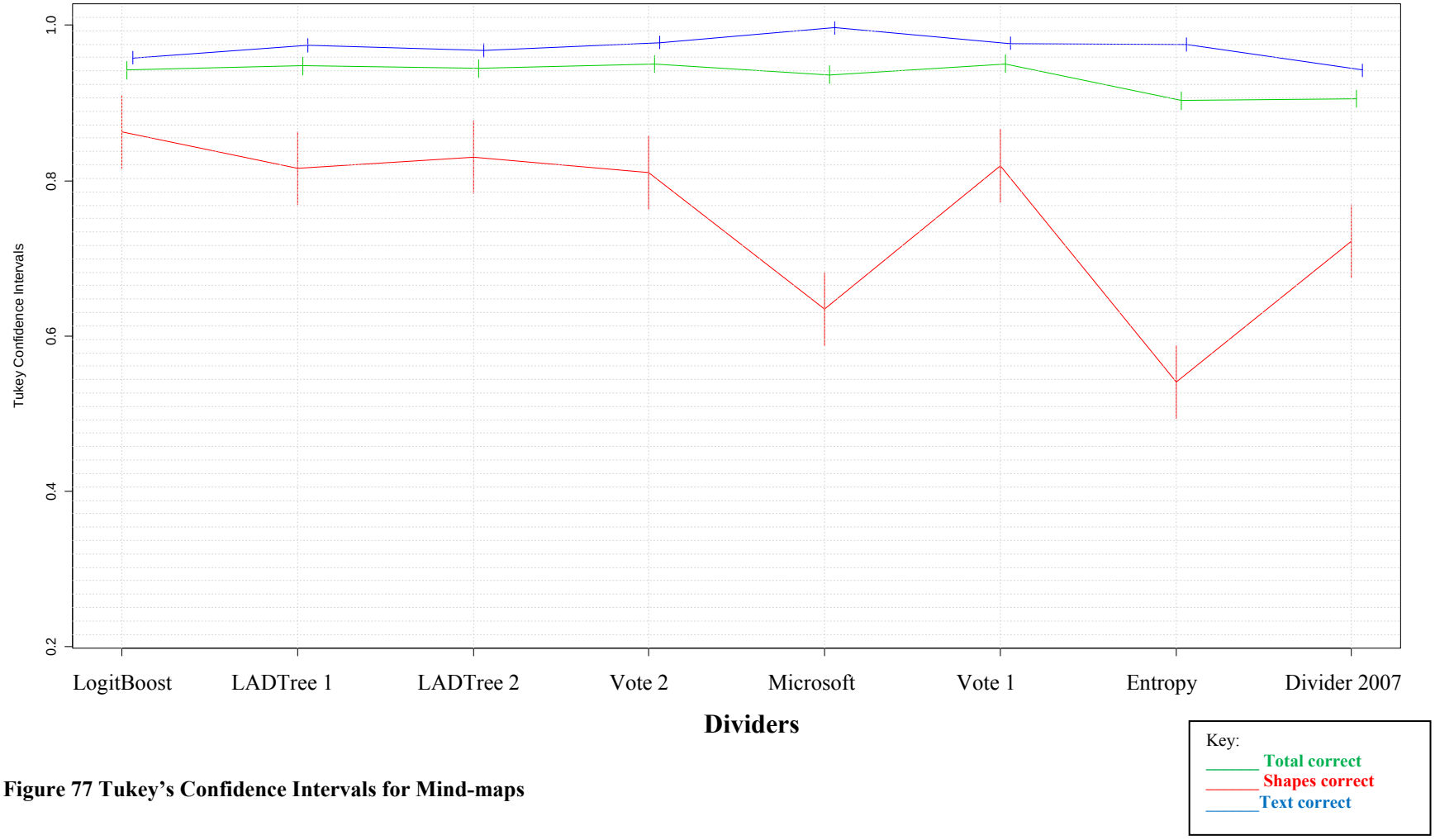
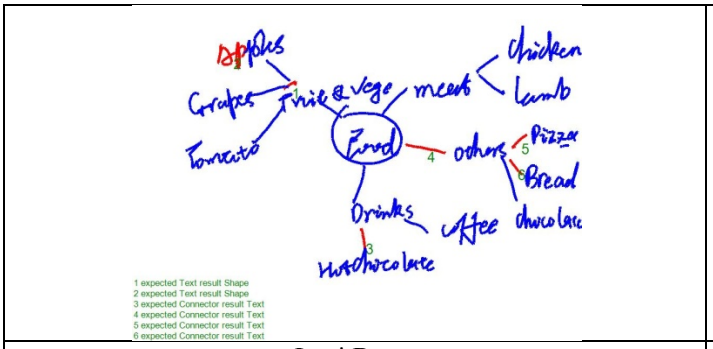
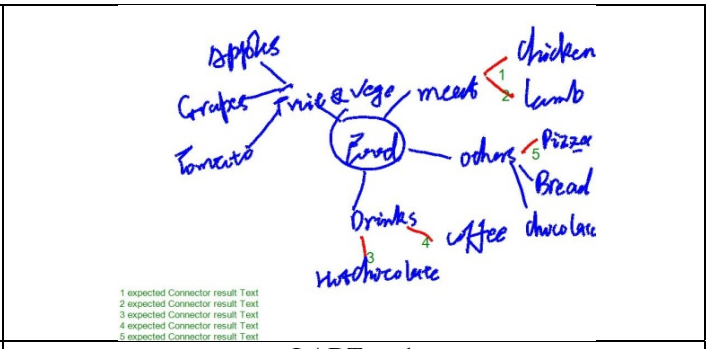


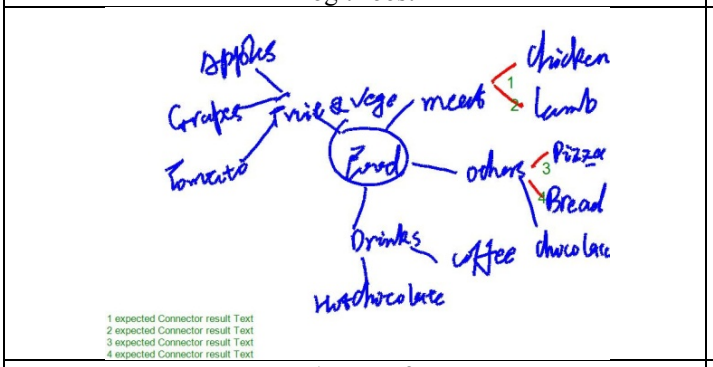
Figure 77 Tukey's Confidence Intervals for Mind-maps



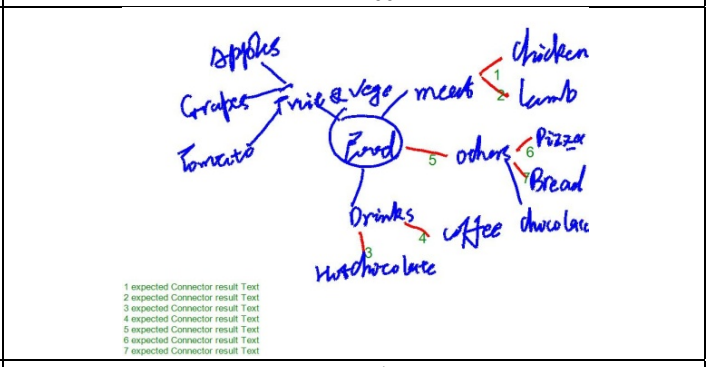
LogitBoost



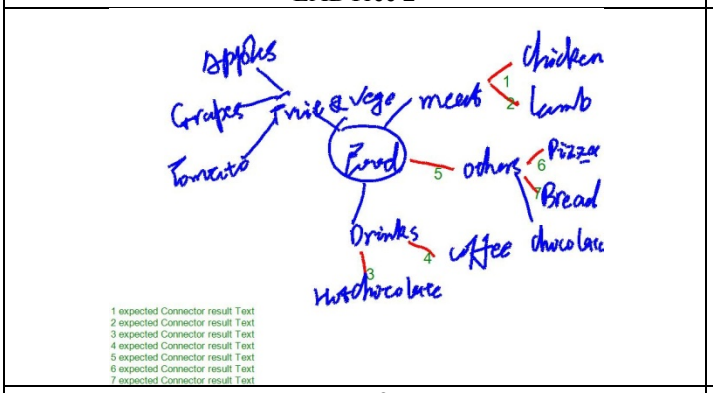
LADTree 1



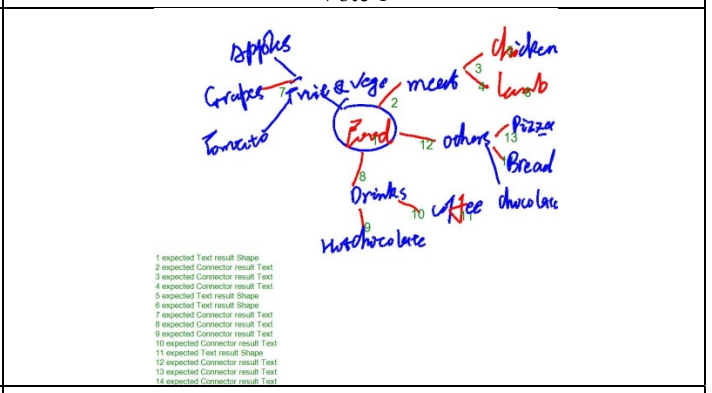
LADTree 2



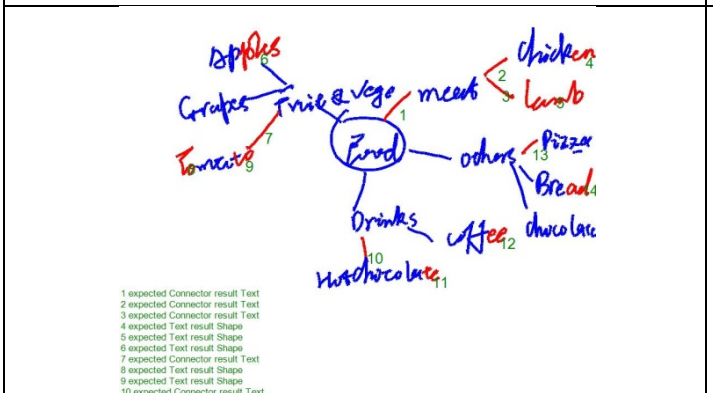
Vote 1



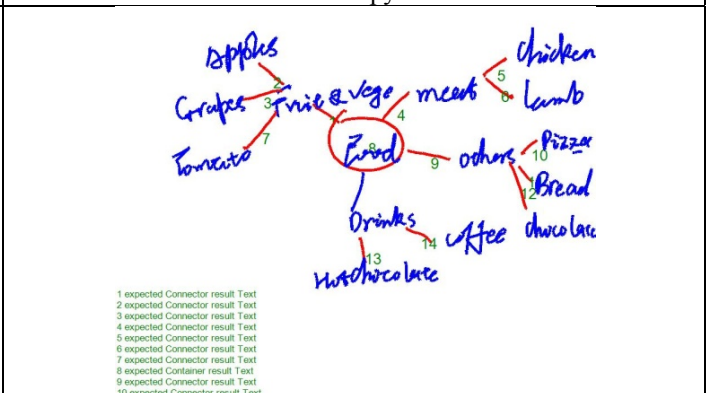
Vote 2



Entropy



Divider 2007



Microsoft

Figure 78 Examples of Strokes that are Misclassified by each Divider for a Mind-map. Strokes in red are misclassified strokes.

All the dividers have problems classifying connectors for this mind-map. The Microsoft divider is the only divider to misclassify the container: in fact, only one shape is classified correctly by this divider. LADTree 2 and LogitBoost are the best at classifying connectors in this mind-map. For LADTree 2 this may be because it has extra features to identify arrows and connectors. In terms of text classification, only Entropy, Divider 2007 and LogitBoost misclassify text; all others are able to identify all text in this mind-map correctly.

In summary, the new dividers are more accurate at classifying the mind-maps dataset than Divider 2007 and Entropy. They are not significantly different from the Microsoft divider, but closer inspection of this dividers results show it to be highly biased towards text.

7.3.2 To-do Lists

Results showing the performance of each divider on the to-do list dataset are shown in Table 43 along with Tukey's confidence intervals in Figure 79. The confidence intervals show that there is no significant difference overall between dividers; classification rates lie between 91.1% and 93.09%.

Results within each class show differences between divider performances. For the classification of text strokes the Microsoft divider is significantly more accurate than all other dividers, with 99.94% of text classified correctly. However, in terms of shape classification, the Microsoft divider is significantly less accurate than all the new dividers, with 34.83% of shape strokes classified correctly. The Entropy divider is also significantly less accurate than the new dividers for shape strokes, with 38.81% of shape strokes correctly classified. This bias is similar to the pattern for these dividers on the mind-map dataset. Overall, the rates of classification for text are much better than shapes in to-do lists. It must be noted that 89% of the to-do list dataset is text strokes.

	Number of strokes correct			**% Correct**		
Existing Dividers	**Text**	**Shapes**	**Total**	**Text**	**Shapes**	**Total**
Entropy	1663	78	1741	97.25	38.81	91.10
Microsoft	1709	70	1779	99.94	34.83	93.09
Divider 2007	1611	143	1754	94.21	71.14	91.78
New Dividers						
LADTree 1	1630	125	1755	95.32	62.19	91.84
LogitBoost	1630	127	1757	95.32	63.18	91.94
Vote 1	1632	122	1754	95.44	60.70	91.78
Vote 2	1630	120	1750	95.32	59.70	91.58
LADTree 2	1620	129	1749	94.74	64.18	91.52
Total	1710	201	1911			

Table 43 Results of All Dividers on To-do List Dataset

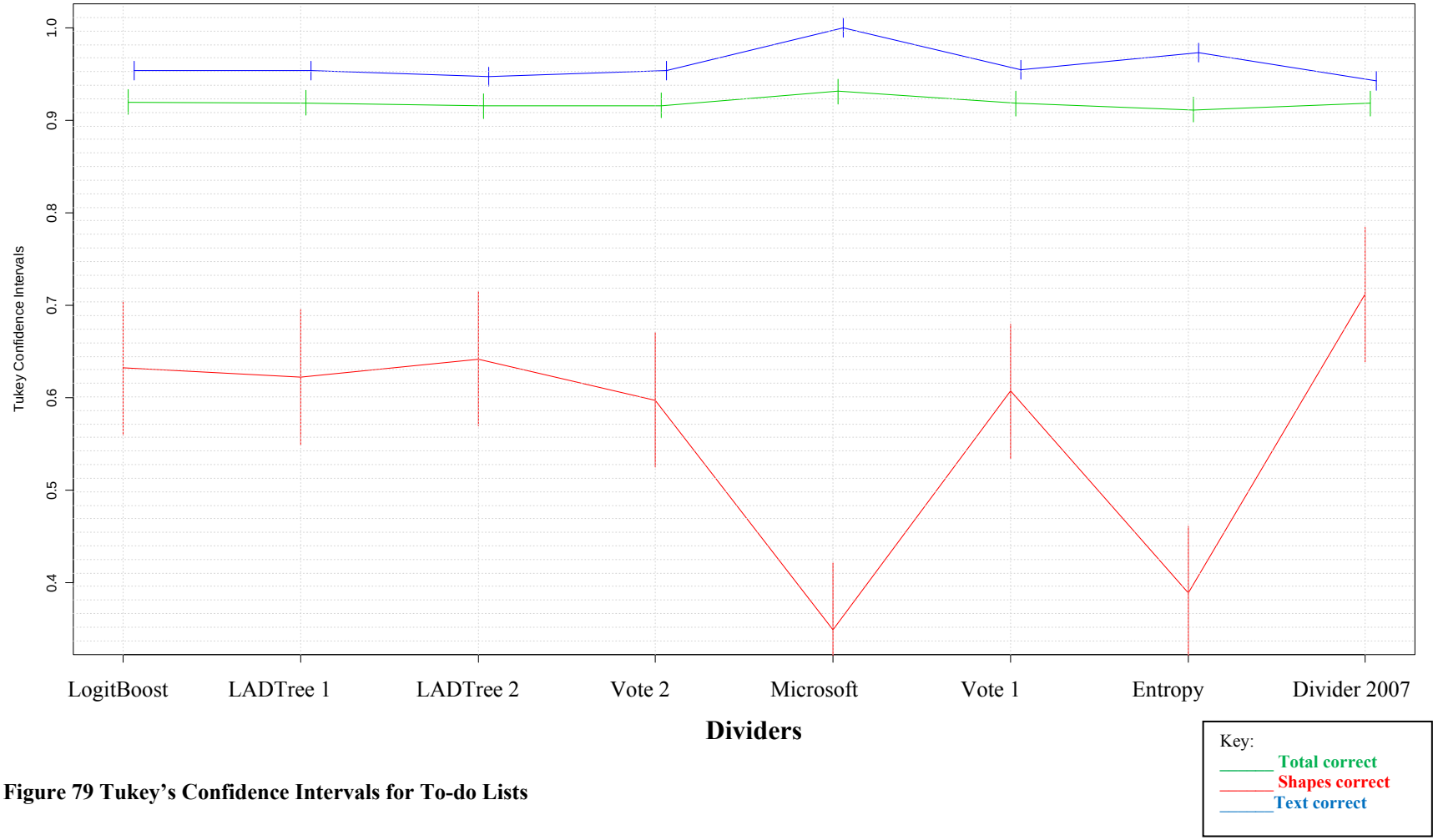
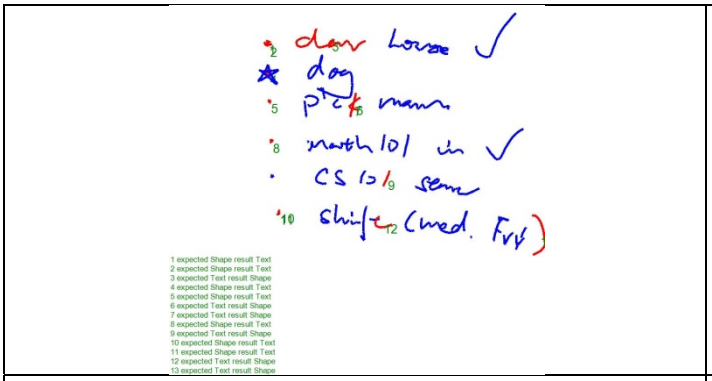
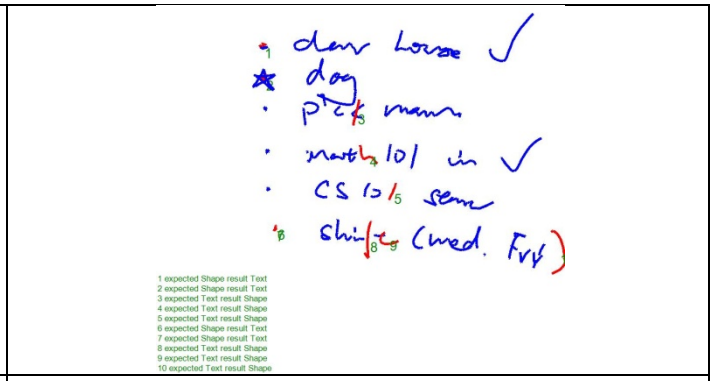


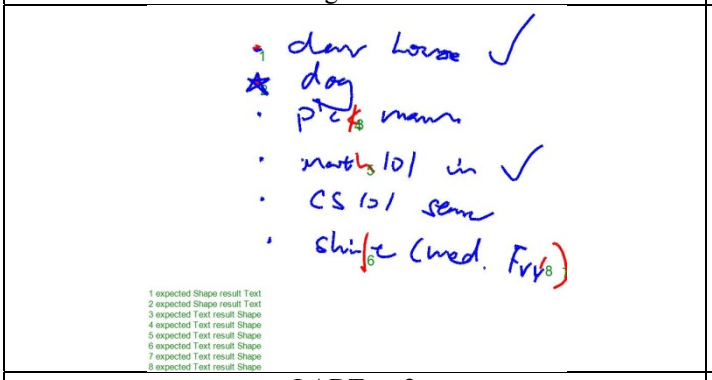
Figure 79 Tukey's Confidence Intervals for To-do Lists



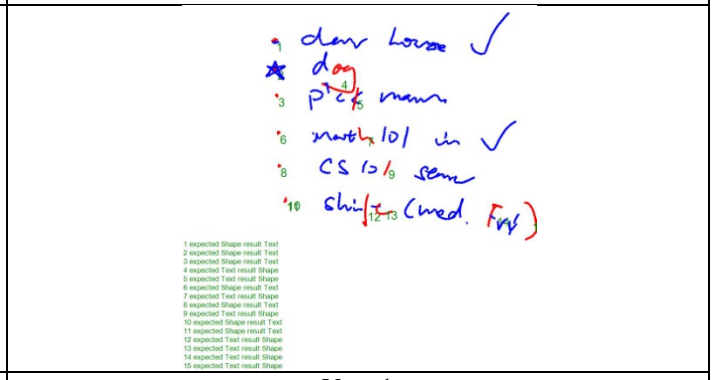
LogitBoost



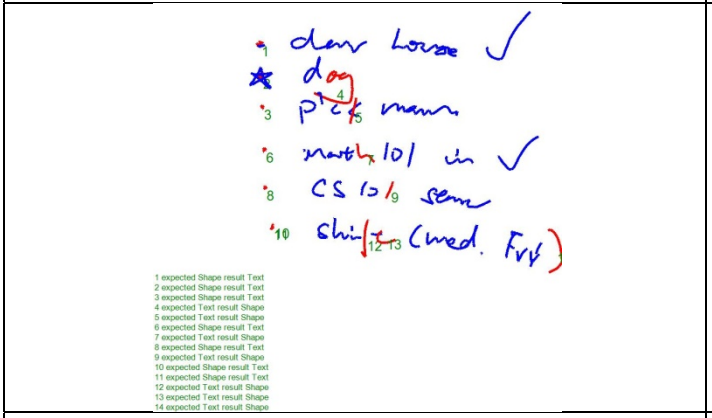
LADTree 1



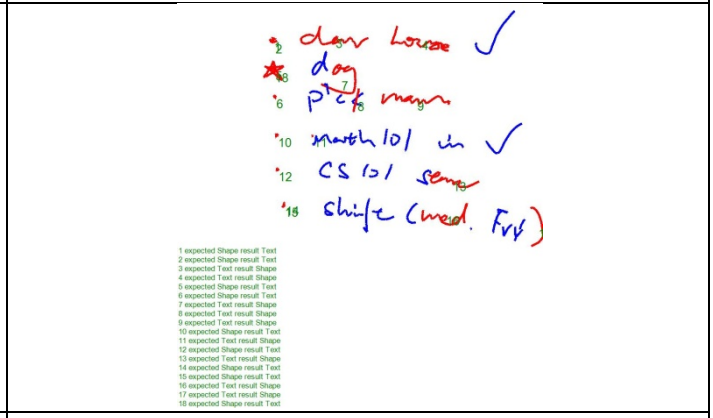
LADTree 2



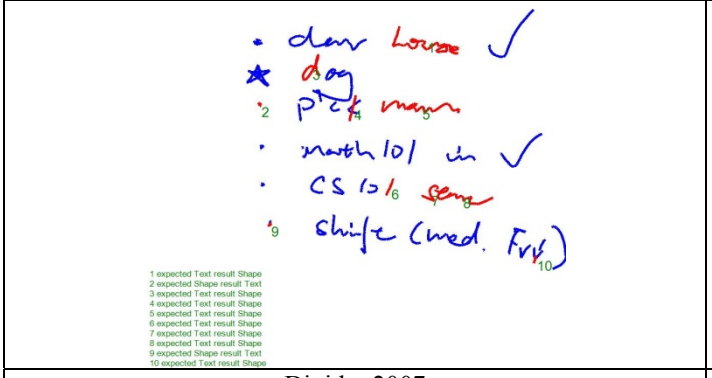
Vote 1



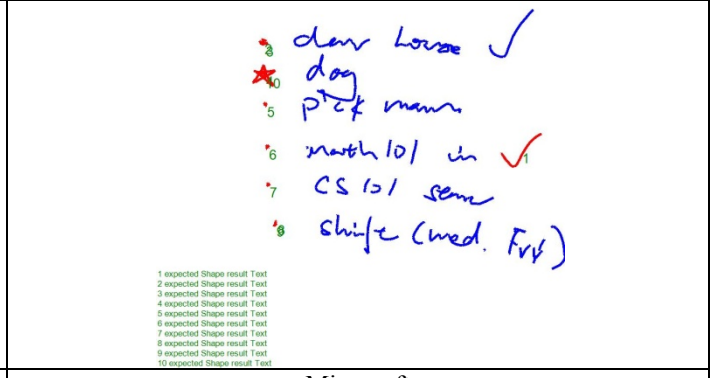
Vote 2



Entropy



Divider 2007



Microsoft

Figure 80 Examples of Strokes that are Misclassified by each Divider for a To-do List. Strokes in red are misclassified strokes.

Snapshots generated by DataManager's Evaluator (Schmieder 2009; Schmieder et al. 2009) for one participant's to-do list are shown in Figure 80. These snapshots highlight misclassified strokes by colouring them in red. This particular example is shown here because its average classification accuracy is very low in comparison to other participants' to-do list examples, so it serves as an interesting case to observe. The results for this example are very mixed. All dividers produce errors for text and shapes except for the Microsoft divider which classifies all text in this example correctly and all shapes incorrectly. The bullet points are a common source of errors in this to-do list. Two possible reasons for this are because of their close proximity to the text in the list and because they are small dots that can easily be confused with the dot of an 'i' or 'j'.

In summary, there is no significant difference between all dividers on to-do list documents. However, Entropy and the Microsoft divider have an extreme bias towards classifying most strokes as text, similar to the case of these dividers on the mind-map dataset. Dividers trained specifically for this document type may improve the results seen here. This is investigated in Section 7.4.1.

7.3.3 UML Class Diagrams

Table 44 shows the results of all dividers for the UML class diagram dataset. Tukey's confidence intervals are also shown in Figure 81 for this dataset. The overall results show that there is no significant difference between all dividers (classification rates are between 96.3% and 97.64%) except for Divider 2007 and Entropy. Divider 2007 and Entropy are significantly less accurate than all others with 85.89% and 90.88% of strokes correctly classified respectively.

The overall ranking of dividers based on Tukey's confidence intervals for the UML class diagram dataset is shown below.

Ranking for UML class diagram dataset:

1. LogitBoost, LADTree 1, LADTree 2, Vote 1, Vote 2, Microsoft
2. Entropy
3. Divider 2007

In terms of the classification of text, Divider 2007 is significantly less accurate than all other dividers; Microsoft and Entropy are not significantly different to Vote 1 and Vote 2, but are significantly more accurate than the other dividers. For shapes, all the new dividers are significantly more accurate than the existing dividers, classifying between 99.22% and 97.39% of shapes correctly. Entropy is significantly less

accurate than all dividers when classifying shapes with 60.57% of shape strokes correctly classified. Like the mind-map and to-do list datasets, most of this dataset is composed of text strokes; only 21% of the dataset is shapes. The difference between the percentage of text and shapes classified for each divider is not as large as what is observed for mind-maps and to-do lists; Entropy is the one exception.

Further statistics for each shape class are provided in Table 45. The number of strokes in each class is shown in brackets in the first row of the table. Rectangles and lines are classified well by most dividers, except for Entropy which correctly classifies 66.49% of the rectangles and 67.95% of lines in this dataset. LADTree 1 and LADTree 2 correctly classify all rectangles and lines. For triangles, arrows and diamonds, the new dividers correctly classify between 71.43% and 100% of these strokes. The existing dividers, on the other hand, have much lower classification rates for these shape classes. Classification rates for triangles lie between 4.76% and 57.14% for existing dividers. Entropy and Divider 2007 fail to classify any arrows or diamonds. The Microsoft divider classifies 37.5% of arrows and 10% of diamonds. However, because the number of triangle, arrow and diamond strokes in this dataset is very low, these results have a small effect on the overall accuracy rates for these dividers. On the other hand, rectangles and lines are more highly represented in this dataset.

	Number of strokes correct			**% Correct**		
Existing Dividers	Text	Shapes	Total	Text	Shapes	Total
Entropy	1462	232	1694	98.72	60.57	90.88
Microsoft	1465	346	1811	98.92	90.34	97.16
Divider 2007	1275	326	1601	86.09	85.12	85.89
New Dividers						
LADTree 1	1424	374	1798	96.15	97.65	96.46
LogitBoost	1422	373	1795	96.02	97.39	96.30
Vote 1	1442	378	1820	97.37	98.69	97.64
Vote 2	1443	375	1818	97.43	97.91	97.53
LADTree 2	1417	380	1797	95.68	99.22	96.41
Total	1481	383	1864			

Table 44 Results for All Dividers on UML Class Diagram Dataset

	% Correct				
Existing Dividers	**Rectangles (188)**	**Lines (156)**	**Triangles (21)**	**Arrows (8)**	**Diamonds (10)**
Entropy	66.49	67.95	4.76	0	0
Microsoft	100.00	94.23	33.33	37.50	10.00
Divider 2007	88.83	94.23	57.14	0	0
New Dividers					
LADTree 1	100	100	71.43	87.5	80
LogitBoost	98.94	98.08	85.71	75	100
Vote 1	100	99.36	90.48	87.5	90
Vote 2	100	99.36	80.95	87.5	80
LADTree 2	100	100	85.71	100	100

Table 45 Classification Rates for each Shape Class in the UML Class Diagram Dataset

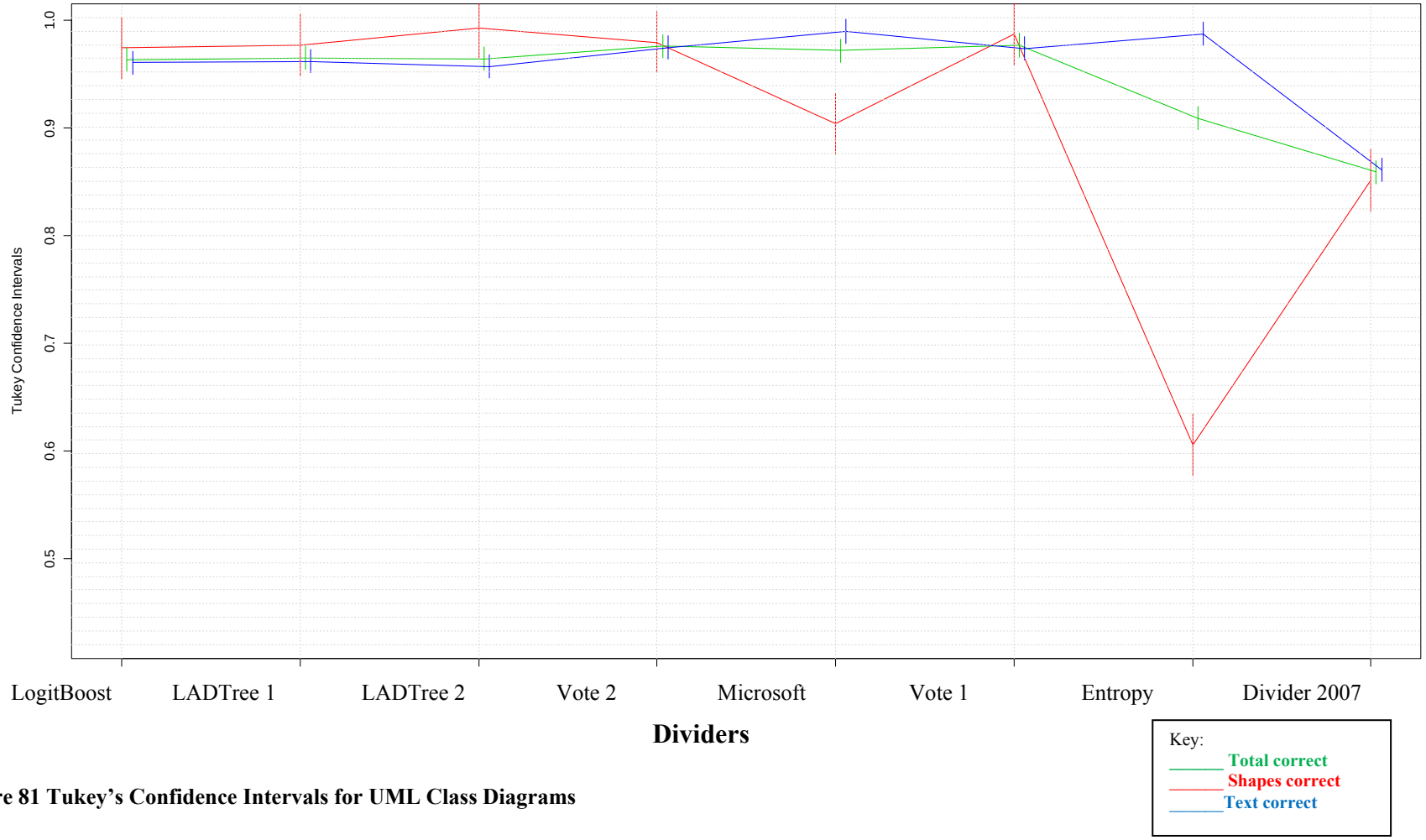


Figure 81 Tukey's Confidence Intervals for UML Class Diagrams

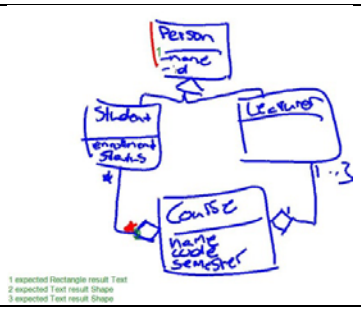
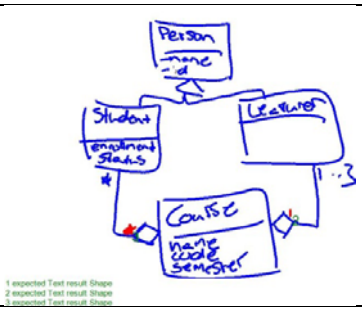
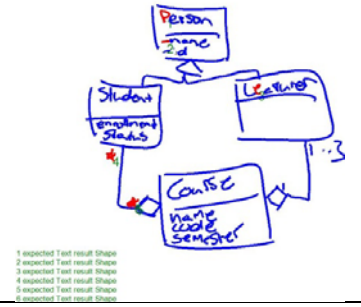
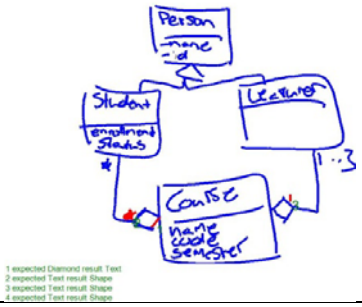
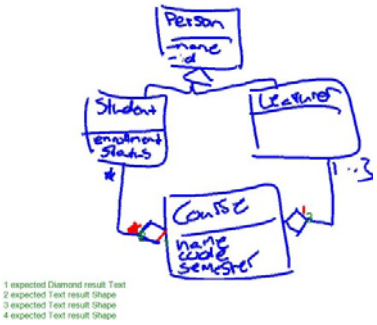
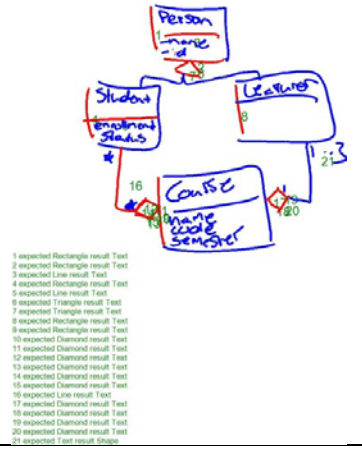
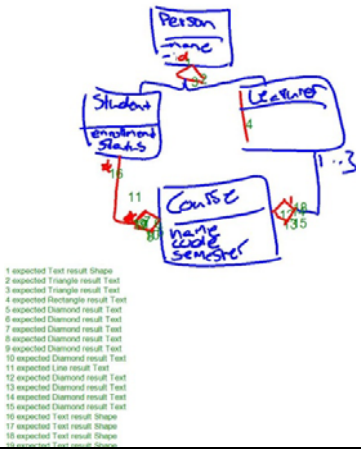
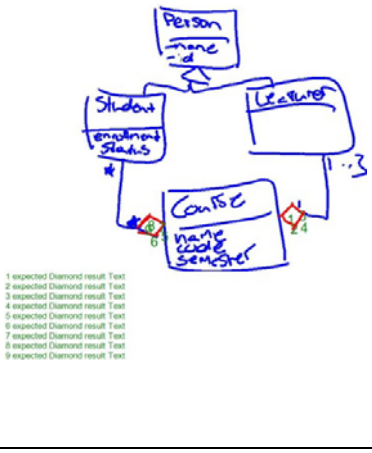
	
LogitBoost	LADTree 1
	
LADTree 2	Vote 1
	
Vote 2	Entropy
	
Divider 2007	Microsoft

Figure 82 Examples of Strokes that are Misclassified by each Divider for a UML Class Diagram

Snapshots of one participant's UML class diagram, generated using DataManager's Evaluator, are shown in Figure 82. Strokes in red are misclassified and those in blue are correctly classified. This particular example is shown here as it has a low average classification rate in comparison to others in the dataset. Entropy and Divider 2007 stand out as they show a high number of misclassified strokes, most of which are shapes. The asterisks marking cardinality are a common source of misclassification for the new dividers, whereas diamonds are commonly misclassified among the existing dividers. Text inside the rectangles is classified well on the whole for this example.

Overall, in comparison to other datasets shown in Figure 76, the UML class diagrams are the most well classified dataset on average. The new dividers are significantly more accurate at classifying the UML class diagram dataset than Divider 2007 and Entropy but not significantly different from the Microsoft divider. The Microsoft divider does not have as much of a bias towards text as in the case for to-do lists and mind-maps, but its classification rate for shapes is significantly lower than the new dividers.

7.3.4 Euler Diagrams

The classification results of the dividers tested against the Euler diagram dataset are in Table 46 and Tukey's confidence intervals are presented in Figure 83. Overall results show that Entropy is the highest performing divider with a correct classification rate of 93.17%; however, the confidence intervals show that it is not significantly different from Vote 2. Vote 2 is also not significantly different to all the new dividers except LADTree 2. Divider 2007 and the Microsoft divider are significantly less accurate than all others with correct classification rates of 65.6% and 74.16% respectively.

The overall ranking of dividers based on Tukey's confidence intervals for the Euler diagram dataset is shown below. Some dividers are ranked at two levels because they are not significantly different to the dividers in both levels. For example, Vote 2 is not significantly different to Entropy so, it is ranked at position 1. However, it is also ranked at position 2 as it is not significantly different to dividers at that level either, whereas Entropy is significantly different to the dividers at level 2.
Ranking for Euler diagram dataset:

1. Entropy, Vote 2
2. Vote 2, Vote 1, LogitBoost, LADTree 1,
3. LogitBoost, LADTree 1, LADTree 2
4. Microsoft
5. Divider 2007

Entropy is significantly more accurate at classifying text when compared to all other dividers, with 92.98% of text correctly classified. The Microsoft divider is not significantly different from the new models when classifying text (77.72% correct) and Divider 2007 is significantly less accurate as classifying text than all other dividers (43.58% correct). For shapes, the Microsoft divider is significantly less accurate than all other dividers, where 69.76% of shape strokes were correctly classified. All other dividers are not significantly different from each other at classifying shapes.

With the exception of the Microsoft divider, all dividers have a higher percentage of correctly classified shapes over text: for all other datasets tested this is generally the opposite. The Euler diagram dataset is the most balanced of all the test sets, where 55% are text strokes and 45% are shape strokes as shown in Table 36. Other datasets have higher proportions of text strokes than shapes, except for logic diagrams. The shapes in Euler diagrams are ellipses and circles which, according to Table 28, are well classified by our divider models when tested on the verification dataset. A possible reason for the difficulty with classifying text is that the text in these Euler diagrams is single letters rather than words.

	Number of strokes correct			**% Correct**		
Existing Dividers	**Text**	**Shapes**	**Total**	**Text**	**Shapes**	**Total**
Entropy	384	312	696	92.98	93.41	93.17
Microsoft	321	233	554	77.72	69.76	74.16
Divider 2007	180	310	490	43.58	92.81	65.60
New Dividers						
LADTree 1	340	305	645	82.32	91.32	86.35
LogitBoost	301	323	624	72.88	96.71	83.53
Vote 1	344	309	653	83.29	92.51	87.42
Vote 2	346	310	656	83.78	92.81	87.82
LADTree 2	301	307	608	72.88	91.92	81.39
Total	413	334	747			

Table 46 Results for All Dividers on the Euler Diagram Dataset

Figure 84 show snapshots generated by DataManager's Evaluator for one participant's Euler diagram. This example has a low average classification rate in comparison to the rest of the dataset. The strokes in red highlight misclassifications. The Microsoft divider is the only one to misclassify shape strokes in this example, but it correctly classifies all text strokes. All other dividers struggle to correctly classify the text; LogitBoost actually fails to classify any text strokes.

In summary, Entropy is significantly more accurate than all other dividers except Vote 2, where there is no significant difference. Euler diagrams have a unique composition in comparison to other domains and this is reflected by the low results obtained by using general dividers on such diagrams. A divider trained specifically for this diagram type may produce higher classification rates: this is investigated in Section 7.4.2.

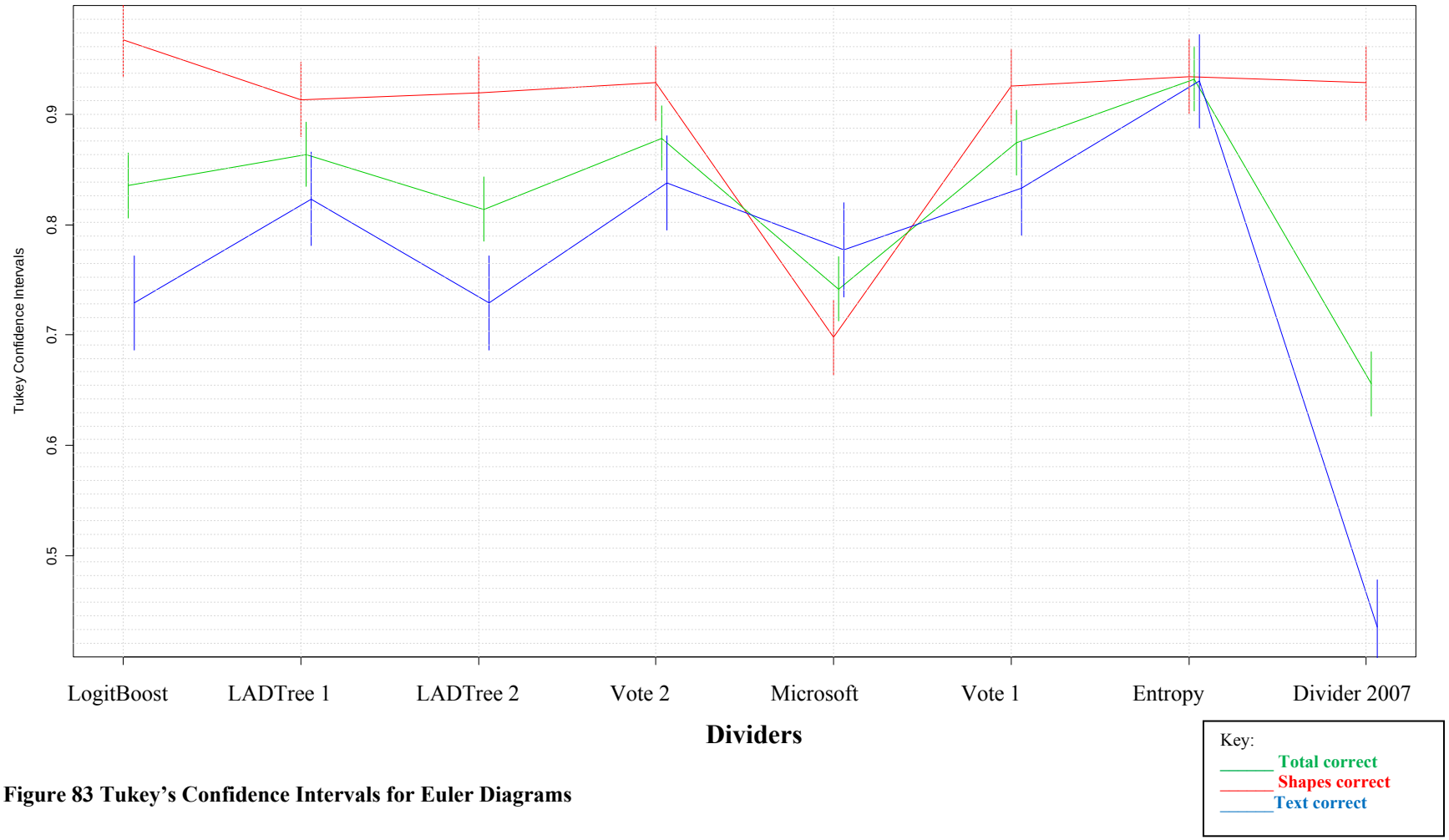


Figure 83 Tukey's Confidence Intervals for Euler Diagrams

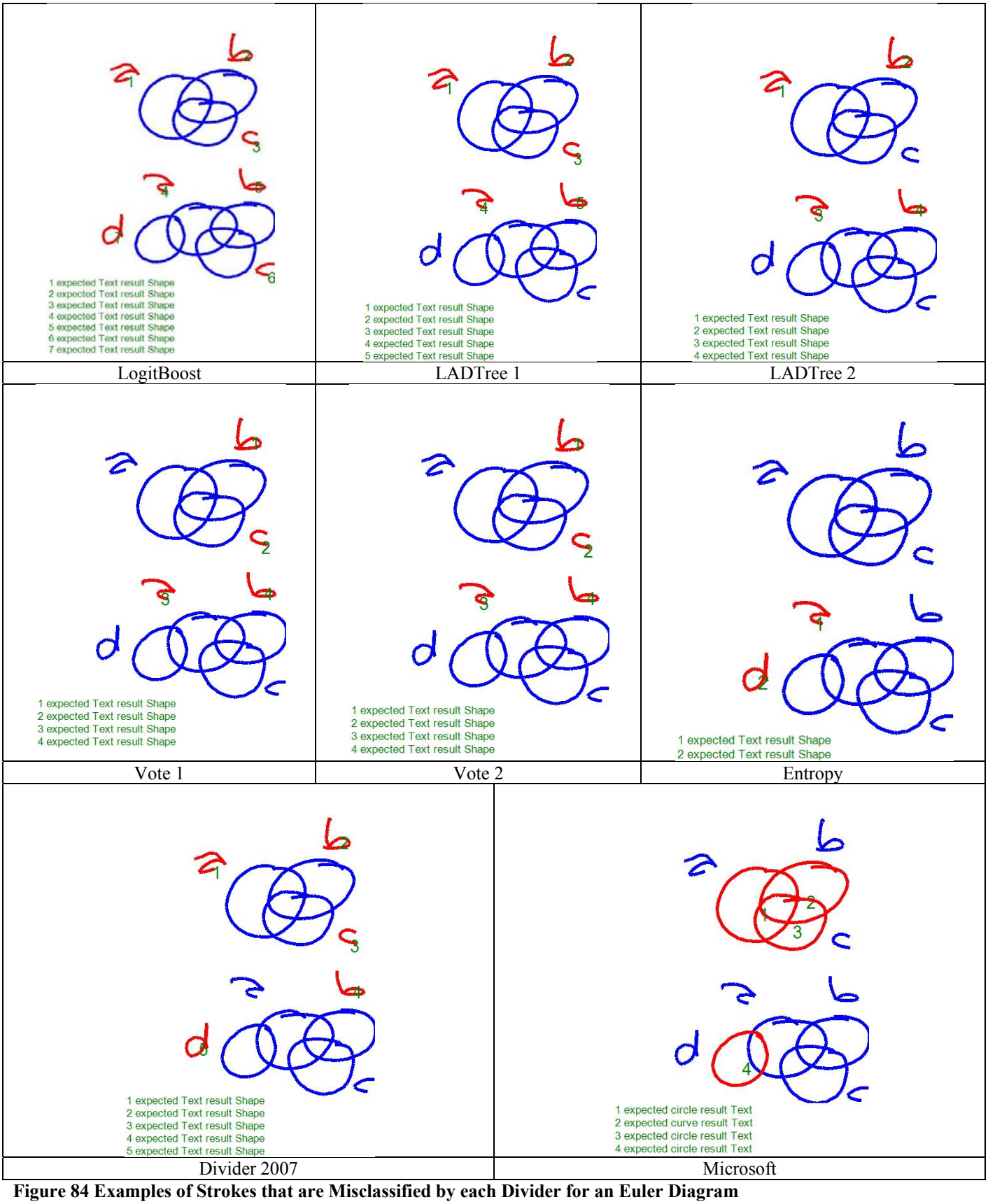


Figure 84 Examples of Strokes that are Misclassified by each Divider for an Euler Diagram

7.3.5 COA Symbols

Table 47 shows all the results for the dividers on the COA dataset and Figure 86 presents their Tukey's confidence intervals. The confidence intervals for the overall results show that there is no significant difference between all dividers, which have classification rates ranging from 89.73% to 93.01%, except the Microsoft divider. The Microsoft divider is significantly less accurate than all dividers, correctly classifying 81.51% of the COA dataset.

The overall ranking of dividers based on Tukey's confidence intervals for the COA symbol dataset is shown below.

Ranking for COA symbol dataset:

1. LogitBoost, LADTree 1, LADTree 2, Vote 1, Vote 2, Divider 2007, Entropy
2. Microsoft

In terms of classifying text, the Microsoft divider is significantly more accurate at classifying text than LogitBoost and LADTree 1, but not significantly different from the other new divider models. Entropy and Divider 2007 are significantly worse at classifying text than Vote 2, and not significantly different from all other new divider models. However, in terms of shape classification, the Microsoft divider is significantly less accurate than all other dividers with 40.19% of shapes classified correctly. All other dividers are not significantly different when classifying shape strokes. Overall the dividers are more successful at classifying text in COA symbols than shapes.

	Number of strokes correct			**% Correct**		
Existing Dividers	**Text**	**Shapes**	**Total**	**Text**	**Shapes**	**Total**
Entropy	490	172	662	94.96	80.37	90.68
Microsoft	509	86	595	98.64	40.19	81.51
Divider 2007	485	175	660	93.99	81.78	90.41
New Dividers						
LADTree 1	484	171	655	93.80	79.91	89.73
LogitBoost	474	181	655	91.86	84.58	89.73
Vote 1	500	179	679	96.90	83.64	93.01
Vote 2	508	161	669	98.45	75.23	91.64
LADTree 2	493	169	662	95.54	78.97	90.68
Total	516	214	730			

Table 47 Results for All Dividers on the COA Dataset

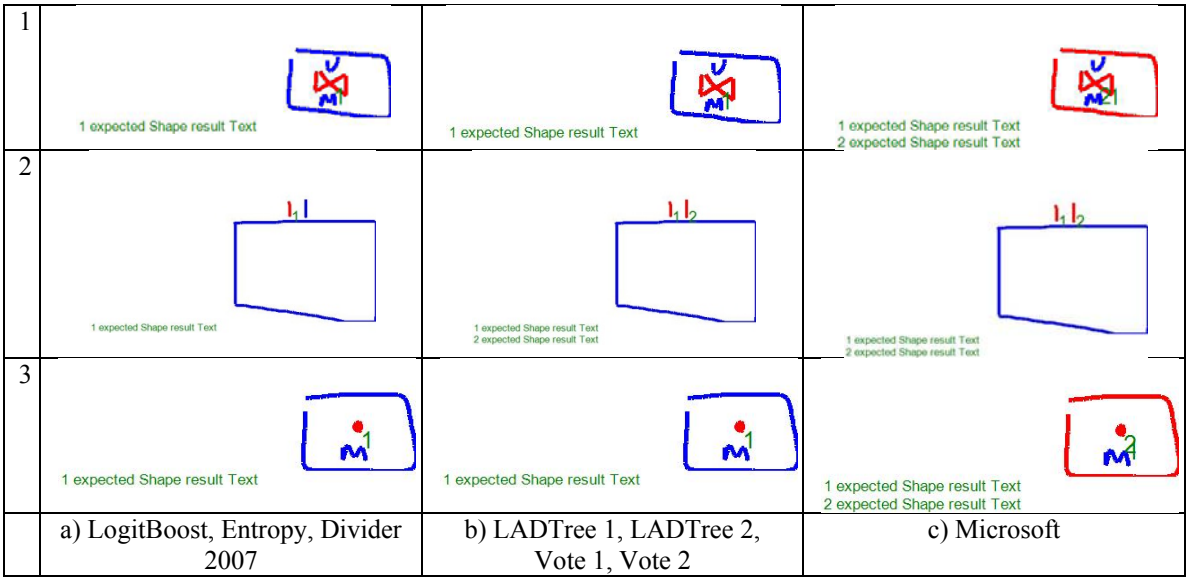


Figure 85 Examples of Strokes that are Misclassified by each Divider for COA Symbols

DataManager's Evaluator was used to generate snapshots of three examples from the COA dataset that had low average classification rates in comparison to the rest of the dataset: they are shown in Figure 85. The snapshots highlight misclassified strokes in red for each divider. The examples are grouped by dividers that share the same misclassifications for these examples. All misclassifications shown in these examples are for shapes. All dividers fail at classifying the triangular symbol in the first example, one of the short lines in example 2 and the dot in example 3. Groups (b) and (c) also fail to correctly classify the second small line in example 2. The Microsoft divider also misclassifies the rectangles in examples 1 and 2. In fact, only one shape in the three examples shown is correctly classified by the Microsoft divider. This highlights the bias this divider has towards text.

In summary, there is no significant difference between the new dividers and Divider 2007 and Entropy on the COA dataset. However, the Microsoft divider is significantly less accurate than the new dividers. The original Entropy divider (Bhat et al. 2009) was trained using this dataset. Although this version of Entropy was trained using our own training set, this divider has been designed and built with COA symbols in mind. The COA dataset is a very simplistic set of symbols rather than full diagrams. Divider 2007 was trained on similar data; in particular it was trained on isolated diagram components rather than full diagrams. In contrast, the new dividers have been trained on full diagrams. These differences may explain why Entropy and Divider 2007 do not have significantly different results to the new dividers on this dataset. A domain-specific divider trained especially for COA symbols may improve results: this is explored in Section 7.4.3.

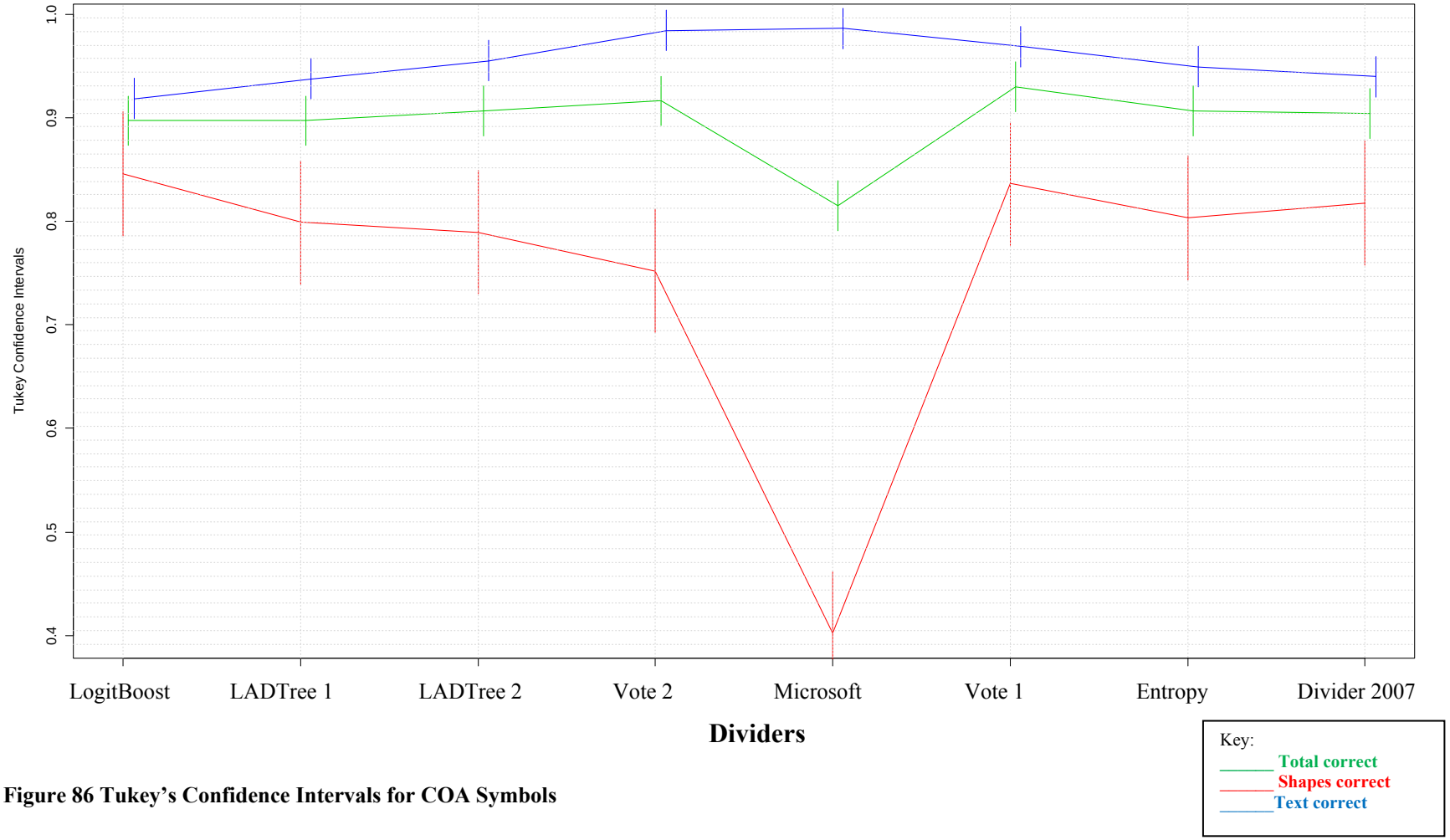


Figure 86 Tukey's Confidence Intervals for COA Symbols

7.3.6 Logic Diagrams

The results of testing each divider on the logic diagram dataset are shown in Table 48, with corresponding Tukey's confidence intervals shown in Figure 87. Overall, these results are the lowest of all the datasets tested, as shown in Figure 76, where classification rates range between 43.05% and 81.94%. This dataset is also the largest of the test group. The balance of text to shape strokes is different to all other datasets as it has a larger proportion of shape strokes than text: this is shown in Table 36 where 73% of the dataset are shape strokes.

LogitBoost is significantly more accurate than any other divider at classifying these logic diagrams with 81.94% of strokes correctly classified. Entropy is significantly less accurate than all other dividers with only 43.05% of strokes correctly classified. Divider 2007 is significantly less accurate than all the new dividers, except Vote 1 which is not significantly different. The Microsoft divider is significantly less accurate than LogitBoost, LADTree 1 and LADTree 2, and significantly more accurate than Vote 1 and Vote 2.

The overall ranking of dividers based on Tukey's confidence intervals for the logic diagram dataset is shown below.

Ranking for logic diagram dataset:

1. LogitBoost
2. LADTree 1
3. LADTree 2
4. Microsoft
5. Vote 2
6. Vote 1, Divider 2007
7. Entropy

In terms of classifying text, Entropy is significantly more accurate than all other dividers, with a classification rate of 98.43%, except for Vote 1 which is not significantly different. The Microsoft divider and Divider 2007 are significantly more accurate at classifying text than LogitBoost, LADTree 1 and LADTree 2 and significantly less accurate than Vote 1 and Vote 2. For shapes, LogitBoost is significantly more accurate than all other dividers with 93.07% of shape strokes correctly classified. Entropy is significantly less accurate at classifying shapes than all other dividers with 22.93% of shapes correctly

classified. Divider 2007 is not significantly different to Vote 1 and significantly less accurate than all other new dividers when classifying shapes. The Microsoft divider is significantly more accurate than Vote 1 and 2 at classifying shapes and significantly less accurate than the other new dividers. The percentage of shapes correctly classified is lower than text for all dividers except LADTree 1 and LogitBoost.

	Number of strokes correct			% Correct		
Existing Dividers	**Text**	**Shapes**	**Total**	**Text**	**Shapes**	**Total**
Entropy	2260	1449	3709	98.43	22.93	43.05
Microsoft	1854	4249	6103	80.75	67.23	70.83
Divider 2007	2078	2140	4218	90.51	33.86	48.96
New Dividers						
LADTree 1	1763	5028	6791	76.79	79.56	78.82
LogitBoost	1178	5882	7060	51.31	93.07	81.94
Vote 1	2229	2109	4338	97.08	33.37	50.35
Vote 2	2141	3356	5497	93.25	53.10	63.80
LADTree 2	1765	4813	6578	76.87	76.16	76.35
Total	2296	6320	8616			

Table 48 Results for All Dividers on the Logic Diagram Dataset

More information on the percentage of strokes correct for each shape class is given in Table 49, along with an example of each shape class and the number of strokes for each class in the dataset. The range of results for each shape class is very large; for example, for wires classification rates range from 23.25% to 93.19%. Because of the large range of results for each shape class, it is difficult to determine which shape class results in the most misclassifications overall. There is very little difference between most shape classes, most include some form of a semi-circle and sometimes with a small circle attached. LogitBoost produces the highest results for each shape class. LADTree 2 has the lowest results for a four of the shape classes and Entropy has the lowest results for three classes. It is surprising that even the simplest shapes such as wires and bubbles have such low classification rates. This may be because wires can have many overlaps and curves which may make them easily mistaken for text, while bubbles are small circles like the letter ‘o’.

Snapshots of one participant’s logic diagram, generated by DataManager’s Evaluator, are shown in Figure 88. This example was chosen as it has a high number of misclassified strokes on average across all dividers. Strokes in red are misclassified strokes. The existing dividers and Vote 1 are all successful at recognising all text strokes in this example but do not identify shape strokes well. The rest of the new dividers are able to classify shapes more successfully. A common source of error for this example is bubbles. As stated earlier, bubbles are small circles similar to the letter ‘o’ and thus can easily be mistaken for text.

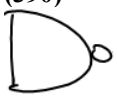
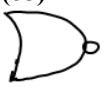
Existing Dividers	% Correct								
	Wire (4349)	**AND (643)** 	**OR (334)** 	**NOT (355)** 	**NAND (390)** 	**XOR (35)** 	**BUBBLE (96)** 	**NOR (85)** 	**Other (33)**
Entropy	25.25	21.46	37.13	1.41	15.90	11.43	0	18.82	6.06
Microsoft	70.75	84.76	73.65	34.37	37.95	100.00	18.75	44.71	60.61
Divider 2007	39.30	32.04	26.95	3.10	22.56	8.57	17.71	14.12	12.12
New Dividers									
LADTree 1	80.34	92.38	90.72	46.76	77.44	85.71	50.00	81.18	66.67
LogitBoost	93.19	97.51	97.90	86.20	89.49	100.00	77.08	96.47	87.88
Vote 1	37.85	35.77	37.13	6.20	11.54	40.00	3.13	25.88	9.09
Vote 2	53.35	71.07	72.16	13.52	50.00	62.86	13.54	61.18	24.24
LADTree 2	23.25	11.98	13.17	58.03	23.33	17.14	48.96	10.59	48.48

Table 49 Classification Rates for each Shape Class in the Logic Diagrams Dataset

In summary, LogitBoost is significantly more accurate than all other dividers on the logic diagram dataset. This dataset differs from others as it was collected "in the wild" from students' assignments, notes and reports. Logic diagrams also have a unique semantic structure not seen in any other datasets. Such diagrams may benefit from a divider trained specifically for the domain rather than a general text-shape divider. Domain-specific dividers for logic diagrams are investigated in Section 7.4.4.

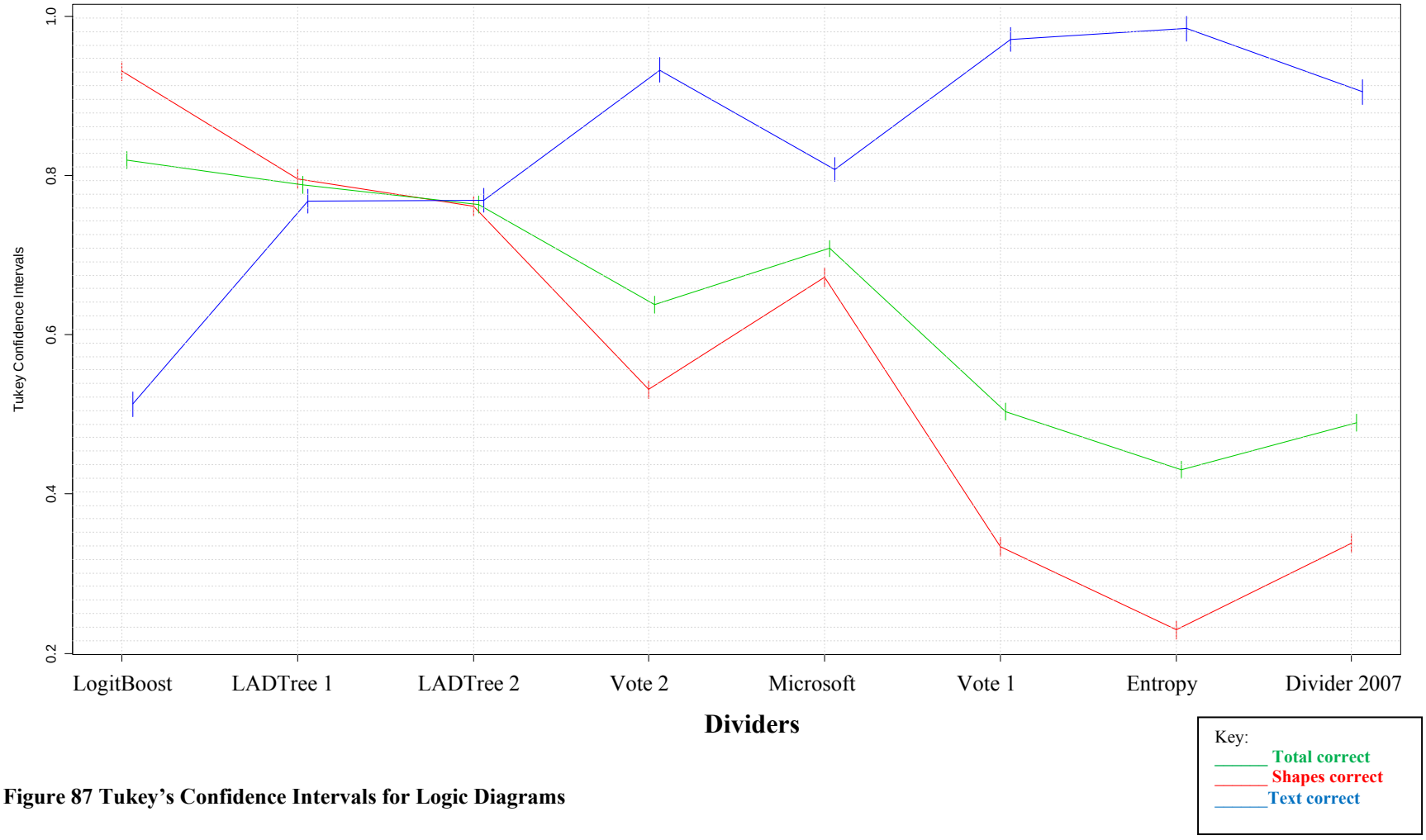


Figure 87 Tukey's Confidence Intervals for Logic Diagrams

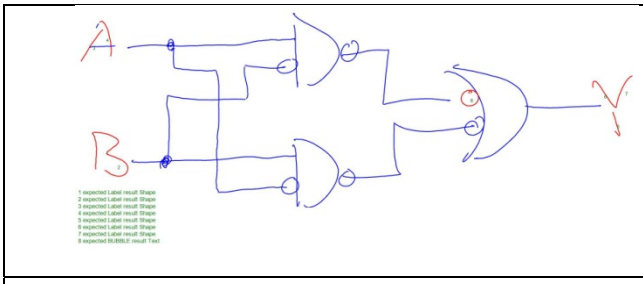
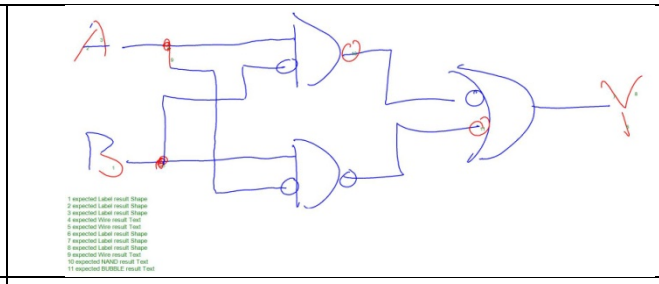
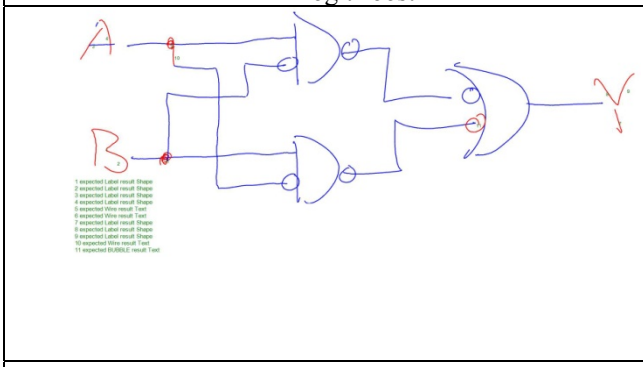
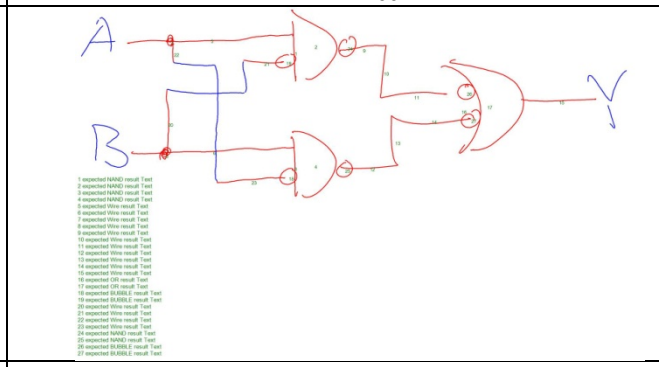
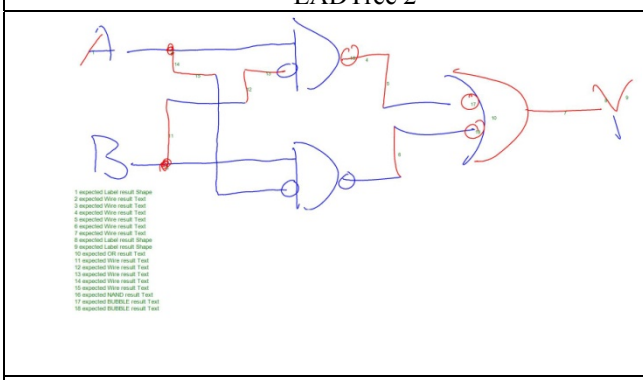
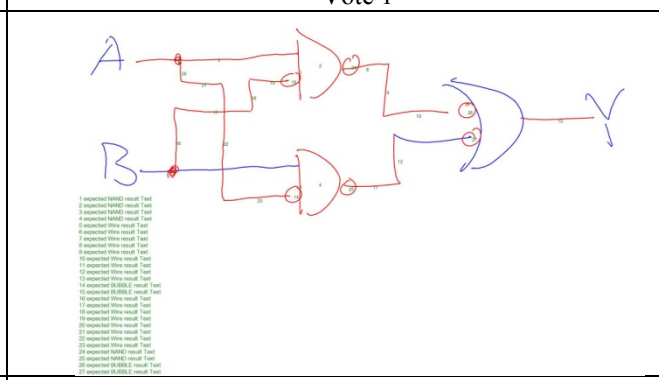
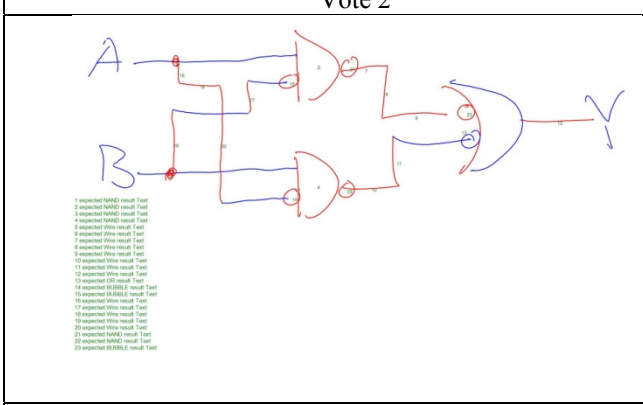
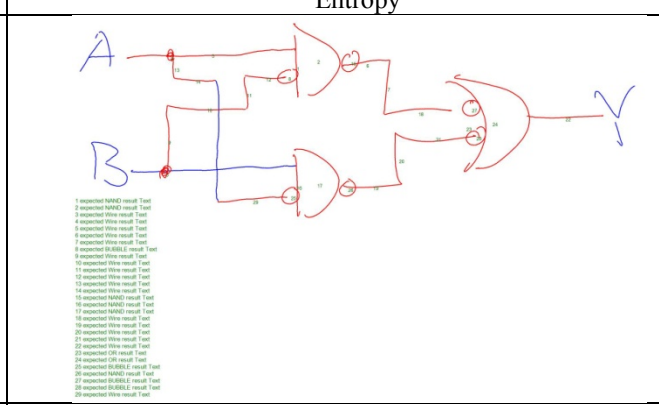
	
LogitBoost	LADTree 1
	
LADTree 2	Vote 1
	
Vote 2	Entropy
	
Divider 2007	Microsoft

Figure 88 Examples of Strokes that are Misclassified by each Divider for a Logic Diagram

7.3.7 Divider Classification Times

The focus of this work is on improving the accuracy of text-shape dividers rather than the time taken for classification. The average time for each divider to classify a stroke is presented in Table 50 for completeness. The new dividers clearly take much longer to classify strokes than the existing dividers. Entropy and Divider 2007 are very fast because there are only 1 to 8 feature calculations required for these dividers. Microsoft does not provide details on the implementation of its divider.

The new dividers use models generated by Weka. To use these models in our implementation, the Weka libraries are utilised. The design of these libraries is such that all the features that the model is trained with must be provided regardless of whether they are used in the final classification or not. This requires that all 114 features in the feature library are calculated for each stroke when making a classification; this clearly increases the amount of time it takes to classify a stroke. There are many ways to make the classification time faster, but addressing this issue is out of scope for this thesis. A more detailed discussion of this can be found in Sections 8.4 and 9.2.

Existing Dividers	Average time (seconds) per stroke
Entropy	0.0004
Microsoft	0.0159
Divider 2007	0.0009
New Dividers	
LADTree 1	0.4561
LogitBoost	0.4652
Vote 1	0.4714
Vote 2	0.4710
LADTree 2	0.4766

Table 50 Average Time (seconds) to Classify a Stroke

7.4 Domain-Specific Dividers

When the domain of a diagram is known, recognisers can be built specifically for that domain with the expectation that they will be more successful at classifying such diagrams. This expectation is based on the fact that recognisers are only trained on the types of diagrams that they will be used to classify. This section describes a brief, final exploration into the value of domain-specific dividers.

The results of the general divider models on each dataset in the previous section show that there is room for improvement in several domains. The four datasets with the lowest results, based on the Tukey confidence intervals in Figure 76, are used for an investigation into the value of domain-specific dividers. These datasets are: to-do lists, Euler diagrams, COA symbols and logic diagrams. The UML class diagrams and mind-maps are classified well by the general dividers so we have chosen not to produce domain-specific dividers for these diagrams.

To generate each domain-specific divider, half of the corresponding dataset is used for training using ten-fold cross validation and the other half for testing. LADTree 1 and LogitBoost are used as the algorithms for these dividers as they are significantly more accurate on average than other algorithms, as shown in Figure 76 and Table 40. The extended feature set has been used here. However, the second parse feature has not been included as results suggest that using this feature comes at the cost of misclassification of other classes, as described in Section 6.5.2.

The results for the domain-specific dividers are described below.

7.4.1 To-do Lists

To build a divider for to-do lists, LADTree 1 and LogitBoost were trained with data from the first ten participants of the to-do list dataset; data from the other ten participants was used for testing. Summary statistics of the training and testing datasets are in Table 51. In addition, the original LADTree 1 and LogitBoost models, trained as general dividers, were tested on the same test dataset, along with the three existing dividers. The results from training and testing on the to-do list dataset are shown in Table 52.

The to-do list trained dividers both have an overall training accuracy of 97.21%. On the test dataset, LADTree 1 (to-do list) correctly classifies 95.18% of all strokes and LogitBoost correctly classifies 95.60% of all strokes. The general dividers' accuracy on the test set ranges from 90.41% to 92.52%.

Figure 89 shows Tukey's confidence intervals for each divider tested on the to-do list dataset. These intervals show that the domain-specific dividers for to-do list are significantly more accurate than the original general dividers, LADTree 1 and LogitBoost, as well as Divider 2007 and Entropy. There is a very slight overlap between the Microsoft divider and the domain-specific dividers' confidence intervals, indicating that they are not significantly different to each other. It is possible that if additional data is added to the test set, this slight overlap may not occur due to the narrowing effect that larger datasets have on these confidence intervals. This must be left to future work.

In terms of classifying text, the domain-specific dividers are significantly more accurate than the original general dividers, LADTree 1 and LogitBoost, and Divider 2007. There is no significant difference between the domain-specific dividers, Entropy, and the Microsoft divider when classifying text strokes.

For shape classification, however, Entropy and Microsoft dividers are significantly less accurate than all others. These dividers show a high bias towards classifying most strokes as text. The Microsoft divider correctly classifies 100% of text in the test dataset and only 37.17% of shapes. Entropy correctly classifies 97.13% of text and only 41.59% of shape strokes. All other dividers are not significantly different from the domain-specific models in their performance on shape strokes.

These results show that domain-specific dividers for the to-do list are more successful at classifying this type of document than the general dividers.

	# Participants	# Text Strokes	# Shape Strokes	Total # Strokes	% Text : % Shape Strokes
Train	10	874	88	962	91:8
Test	10	836	113	949	88:12
Total	20	1710	201	1911	89:11

Table 51 Summary Statistics for To-do List Training and Testing Datasets

Dividers	% Text Correct	% Shapes Correct	% Total Correct
Train: Domain-specific Dividers			
LADTree 1 (to-do list)	99.54	74.73	97.21
LogitBoost (to-do list)	98.86	81.32	97.21
Test: Domain-specific Dividers			
LADTree 1 (to-do list)	98.69	70.09	95.18
LogitBoost (to-do list)	98.81	72.65	95.60
Test: General Dividers			
LADTree 1	93.78	65.49	90.41
LogitBoost	94.26	65.49	90.83
Divider 2007	94.26	73.45	91.78
Entropy	97.13	41.59	90.52
Microsoft	100.00	37.17	92.52

Table 52 Results of Dividers on To-do List Dataset with Domain-specific Dividers

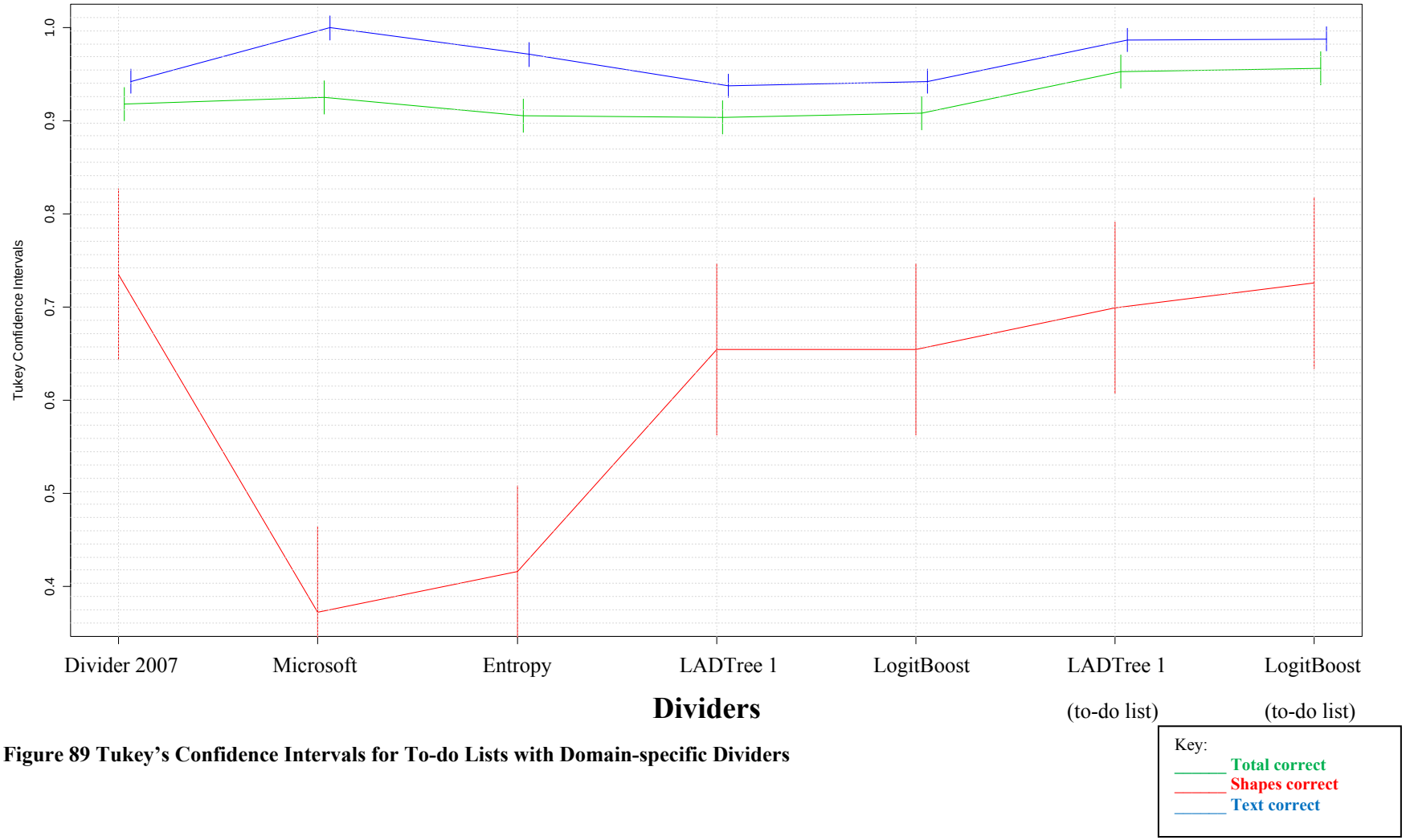


Figure 89 Tukey's Confidence Intervals for To-do Lists with Domain-specific Dividers

7.4.2 Euler Diagrams

Domain-specific dividers for Euler diagrams were trained using data from the first five participants in the dataset. Data from the other five participants were used for testing the domain-specific dividers as well as the original general dividers, LADTree 1 and LogitBoost, and the three existing dividers. A summary of the datasets used for training and testing is in Table 53. Table 54 shows the results for training and testing with Euler diagrams on these dividers.

	# Participants	# Text Strokes	# Shape Strokes	Total # Strokes	% Text : % Shape Strokes
Train	5	189	156	345	55:45
Test	5	224	178	402	56:44
Total	10	413	334	747	55:45

Table 53 Summary Statistics for Euler Diagram Training and Testing Datasets

Dividers	% Text Correct	% Shapes Correct	% Total Correct
Train: Domain-specific Dividers			
LADTree 1	99.47	100	99.71
LogitBoost	97.37	99.36	98.27
Test: Domain-specific Dividers			
LADTree 1	99.53	94.22	97.14
LogitBoost	97.64	90.17	94.29
Test: General Dividers			
LADTree 1	87.50	88.76	88.06
LogitBoost	81.25	95.51	87.56
Divider 2007	51.79	89.89	68.66
Entropy	98.66	92.70	96.02
Microsoft	76.34	75.28	75.87

Table 54 Results of Dividers on Euler Diagram Dataset with Domain-specific Dividers

The Euler diagram trained LADTree 1 has a training accuracy of 99.71% and testing accuracy of 97.14% overall. The domain-specific LogitBoost model has a training accuracy of 98.27% and a test accuracy of 94.29% overall. The general dividers' accuracy on the test set ranges from 68.66% to 96.02%.

Tukey's confidence intervals for the test results are shown in Figure 90. The confidence intervals for the total percentage of strokes correct show that the domain-specific dividers are significantly more accurate than the original general dividers, LADTree 1 and LogitBoost, and also existing dividers, Microsoft and Divider 2007. There is no significant difference between Entropy and the domain-specific dividers. Entropy correctly classifies 96.02% of the Euler diagram test set.

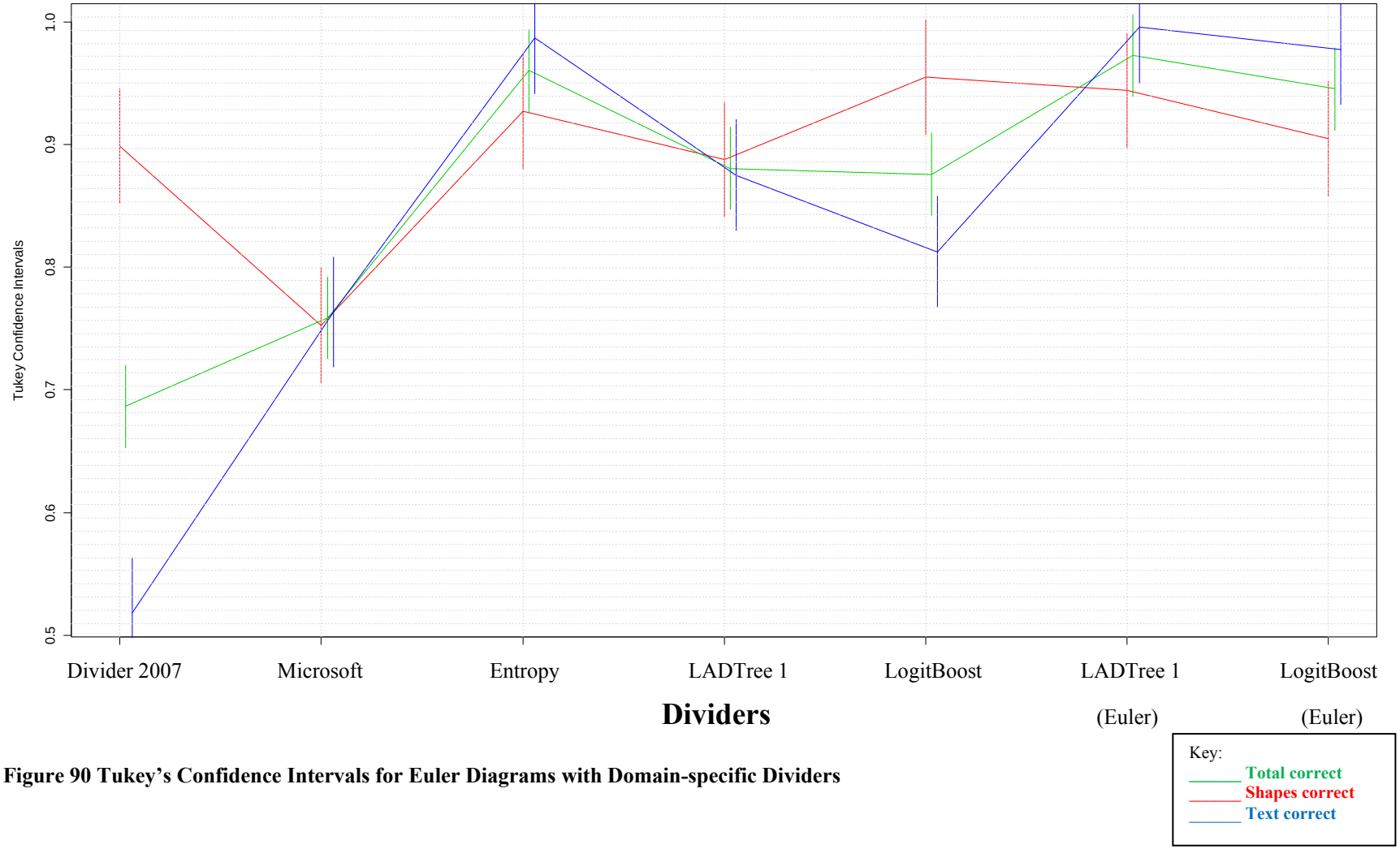


Figure 90 Tukey's Confidence Intervals for Euler Diagrams with Domain-specific Dividers

A similar pattern appears for the classification rates of text strokes. The domain-specific dividers are significantly more accurate than all dividers except for Entropy. For shapes, there is no significant difference between all dividers except for the Microsoft divider, which is significantly less accurate at classifying shape strokes than all others.

Overall, the domain-specific dividers for Euler diagram are not significantly different to Entropy in terms of classification accuracy although the domain-specific LADTree 1 has the highest classification accuracy of all dividers. The Euler diagram dataset is one of the smallest test datasets: with more training and testing examples the results may be better. More test data would certainly result in narrower confidence intervals, which may influence the differences found between dividers. This must be left to future work.

7.4.3 COA Symbols

LADTree 1 and LogitBoost were trained on one half of the COA symbols dataset to build domain-specific dividers. The other half of the dataset was used for testing these dividers along with the original general dividers for LADTree 1 and LogitBoost, and the three existing dividers. Further information on the training and testing datasets is given in Table 55. The dataset could not be separated by participants, as for the to-do list and Euler diagram datasets, because this dataset originated from another research group, so it had to be converted to the DataManager format which resulted in some participant information being lost. The results for training and testing using these datasets are shown in Table 56.

	# Text Strokes	# Shape Strokes	Total # Strokes	% Text : % Shape Strokes
Train	273	93	366	75:25
Test	243	121	364	67:33
Total	516	214	730	71:29

Table 55 Summary Statistics for COA Symbols Training and Testing Datasets

Dividers	% Text Correct	% Shapes Correct	% Total Correct
Train: Domain-specific Dividers			
LADTree 1	99.63	95.7	98.63
LogitBoost	98.90	88.17	96.17
Test: Domain-specific Dividers			
LADTree 1	99.59	74.38	91.21
LogitBoost	99.59	77.69	92.31
Test: General Dividers			
LADTree 1	94.24	73.55	87.36
LogitBoost	90.12	76.03	85.44
Divider 2007	97.94	72.73	89.56
Entropy	96.30	69.42	87.36
Microsoft	100.00	33.06	77.75

Table 56 Results of Dividers on COA Symbols Dataset with Domain-specific Dividers

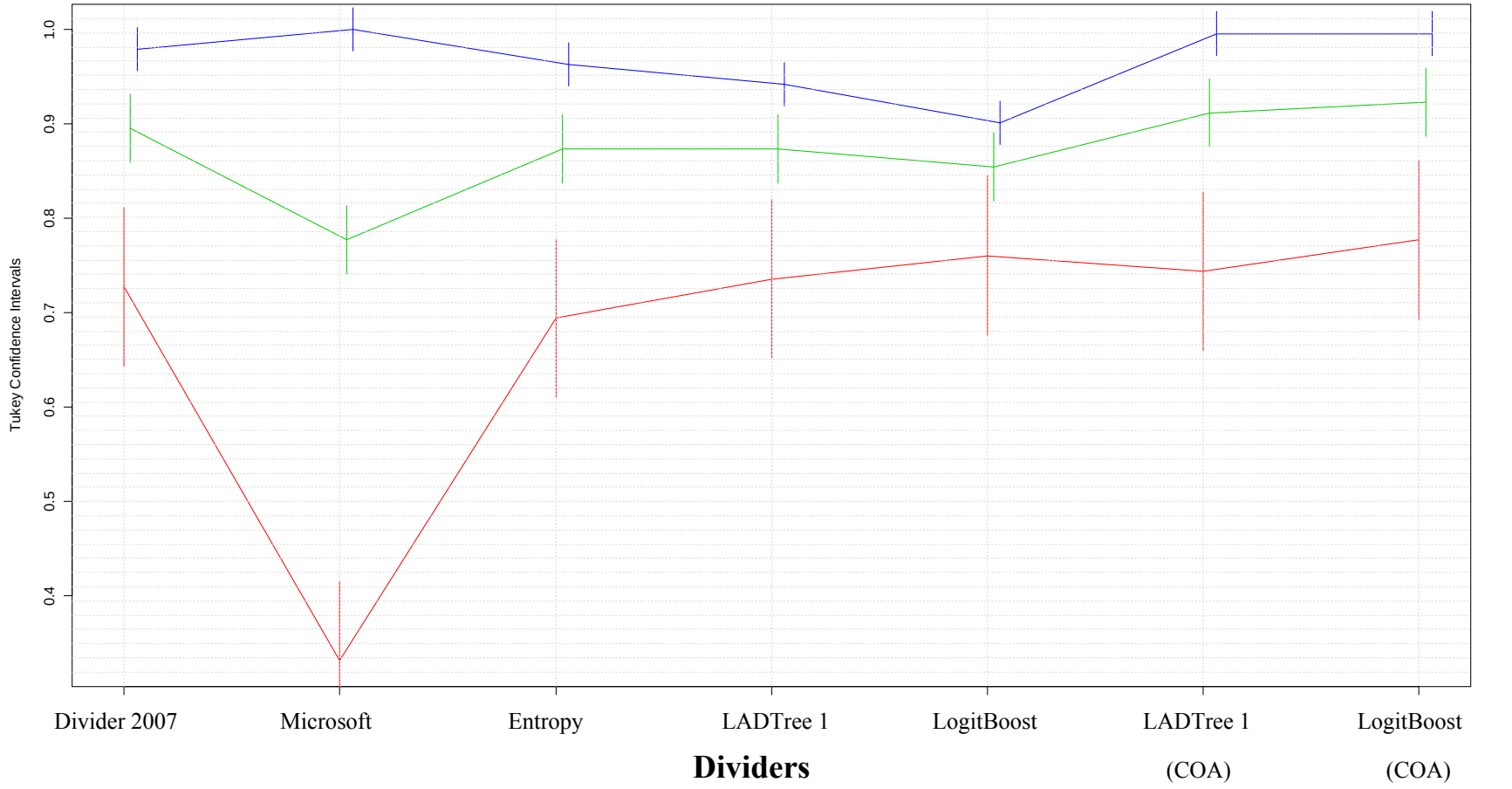


Figure 91 Tukey's Confidence Intervals for COA Symbols with Domain-specific Dividers

The COA specific LADTree 1 and LogitBoost models have training accuracies of 98.63% and 96.17%, and overall classification rates for the test dataset of 91.21% and 92.31% respectively. The general dividers' overall accuracies lie in the range of 77.75% and 89.56%.

Figure 91 shows the Tukey's confidence intervals for these results on the test dataset. The overall confidence intervals show that the domain-specific LogitBoost model is significantly more accurate than the Microsoft and general LogitBoost dividers. The domain-specific LADTree is significantly more accurate than the Microsoft divider. The domain-specific dividers are not significantly different from the other dividers overall.

In terms of the classification of text, the domain-specific dividers are significantly more accurate than the general LADTree 1 and LogitBoost dividers; they are not significantly different to all other dividers for text classification. For shape classification, the Microsoft divider is significantly less accurate than all others. There is no significant difference in shape classification accuracy between the remaining dividers.

Overall, the use of domain-specific dividers for COA symbols does not have a significant effect on classification accuracy although they produce the highest correct classification rates on the test set. This is the smallest test dataset, which is evident by the fact that the confidence intervals are very wide - resulting in less significant differences being found. With more test data, the results may improve.

7.4.4 Logic Diagrams

Domain-specific dividers for logic diagrams were built by training LADTree 1 and LogitBoost with half of the logic diagram dataset. The other half of the dataset was used for testing the domain-specific dividers as well as the original, general models and the three existing dividers. The training and test sets could not be separated by participants as this information was lost when converting this dataset to DataManager's format. A summary of the training and testing datasets is in Table 57. This dataset is especially interesting as it contains real world examples collected from student coursework. The results for training and testing using these datasets are shown in Table 58.

The domain-specific LADTree 1 model for logic diagrams had a training accuracy of 97.90% and testing accuracy of 91.80% overall. The domain-specific LogitBoost model has a training accuracy of 96.70% and a test accuracy of 90.73% overall. The general dividers' accuracy on the test set ranges from 42.45% to 80.34%.

Tukey's confidence intervals, shown in Figure 92, show that the domain-specific dividers are significantly more accurate than all others tested on the logic diagram test set overall. Entropy is significantly less accurate than all other dividers overall on the logic diagram test set.

In terms of text classification rates, Entropy is significantly more accurate than all other dividers. The domain-specific dividers are significantly more accurate at classifying text than the original general dividers (LogitBoost and LADTree1) and not significantly different to Divider 2007. The domain-specific LADTree 1 is also significantly more accurate then the Microsoft divider when classifying text; whereas the domain-specific LogitBoost model is not significantly different to the Microsoft divider in this case.

For shape classification, the domain-specific dividers are significantly more accurate than all others, except for the general LogitBoost model, where there is no significant difference. Entropy is significantly less accurate at shape classification for the logic diagram dataset than all other dividers.

In summary, use of domain-specific dividers for logic diagrams produces significantly more accurate results than general dividers. As this is a large dataset, the confidence intervals are narrower than the previous cases, allowing us to detect significant differences reliably.

	# Text Strokes	# Shape Strokes	Total # Strokes	% Text : % Shape Strokes
Train	1149	3271	4420	26:74
Test	1147	3049	4196	27:73
Total	2296	6320	8616	27:73

Table 57 Summary Statistics for Logic Diagrams Training and Testing Datasets

Dividers	% Text Correct	% Shapes Correct	% Total Correct
Train: Domain-specific Dividers			
LADTree 1	94.78	98.99	97.90
LogitBoost	93.04	97.98	96.70
Test: Domain-specific Dividers			
LADTree 1	90.67	92.23	91.80
LogitBoost	89.97	91.01	90.73
Test: General Dividers			
LADTree 1	78.03	76.39	76.84
LogitBoost	52.31	90.88	80.34
Divider 2007	89.54	34.50	49.55
Entropy	98.17	21.48	42.45
Microsoft	86.49	60.61	67.68

Table 58 Results of Dividers on Logic Diagrams Dataset with Domain-specific Dividers

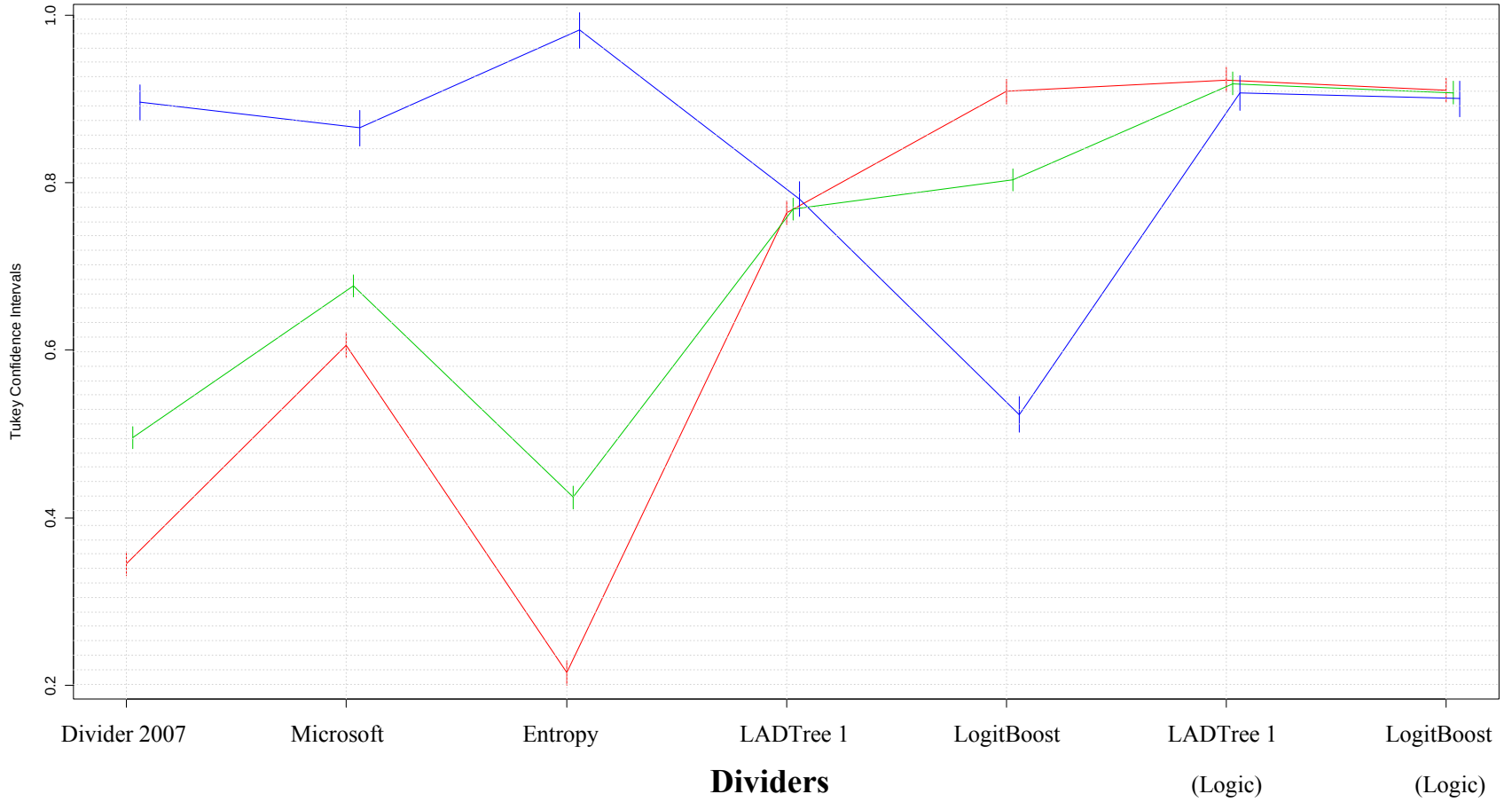


Figure 92 Tukey's Confidence Intervals for Logic Diagrams with Domain-specific Dividers

7.5 Summary

The top five new dividers were evaluated against three existing dividers, Entropy (Bhat et al. 2009), Divider 2007 (Patel 2007) and the Microsoft divider (Microsoft Corporation 2008). Datasets from six different domains were used in the evaluation. We collected mind-maps, to-do lists and UML class diagrams ourselves and obtained Euler diagrams, COA symbols and logic diagrams from other sketch recognition researchers. DataManager's Evaluator (Schmieder 2009; Schmieder et al. 2009) was used to compare the accuracy of these dividers on each dataset.

The results over all datasets show that LogitBoost and LADTree 1 are the most accurate of all dividers, followed closely by LADTree 2. They are significantly more accurate than the three existing dividers. The results for each dataset are mixed. The UML class diagram and mind-map datasets are the best classified by most dividers. However, the remaining datasets have much higher misclassifications.

An investigation into the use of domain-specific dividers was carried out for to-do lists, Euler diagrams, COA symbols and logic diagrams. Domain-specific dividers for to-do lists and logic diagrams are significantly better than all other general dividers tested. For the Euler diagram dataset, the domain-specific dividers are significantly more accurate than all dividers except for Entropy, where no significant difference was found. The domain-specific dividers on the COA diagram dataset have the highest classification rates overall but they are not significantly different to most of the general dividers. With the exception of the logic diagrams and to-do lists, the small datasets make it difficult to detect significant differences. These results are discussed further in the next chapter.

Chapter 8

Discussion

This research has investigated using data mining for sketched diagram recognition. In particular, we have focused on the division of writing and drawing, but many of the techniques and tools can be applied to other associated recognition problems. Our goal was to develop more accurate recognisers, using text-shape dividers as an exemplar. Data mining techniques were employed to build several new dividers. Our evaluation of these new dividers shows that the use of data mining in this context produces dividers that are significantly more accurate than three existing dividers.

There is a large amount of work already in the area of sketch recognition but the division of shapes and text has received little attention. A small number of general text-shape dividers have been developed (Bishop et al. 2004; Patel 2007; Microsoft Corporation 2008; Rodríguez et al. 2008; Avola et al. 2009; Bhat et al. 2009) using various feature sets and algorithms. However, to our knowledge, no systematic study of algorithms has been carried out and a comprehensive feature library such as our own has not been employed for this task.

Considering diagram and document recognition from a wider perspective, there are three possible approaches: bottom-up, top-down or a combination of both. A bottom-up approach begins the recognition process at the primitive stroke level. This is typically followed by a progressive joining of strokes into larger and more complex groups, thus developing an overall semantic understanding of the diagram. On the other hand, a top-down system starts with a high-level analysis of the structure and uses this information to aid recognition of the composite parts. There can also be hybrid approaches that combine both bottom-up and top-down methods by considering primitives and overall layout together to try to resolve ambiguities.

Thus far, our approach has been bottom-up, using feature-based recognition. A top-down approach is not suitable for a general divider as it requires knowledge of domain-specific semantic structures that are not

available in a general diagramming setting. Feature-based recognition is used as we believe it is the most ideal method for the text-shape divider problem. As discussed in Chapter 2, other methods such as template matching, textual descriptions and hard-coded recognisers are not suitable for division due to the large variation in the classes of text and shapes and the cumbersome nature of these techniques. For text-shape division, we require some way of measuring key characteristics of digital ink in order to make comparisons between classes. We believe feature-based recognition is the best way to achieve this. Feature-based recognition has also been used for previous text-shape dividers (Bishop et al. 2004; Patel et al. 2007; Bhat et al. 2009).

The following is a discussion of the tools we developed, techniques used and results obtained through our development of text-shape dividers in the wider context of knowledge in this field.

8.1 Features and Feature Search

Features are a foundation of recognition algorithms. Early work in feature-based recognition was conducted by Rubine (1991) who developed a feature-based gesture recogniser. The feature set he developed, based on 13 features, has since been used in numerous other sketch recognisers (Landay et al. 1995; Damm et al. 2000; Lin et al. 2000; Long et al. 2000; Chen et al. 2003; Plimmer et al. 2003b; Bickerstaffe et al. 2007; Freeman et al. 2007; Patel et al. 2007; Zhang et al. 2007; Paulson et al. 2008b; Meyer et al. 2009; Willems et al. 2009). Others have used feature-based recognition with their own feature sets and different algorithms (Machii et al. 1993; Fonseca et al. 2001; Sezgin et al. 2001; Calhoun et al. 2002; Fonseca et al. 2002; Hammond et al. 2002; Bishop et al. 2004; Qin 2005; Paulson et al. 2008a; Bhat et al. 2009).

However, there is little evidence of in depth feature searches conducted and no comprehensive libraries available except for recent work by Willems et al (2009). Their feature set is built on a base set of 48 features with various extensions applied to this base set. The base set is a similar size to the feature set used to develop our previous divider, Divider 2007 (Patel 2007).

When developing Divider 2007 (Patel 2007), we found that the simple feature of bounding box width can correctly classify approximately 85% of the training dataset used for that study. With the addition of seven other features in a decision tree, to form Divider 2007, this divider is able to classify approximately 79% of our test dataset (according to the simple average shown in Table 39). The Entropy divider uses a single

feature for classification, although this feature is much more complex than the width of a strokes bounding box. The Entropy feature is able to correctly classify approximately 83% of our test dataset (according to the simple average shown in Table 39). These results show that single features, or a small group of features, are able to classify 79-85% of strokes with ease, but getting beyond this level of classification is difficult.

To improve on previous dividers, the first step in this research was to compile a comprehensive feature library, including features from our previous feature set (Patel 2007), from related work, and some of our own new additions. Our feature set is comprised of 114 features measuring aspects of stroke curvature, density, size, temporal context, spatial context, divider results, direction, time/speed, intersections and pressure. In our second round of analyses, additional features were included in the feature library to focus on identifying commonly misclassified connectors. However, our evaluation showed that using these additional features had no significant effect on divider accuracy. We believe that it is unlikely that more features will have a significant effect on the performance of general text-shape dividers. As mentioned above, recent work by Willems et al (2009) describes a feature set for multi-stroke gestures composed of 758 features. These features come from a base of 48 features and then add variations of this base set which inflate the feature set size. Many of their features are already represented in our library; others that are not already present could be added at a later date. However, we do not expect these additions to make a statistically significant difference to text-shape divider accuracy. This assessment is based on the results of our second round analyses and the comprehensive range of features already present in our library. In addition, the use of more features requires greater computation time.

The use of feature selection can assist in identifying significant features, thus reducing a feature set size by eliminating non-contributing features. We investigated feature selection as part of our analysis: a discussion of our results is in Section 8.3.2.

Our evaluation results show that the current feature library describes the text-shape divider problem space well. Even for unique domains where no prior training was given, such as logic diagrams, the new dividers significantly outperform the existing dividers. This shows that the features can be applied successfully to new domains. With domain-specific training, using an appropriate algorithm, the feature library produces even better results.

The features found in the top levels of the LADTree with 1500 iterations were described in Section 6.2.8. It was found that these features come from the following categories of our taxonomy: divider results, size, spatial context and curvature, with most of these top level features coming from the last two categories. Regarding these features, the main differences, in comparison to other dividers, are the use of divider results and spatial context. Bishop et al's divider (2004) uses features measuring curvature, direction, size and temporal context. Divider 2007's (Patel et al. 2007) features include measures of curvature, density, size and temporal context. Bhat et al's divider (2009) relies on a single feature of density. The main features in Divider 2007 and Bishop's divider measure aspects of temporal context. We believe the use of temporal context may not be as reliable as spatial context, used by our LADTree, for sketched diagram recognition due to the problem of interspersed strokes as identified by Sezgin et al (2007). Further, the use of previous divider results as a pre-parse of the data provides valuable information to our LADTree.

The feature library is not limited to the text-shape divider problem. It has been used as an input to other sketch recognition problems such as developing basic shape recognisers (Chang 2010) and in an application for sketching Euler diagrams (Delaney et al. 2010). Chang's best single stroke basic shape recogniser has a recognition rate of 98.0% compared with the best of existing basic shape recognisers that can correctly classify only 89.7% on the same dataset. This demonstrates that the value of this feature library extends beyond text-shape division.

A taxonomy was also constructed to categorise the feature library; the taxonomy is described in Chapter 4. The taxonomy helps us to gain a better understanding of the information we can obtain from ink by providing a clear overview of various characteristics of ink which the features measure. When we examine the subsets chosen by feature selection methods, having a taxonomy to refer to is a valuable tool. Grouping the feature subsets using the taxonomy helps us to gain a more intuitive understanding of the features that are significant to the problem, rather than overwhelming us with each individual feature's characteristics.

8.2 Data Collection and DataManager

The data collection process was assisted by DataManager, a tool developed as a part of this project for collecting and labelling sketches and automatically generating datasets. Using DataManager for these tasks made the data collection process fast and efficient and ensured that tasks could be easily repeated. The sketching community needs tools like this as a basis for developing benchmarking datasets. Schmieder's (2009; 2009) extension to DataManager, adding an evaluation platform for running comparative studies

across multiple recognisers, enhances DataManager's potential to be used as a benchmarking tool for developing sketch recognisers.

Other data collection tools for digital sketches exist, but none are able to perform all the functions available within DataManager. Many are restricted to the collection of isolated components rather than full diagrams (Bickerstaffe et al. 2007; Signer et al. 2007a; Signer et al. 2007b; Avola et al. 2008; Meyer et al. 2009). Recent studies have found that recognition rates of algorithms trained with full diagrams are better than those trained with isolated components; given that they are meant for recognising full diagrams (Field et al. 2009; Schmieder et al. 2009). Of note also, in favour of full diagram collection, is the appearance of features measuring spatial context in the top level of the LADTree. In addition, some tools are limited to one task (Wolin et al. 2007; Paulson et al. 2008c; Johnson 2009; Johnson et al. 2009a; Kaster et al. 2009) or a particular diagram domain (MacLean et al. 2009).

GestureLab (Bickerstaffe et al. 2007; Meyer et al. 2009) is one of the few tools with a built in feature set. They use their data in conjunction with the feature set to train an SVM algorithm and produce domain-specific recognisers. However, this feature set is limited to Rubine's 13 features (1991). In addition, they provide no functions for automatically generating datasets so that other algorithms may be explored using external data mining tools such as Weka. This limits researchers to the use of SVM, or requires additional algorithms to be built into the tool. Even were this done, tools like Weka have specialised functionalities that assist in data mining, such as data visualisation and parallelisation of algorithms for training on remote machines.

Labels are required for data when employing supervised machine learning techniques, as we have done in this work. Designing the labelling interface in DataManager was a difficult task. The divider algorithms work on a stroke by stroke basis, so strokes require individual labels. Strokes can also have more than one label: for example, a stroke may be labelled as a "shape" and a "line". DataManager has also been used for developing and evaluating basic shape recognisers (Schmieder 2009; Schmieder et al. 2009; Chang 2010; Chang et al. 2010). In this case, multi-stroke labelling is required where one label is applied to a group of strokes that make up a basic shape. For example, a rectangle drawn in four strokes has one label that groups the four strokes together into the basic shape that they represent. DataManager was extended to provide a multi-stroke labelling option for this case (Blagojevic et al. 2009). However, this function is independent of the original labelling method. Combining individual and multi-stroke labelling while still

providing the option of applying more than one label to each stroke in an easy to use, flexible and efficient interface is an unresolved issue.

DataManager's value extends beyond the development of text-shape dividers. To date, DataManager has been used and extended by several projects including: basic shape recogniser development with the extension of an interface for automatically generating recognisers (Chang 2010; Chang et al. 2010); evaluation of basic shape recognisers with the extension of an evaluation platform (Schmieder 2009; Schmieder et al. 2009), and as a data collection tool in the development of an application for hand-drawn Euler diagrams (Delaney et al. 2010). Data collected using DataManager has also been used in an evaluation study by Field et al (2009). The use and extension of DataManager in these projects demonstrate the wide ranging potential of the tool.

8.3 Data Analysis

While there have been other studies comparing the use of different algorithms for sketch recognition, to our knowledge, none have been as thorough or systematic as ours. In the area of text-shape division, studies have been conducted exploring two to three algorithms or variants of algorithms (Bishop et al. 2004; Waranusast et al. 2009) but without the use of comprehensive feature sets. More work has been done in shape recognition. Willems et al (2009) compiled a larger feature set and compared three algorithms for shape recognition. However, their study was focused on exploring the feature set rather than a comparison of algorithms. Zhang et al (2007) ran a comparative study using Support Vector Machines, Hidden Markov Models and a Bayesian Belief Network for multi-stroke shape recognition. More recent work (Chang 2010; Tumen et al. 2010) has explored the use of a larger number of algorithms as well as feature selection and ensembles. However, this work is all in the domain of shape recognition.

For text-shape division in particular, a range of algorithms has been employed in the past, including: Decision Trees (Shilman et al. 2004; Patel 2007; Zeleznik et al. 2008); Neural Networks (Mochida et al. 2003; Bishop et al. 2004; Mochida et al. 2004); linear classifiers (Plimmer et al. 2007; Rodríguez et al. 2008); Support Vector Machines (Ao et al. 2006; Waranusast et al. 2009); K-Nearest Neighbour (Waranusast et al. 2009); Boosting (Shilman et al. 2004; Peterson et al. 2010), and Hidden Markov Models (Bishop et al. 2004). Our analysis has included an investigation of all these types of algorithms, except Hidden Markov Models used by Bishop et al (2004) and the linear classifiers used by Plimmer et al (2007)and Rodríguez et al (2008).

In regards to the linear classifiers, from our previous evaluation of Divider 2007 (Patel 2007) we know that this divider performs better than the InkKit divider (Plimmer et al. 2007) which uses Rubine's (1991) linear classifier. Therefore, we believe that this linear classifier would not be better than our new dividers which are significantly more accurate than Divider 2007; although we would have to test this algorithm with our own feature set. In regards to the Fisher linear discriminant analysis used by Rodríguez et al (2008), their own evaluations show it to be less accurate than a handwriting recogniser. These results do not give us much confidence in such linear classifiers. Hidden Markov Models used by Bishop et al (2004) on the other hand, is a potential algorithm to investigate in future work.

To our knowledge, the most successful algorithms we found, LogitBoost and LADTree, have not previously been used for text-shape division. The identification of these algorithms shows the value of our systematic study of algorithms rather than focusing on those already used in this area.

8.3.1 Preliminary Analysis and Classifier Tuning

Our data analysis began with a preliminary investigation of a wide range of data mining algorithms from Weka. This was a valuable exploratory exercise that helped us gain a feel for the correct classification rates of each classifier with basic default parameters on the training data. Some of the algorithms used previously for text-shape division, such as K-Nearest Neighbour, AdaBoost and the Decision Tree used by (Patel 2007; Zeleznik et al. 2008; Peterson et al. 2010) did not perform well enough in our preliminary analysis to warrant further investigation.

The next step involved tuning the parameters of seven classifiers chosen from the preliminary analysis to find optimal settings. This step was time consuming but made a significant difference to the correct classification rates obtained from LogitBoost, LADTree and SMO. Considering the overall success of the tuned LogitBoost and LADTree classifiers, this step in the analysis had a significant effect on text-shape divider accuracy.

Varying the parameter settings for Bagging and RandomForest made no significant difference to classification rates. RandomForest is essentially Bagging using a RandomTree as the base learner, so these techniques are very similar. They involve the construction of multiple trees and use a voting system to make the final predictions. The default setting in Weka is to construct ten trees. We varied this setting between 10 and 5000 trees for these classifiers. However, given that there was no significant difference

found when the number of trees was varied, we believe that the variation represented in ten trees is sufficient for classifying a wide range of data when using these classifiers.

8.3.2 Feature Selection

Three methods of feature selection were used to try to improve on the results of the previous step in the analysis. None of the methods used made a significant improvement on the classification rates achieved prior to feature selection. There are several possible reasons for this lack of improvement. It is possible that all the features in the feature library make significant contributions to recognition accuracy. In this case, reducing the feature set can be detrimental to classification rates. Also, because of long classifier training times when using feature selection, the parameter configurations for the classifiers were set to those with the fast training times rather than the highest accuracy. However, the accuracy of the classifiers with the chosen parameter configurations is not significantly different to the classifiers with the highest classification rates based on the training data. The one exception was for LADTree which had a considerably longer training time when a high number of iterations was used. Therefore, a trade-off had to be made between training time and accuracy. Although these parameter configurations produced classification rates that were not significantly different to the highest models accuracy on the training data, they still may have had a negative effect on the accuracy obtained by feature selection with these classifiers.

Chang (2010) observed similar results for feature selection in basic shape recognisers where no significant improvement was able to be made over the original classifiers. Chang stated that classifiers that already use an inner voting strategy require variation in data to be successful: eliminating features reduces variation and therefore can result in poor results for feature selection. In addition, Chang stated that classifiers with tree structures may not benefit from feature selection as they already use base splits in the tree on the most valuable features. Some trees also have pruning mechanisms that eliminate low level features from the tree to avoid over-fitting. Therefore feature selection can be redundant and even detrimental if good features are not retained. Five of the seven classifiers explored in our analysis use voting strategies or tree structures. It is possible that no significant improvements were made to these classifiers as either the variation was lost by eliminating features or good features were not retained.

Chang also conducted a study (Blagojevic et al. 2010) looking at the effect of feature selection on Rubine's gesture recogniser (1991), a linear classifier. The use of feature selection in this context produces significantly more accurate results than the original Rubine's algorithm (1991), InkRubine (Plimmer et al.

2007) and $1 (Wobbrock et al. 2007). These results show that feature selection can be beneficial in combination with simpler classifiers. The classifiers used in this research are more sophisticated than Rubine's linear classifier (1991) and in most cases utilise voting strategies or tree structures to ensure that the features are used wisely.

An investigation into the effect of aliases in the feature set would be an interesting addition to this work. Aliases exist when two or more features measure the same characteristic but in varying ways. If aliases are very similar in nature, they effectively increase the weighting of that characteristic in algorithms like Rubine's linear classifier (1991) regardless of their actual importance to the recognition problem. One approach to such an investigation is to reduce the feature set by eliminating known aliases and re-test the classifiers using this reduced feature set. The optimal number of features may be lower than the current feature library. For example, we have observed from our feature selection results that feature sets with 50 features produce results that are not significantly different from using the entire feature set, although this is algorithm dependent. Tumen et al (2010) observed that optimal feature set size is dependent on the domain of use. The components in some domains can be more easily distinguished and thus require less information for recognition than others do.

8.3.3 Ensembles

Various classifier ensemble combinations were also investigated to identify if further improvements could be made to classification rates. The use of ensembles did not result in significant increases in accuracy. However, the classification rates of the ensembles tested in the evaluation were higher than all new dividers for four of the six datasets. It is possible that with the addition of more data, the ensemble results may be significantly more accurate; the width of Tukey's confidence intervals is influenced by dataset size.

A possible reason for the performance of the ensemble here may be that four of the seven classifiers investigated, RandomForest (Breiman 2001), Bagging (Breiman 1996), LADTree (Holmes et al. 2002) and LogitBoost (Friedman et al. 2000), already use inner voting mechanisms to make predictions. Therefore, using these in outer ensemble combinations may have no significant effect on the accuracy level already obtained.

As with feature selection, because of long classifier training times, the parameter configurations for the classifiers within the ensembles were set to those with fast training times. These configurations still produce results that are not significantly different to the most accurate models obtained for the particular classifier on the training data, with the exception of the LADTree. However, they may still have had a negative effect on the accuracy obtained by the ensembles. Although no significant difference was found when comparing the results of classifiers with different parameter configurations, the differences may not be apparent due to the size of the datasets used.

Others (Chang 2010; Tumen et al. 2010) have also used ensembles for developing shape recognisers for sketched diagrams. Ensembles were found to produce significantly better results than single classifiers and existing basic shape recognisers in these studies. The main difference between text-shape division and basic shape recognition is the number of classes. For Chang's study (2010), the number of classes in each test dataset range from three to six basic shapes. Schmieder's study (2009) of basic shape recognisers found that recognisers generally performed better on data with a smaller number of classes. In our case, it is possible that single classifiers are able to perform just as well as ensembles because there are only two classes. But the within class variation for the text and shape classes is far greater than what is usual for shape classifiers. For basic shape recognition, single classifiers may not be able to accurately distinguish between three or more classes as well as an ensemble of classifiers. This may explain the difference in results when using ensembles for dividers and basic shapes.

8.3.4 Second Round Analysis

The final part of the analysis involved a second round investigation into features that could help correct the common misclassification of connectors. Connectors, particularly arrowheads, are known to be difficult to recognise. We used variations on features developed for connectors in past research (Kara et al. 2004; Freeman et al. 2007; Willems et al. 2008) to try to solve this problem. We added three features to the feature library. The results showed that classifiers trained with the extended feature set were not significantly different to classifiers trained with the original feature set. As mentioned earlier, the original feature set is quite comprehensive and we believe that further additions are unlikely to make statistically significant differences.

A second parse feature was also added, specifically to find arrowheads. The preliminary evaluation showed that adding the second parse feature increased the percentage of correctly classified arrows. However, the differences were not statistically significant and this increase came at a cost of

misclassifying more text. The dataset used here was composed of 68% text and 5% arrows. Therefore, overall results of using the second parse feature were significantly less accurate than when the second parse feature was not used. How this step can be further improved without the high cost of misclassifying text is an unresolved issue.

In the final evaluation, LADTree 2, the model trained with the extended feature set and using the second parse feature, ranked third of a total of eight dividers. This ranking is above all existing dividers and the new divider ensembles. The results of LADTree 2 on domains with connectors consistently show higher classification rates for shapes than LADTree 1, the model without the second round features. However, the differences are not statistically significant.

8.4 Evaluation

The top five new dividers were evaluated against three existing dividers. A ranking based on the overall evaluation results ranked all the new dividers above the existing dividers, with the exception of Vote 1, which was ranked below the Microsoft divider. The differences between these two dividers are as follows. Based on Tukey's confidence interval of the overall weighted average for accuracy, Vote 1 is significantly less accurate than the Microsoft divider. Vote 1 is also significantly less accurate than Vote 2, whereas the Microsoft divider is not significantly different to Vote 2 according to the weighted average accuracy. Vote 1's main weakness is the logic diagram dataset, where it correctly classifies 50.35% of the dataset in comparison to 70.83% for the Microsoft divider.

The overall weighted average takes dataset size into account. The logic diagram dataset is the largest of all test datasets with a total of 8616 strokes. Therefore, the performance of dividers on the logic diagram dataset has a large bearing on the weighted average accuracy results. As Vote 1's performance is low on logic diagrams, its overall weighted average accuracy suffers. The simple average accuracy, on the other hand, weights all datasets equally. Vote 1 and the Microsoft divider are not significantly different if only the simple average is used for comparison. The overall ranking based on the simple average is shown below, along with the original ranking. The general order of dividers is the same; the only differences are that LogitBoost and LADTree 1 share first place and Vote 1 and the Microsoft divider share fourth place. There is no pre-defined way of aggregating results of Tukey's confidence intervals, so we have based the ranking on both averages to provide us with as much information as possible.

Original Ranking (based on both averages)

1. LogitBoost
2. LADTree 1
3. LADTree 2
4. Vote 2
5. Microsoft
6. Vote 1
7. Entropy
8. Divider 2007

Ranking (based on simple average only)

1. LogitBoost, LADTree 1
2. LADTree 2
3. Vote 2
4. Microsoft, Vote 1
5. Entropy
6. Divider 2007

8.4.1 The Effect of Class Proportions and Diagram Structure

The proportion of each class has an effect on the overall accuracy as the classification of text is generally better than shapes for most test datasets. The training dataset for the new dividers and Entropy is 77% text; therefore more training examples for text are provided than shapes. The Euler diagram dataset is an exception, as shapes are more highly classified than text in general. The Euler diagram dataset is composed of ellipses, circles and text and is the most balanced of all test datasets with 55% text. Ellipses and circles have high classification rates as we have seen with tests done using the verification dataset in Table 28 and the high classification rates for containers in the mind-map dataset (see Table 42). The ellipses and circles in the Euler diagrams are no different to other domains except that they frequently intersect.

Text in the Euler diagrams, on the other hand, is single isolated letters. These are harder to identify than letters that are part of words, due to differing characteristics related to spatial and temporal context. The training dataset has examples of single isolated letters in directed graphs but they are always contained within shape strokes, unlike text in Euler diagrams. These differences in the nature of text in Euler diagram may explain why shapes are more highly classified than text. The domain-specific dividers, trained with Euler diagram examples, have higher correct classification rates for text than shapes. The text classification rates for the domain-specific dividers are significantly higher than their general divider counterparts. These results show that training with examples of single isolated text make a significant difference to correct text classification rates for Euler diagrams.

We have observed that the Microsoft divider and Entropy, in particular, have large biases towards classifying strokes as text rather than shapes. This results in very low correct classification rates for shapes but very high correct classification rates for text. As most datasets in the test set have a larger ratio of text to shapes, high correct classification rates of text have a positive influence on overall classification rates despite the bias. A similar bias was also found when evaluating an older version of the Microsoft divider

(Microsoft Corporation 2005) in previous work (Patel 2007). However the source of this bias is unknown as there is no documentation describing the inner workings of the Microsoft dividers. We believe that these dividers are built more for documents that have much larger amounts of text than diagrams. Regarding the Entropy divider, Bhat et al (2009) also observed that their divider classifies text better than shapes, but they did not discuss this issue any further.

The results show a relationship between overall classification rates across all dividers and the ratio of shape to text strokes for each dataset. A ranking of the ratio of text to shapes (from highest to lowest percentage of text) in each dataset is shown in Table 59, alongside a ranking of the overall classification accuracy (from highest to lowest classification rates) of each dataset based on Tukey's confidence intervals in Figure 76. The rankings match exactly except for the placements of the to-do list and UML class diagrams. This provides further evidence to show that the dividers, on the whole, are more successful on datasets with higher ratios of text and also begins to answer the question of why some domains are easier to classify than others.

	% Text : % Shape Strokes	**Ranking of overall classification accuracy (from highest to lowest)**
To-do list	89:11	UML Class diagram
Mind-map	83:17	Mind-map
UML Class diagram	79:21	To-do list
Training data	***77:23***	
COA	71:29	COA
Euler	55:45	Euler
Logic	27:73	Logic

Table 59 Ratio of Text to Shapes in Datasets

Another possible reason why some domains are easier to classify than others relates to the structure of the test data in comparison to the training data. We set out to rigorously test the dividers in the evaluation study by using a variety of datasets, some of which were quite similar to the training data and others which are not diagrams but contain both shapes and text; for example the to-do list. The difference in ranking in Table 59 for the to-do list may be related to the fact that they are documents rather than diagrams. The main differences lie in the fact that to-do lists contain much higher amounts of text and have a different structure than the diagrams in the training dataset. In comparison to the training dataset, which was used to train all the new dividers and Entropy, the to-do list has 12% more text. In addition, to-do lists have more formally structured text written on baselines and very random shapes such as bullet points, dashes, stars and cross out lines. Mind-maps have the closest ratio of text to shapes as to-do lists; but the results for

dividers on this dataset are higher than to-do lists. The structure of a mind-map has text scattered more haphazardly but have more common shapes than to-do lists.

Logic and Euler diagrams are also prime examples of uniquely structured diagrams, very different to the training dataset, and therefore are not classified as well as other datasets by the general dividers. This is supported by the fact that the domain-specific logic and Euler dividers, only trained with examples from the particular domain, are significantly more accurate than their general divider counterparts. UML class diagrams have a very similar structure to organisation charts, which were included in the training dataset. They both have rectangles containing text and connectors between them. This shows that the dividers are more successful on domains that have similar characteristics to the training data. One direction for future work is to train a divider with examples from all the domains used in our training and test sets. Based on our observations discussed above, we believe this divider may be even more successful.

8.4.2 Domain-specific Dividers

The development of domain-specific dividers, although not directly part of the research goal, was an obvious solution for more accurately dividing unique datasets. One of the limiting factors was the size of the datasets available as each dataset had to be split into equally sized training and test sets. The domains with larger datasets available produced better results than those with smaller datasets. The to-do list and logic diagram datasets have 1911 and 8616 total instances respectively in comparison to 730 and 747 total instances for COA symbols and Euler diagrams respectively. As this investigation was not directly related to the overall research goal, we have left the collection of more data to future work.

The domain-specific dividers for to-do lists and logic diagrams were significantly more accurate than all other general dividers tested. The success of these domain-specific dividers show that the feature library in combination with LADTree and LogitBoost can cross over to divide documents and more complex diagrams that have unique semantic structures well. In addition, the logic diagram dataset is real world data collected from students' class work; the success of our dividers on such a dataset shows that their performance is consistent when testing on real examples.

The domain-specific Euler diagram dividers are significantly more accurate than all general dividers except for Entropy, where there is no significant difference. The original Entropy divider was shown to be very successful at distinguishing between text and circles (Bhat et al. 2009). An example of entropy values

for different shapes is shown in Figure 93. The value of Entropy for a circle is 0.342 which is vastly different from text at 12.205. The Euler diagram dataset is composed of circles, ellipses and text. The particular nature of these diagrams means that Entropy can easily distinguish between the shapes and text in Euler diagrams.

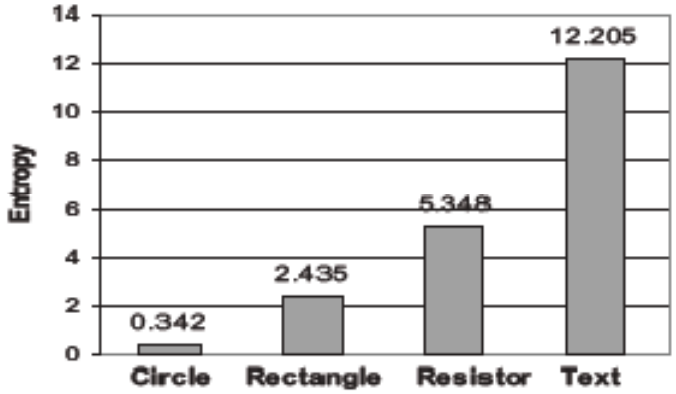


Figure 93 Entropy Values on Different Shapes (Bhat et al. 2009)[17].

The domain-specific dividers for COA symbols are not significantly different from the majority of the general dividers, but the classification rates for the domain-specific dividers are higher than all others. With a larger dataset, these differences may become significant as the width of Tukey's confidence intervals becomes smaller with more data. In addition to dataset size, the dividers for COA symbols may suffer from over-fitting. Although the classifiers are trained with domain-specific examples, the classifier parameters have been tuned with very different data. The original training set is composed of full diagrams as opposed to isolated components such as COA symbols. The parameters used with the LogitBoost and LADTree 1 domain-specific dividers may be more suitable for full diagrams. Temporal and spatial context features measure important information in full diagrams but may not add anything of value to isolated components such as COA symbols. A domain-specific divider for COA symbols trained without these features may produce better results.

The general divider results for COA and Euler diagrams show that Vote 1 and Vote 2 have higher classification rates on these datasets. Generating domain-specific dividers using the Vote 1 and Vote 2

[17] Image obtained from Bhat, A. and T. Hammond (2009). Using Entropy to Distinguish Shape Versus Text in Hand-Drawn Diagrams. International Joint Conference on Artificial Intelligence (IJCAI '09), Pasadena, California, USA.1395-1400. Reproduced with permission from the author.

classifiers may produce better results than LADTree and LogitBoost. These ensembles use simpler parameter configurations of the inner classifiers, due to long training times, than the classifiers tuned individually. For example, the LADTree within these ensembles uses 500 iterations as opposed to the 1500 iterations used for the individual classifier. These more simple parameters may result in models that do not suffer from as much over-fitting as the individual models.

Domain-specific dividers can be easily generated with Chang's extension to DataManager, RATA (Chang 2010). They are useful in applications when the diagram domain is known, such as in Delaney et al's (2010) Euler diagramming tool. For cases where the diagram domain is unknown, a general divider can be used that still produces accurate classifications.

We focused on accuracy over efficiency when developing recognisers here. This issue was briefly mentioned in Section 7.3.7 where the average time taken for recognition using the new and existing dividers was presented. The new dividers take a considerably longer time to perform recognition than the existing dividers. Currently, the new dividers use models that have been generated by Weka. The models are integrated into DataManager's evaluation platform using Weka libraries. Due to the design of these libraries, all the features that were used to train the model must be calculated prior to predicting the class of a new instance using the model, regardless of whether all the features are used for classification or not. Therefore all 114 (or 118 if second round features are used) features in the feature library must be calculated for each new instance to make classifications. This prolongs the recognition process unnecessarily. There are numerous ways in which the efficiency of these recognisers could be improved: they are described in Section 9.2.

Our feature library and DataManager have been invaluable tools in constructing good datasets. With the use of data mining to perform a systematic analysis of these datasets, we have been able to produce more accurate text-shape dividers. These tools and techniques are not limited to divider development but can also be applied to other sketch recognition problems.

Chapter 9 Conclusions and Future Research

We have conducted an investigation of the use of data mining for sketched diagram recognition. In particular we have focused on the automatic division of text and shapes. However, the tools we have developed and techniques used for building text-shape dividers can easily be applied to other recognition problems. This chapter provides a summary of the thesis and review of the contributions made to the area of sketch recognition through the development of more accurate recognisers. It also provides proposals for future research.

9.1 Conclusions

The key contributions of this research include the following.

- A comprehensive library of 118 ink features (including features from our second round analysis) was developed for sketch recognition. This includes a taxonomy to complement the feature library consisting of ten categories.
- DataManager was developed; a software tool to support the collection, labelling and automatic dataset generation of sketch data in a more efficient manner.
- A repository of labelled sketch data in a consistent format was developed. This includes our training dataset consisting of user interface diagrams, directed graphs and organisation diagrams and data collected for our evaluation, including mind-maps, to-do lists and UML class diagrams. Data from other researchers were also labelled and converted to a consistent format. This data includes ER and process diagrams, logic diagrams, Euler diagrams and COA symbols.

- A systematic investigation of data mining techniques for sketched diagram recognition, using text-shape division as an exemplar, was carried out. In particular a preliminary investigation, tuning of classifiers, the use of feature selection and ensembles and a second round of analysis. This investigation identified LogitBoost and LADTree as the optimal algorithms for text-shape division.
- An improvement of sketch recognition techniques for diagrams as a result of building more accurate recognisers.

The objective of this research programme was to improve recognition of hand-drawn diagrams by developing more accurate recognisers using data mining. Our sub-objectives, as stated in Chapter 1, were to assemble a comprehensive ink feature library, build a repository of hand-drawn diagrams and use text-shape division as an exemplar to systematically identify the most optimal algorithms, in combination with our feature library, for this problem using data mining techniques. All objectives were fulfilled as described by our key contributions above.

Chapter 2 presented related work on sketch recognition techniques, features, data collection and data mining tools and techniques. Our review of sketch recognition techniques found that very little attention had been given to the automatic division of text and shapes for sketched diagrams. The existing dividers used a number of features and algorithms. However, to our knowledge, no systematic analysis of algorithms had been carried out for text-shape division, or indeed more general shape recognition, and accuracy was still lacking. Our examination of the role of features identified numerous studies all showing the importance of good features to achieve accurate recognition. Although many feature sets exist, there was no comprehensive feature library bringing these sets together as a resource for recogniser development. In the area of data collection, we found that the data publicly available was insufficient for our analysis. Numerous data collection tools were identified; but none were able to support data collection, labelling and dataset generation with the use of a comprehensive feature library. In addition, we presented background information on data mining tools and techniques. This method of analysis, with the assistance of the data mining tool Weka, was identified as an ideal method of analysis for our research.

The first step in our investigation was to compile a feature library. Our library consists of 118 features (which include four features from our second round of analysis) obtained from previous work and including our own additions: see Appendix D for a full listing. Each feature comes with corresponding implementation. We developed a feature taxonomy to assist us in describing the feature library. The taxonomy consists of the following categories: curvature; density; direction; intersections; pressure; size;

spatial context; temporal context; time/speed, and divider results. The feature library is presented using this taxonomy.

In order to collect and manage data for our analysis we built a data collection tool, DataManager, to assist in collection and labelling of data and automatic dataset generation. This tool was designed to minimise biases that may occur when collecting data. For example, sketches are constructed by participants using a written description rather than by copying a pre-drawn example. This ensures that the timing data obtained from the sketches are more realistic. The labelling of data can be done automatically using a previous divider implementation and then corrected manually. This greatly reduces time spent labelling. Dataset generation is also done automatically with the use of our feature library. Features are calculated for each stroke of data collected to generate feature vectors which are then written to a spreadsheet. This dataset file can be used for analysis in external data mining tools such as Weka. DataManager has since been extended by others (Schmieder 2009; Chang 2010) to include several new functions and has been used in other research to collect and label data and generate datasets (Delaney et al. 2010).

A usability study was conducted to test the ease of data collection for participants, and the efficiency of labelling and dataset generation. The results of the study showed that the data collection interface provides a good environment for capturing ink data. In addition, the labelling and dataset generation could be completed in 10.5 minutes for 476 strokes. In comparison to our past experience, where it took more than three days to complete the same task for 1519 strokes, the use of DataManager improved the efficiency of performing these tasks immensely.

A training dataset for analysis was collected using DataManager. It consisted of user interface sketches, directed graphs and organisation charts with a total of 7248 strokes. These domains were selected to provide a good variation of data for training. A verification dataset, collected in previous work (Schmieder et al. 2009), was also described here. It consisted of ER and process diagrams. This dataset was used as an independent preliminary test dataset during the recogniser development process.

With our feature search and data collection complete, we proceeded to analyse our training dataset using data mining with Weka. The first stage involved a preliminary analysis of 39 algorithms. Based on their initial performance on the training data and expert knowledge, this list was reduced to a group of seven algorithms: Bagging; RandomForest; LogitBoost; LADTree; LMT; Multilayer Perceptron, and SMO. The next step involved tuning the parameters of this group of algorithms to determine the optimal settings. This

step resulted in significant increases in accuracy for some algorithms. The tuned LADTree and LogitBoost were found to be significantly more accurate than all other models tested with an average of 97.48% and 96.70% respectively of the training dataset correctly classified.

Three methods of feature selection were then applied to each tuned algorithm to determine if any further improvement in accuracy could be made. Various ensembles of algorithms were also tested to see if any gains could be made. However, the use of feature selection and ensembles did not result in any statistically significant improvements in classification accuracy.

The last stage of the analysis involved a second round of searching for features to identify commonly misclassified arrows/connectors. Four features were added to the feature library. Although there was an increase in correct identification of arrows/connectors, the results were not statistically significant and came at the high cost of misclassifying more text. Based on these results, the tuned LogitBoost and LADTree classifiers (LADTree 1) were chosen as the best dividers to use in our evaluation. We also included the top two ensembles (Vote 1 and Vote 2) and a second round version of the LADTree (LADTree 2) to further evaluate their performances on a more comprehensive test set.

These five new dividers were evaluated against three existing dividers: Entropy, Divider 2007 and the Microsoft divider. A fresh independent test set of diagrams was used for the evaluation. Using DataManager, we collected mind-maps, to-do lists and UML class diagrams, and obtained Euler diagrams, COA symbols and logic diagrams from other sketch recognition researchers (Alvarado et al. 2007; Bhat et al. 2009; Delaney et al. 2010). There were 16,047 strokes in total.

LogitBoost and LADTree 1 were the best performing dividers overall, followed closely by LADTree 2. They were found to be significantly more accurate than the three existing dividers. An overall ranking is shown in Table 60 with simple and weighted averages obtained from Tukey's confidence intervals. All the new dividers ranked above the existing ones except for Vote 1, which is below the Microsoft divider.

Divider	Simple Average	Weighted Average
1. LogitBoost	89.61	90.50
2. LADTree 1	89.66	89.93
3. LADTree 2	88.47	89.01
4. Vote 2	87.89	86.37
5. Microsoft	85.06	87.07
6. Vote 1	85.87	82.93
7. Entropy	83.19	78.16
8. Divider 2007	78.86	77.67

Table 60 Overall Ranking and Averaged Results

Results varied within each diagram domain. For mind-maps and UML diagrams, the new dividers were significantly better than Divider 2007 and Entropy but not significantly different to the Microsoft divider. These diagrams were the most well classified domains in our evaluation. However the Microsoft divider showed a large bias towards text for mind-maps. There was no significant difference between all the dividers for to-do lists, but biases towards text were apparent for Entropy and the Microsoft divider. Entropy and Vote 2 were the most successful at classifying Euler diagrams. For the COA symbols, the dividers were not significantly different except for the Microsoft divider, which was significantly less accurate than all others. The results for the logic diagram dataset were the lowest overall. LogitBoost was significantly more accurate than all other dividers for this domain.

A short study of domain-specific dividers on more unique diagram datasets was conducted. In particular, to-do lists, logic diagrams, Euler diagrams and COA symbols were explored. Domain-specific dividers were trained using half of the dataset for training and the other half for testing. The domain-specific dividers for to-do lists and logic diagrams were more accurate than the general dividers. For Euler diagrams, the domain-specific dividers were significantly better than all the general dividers except Entropy, which was not significantly different. For the COA symbols, the domain-specific dividers were significantly more accurate than the Microsoft divider but not significantly different to all other dividers. When the domain of a diagram is known, it may be worth building a domain-specific divider.

Our new general dividers, in particular LADTree 1 and LogitBoost, produce significantly more accurate results than three existing dividers. Therefore, we conclude that with a systematic data mining analysis of algorithms and with the use of a comprehensive feature set, we have been able to improve the accuracy of text-shape dividers. In doing so, we have improved the recognition of hand-drawn diagrams.

9.2 Future Research

We have thoroughly explored the use of data mining for dividers and have had some successful results as highlighted above. The question is, how can further improvements be made?

We believe that in order to further improve the accuracy of text-shape dividers, the next step is to focus more on the context of strokes. Although we have incorporated features on spatial and temporal context into our library, this is restricted by the fact that these are measured in single values; there is no provision for adding more fuzzy conditions or to use semantic references to surrounding strokes or objects. There has been previous work in this area, such as LADDER (Hammond et al. 2005) and Costagliola et al (2005), which both use sketching languages to describe how sketched components are drawn and the relationships between components. Avola et al (2008) also propose contextual features that may be useful for building semantic models of a sketch. However, there is potential for further investigation.

Another area of potential improvement is to employ techniques that continue to learn from user specific training data. Field et al (2009) found that recognition rates are higher when recognisers are trained with data provided by the user. With this training, the recognisers are exposed to the user's particular drawing style and thus are able to classify further examples more accurately.

User feedback on misclassifications made by recognisers could also be used. Multiple recognisers could run concurrently and an application could learn, from user feedback on misclassifications, which recogniser is best. For example, the recogniser that most successfully classifies the user's diagram is rated higher. This recogniser's classification could be given a higher weighting, if an ensemble is being used, or it could be chosen as the only recogniser to use for future classifications.

We would like to collect more data to more thoroughly investigate the effects of domain-specific training. For Euler diagrams and COA symbols especially, we had small datasets that were further split into training and test sets. The confidence intervals, used to show the significant differences between the domain-specific dividers and general dividers, were particularly wide for these domains. The confidence intervals are wide because the dataset used for testing is very small and therefore not enough information is available to predict the interval where the true mean might lie. Wide confidence intervals mean that there are more overlaps between dividers' intervals, making it more difficult to observe significant differences.

With more data, the confidence intervals would be narrower and we would be able to detect significant differences more reliably.

Further work on feature selection is also possible. As mentioned in Chapter 8, we would like to investigate the effects of aliases in the feature set on recognition accuracy. One approach is to remove known aliases and re-train and test classifiers on the reduced feature set.

Future work can also work towards accelerating the recognition process. There are many ways to do this. Instead of using the models and Weka libraries, the classifiers could be re-implemented. The re-implementation could ensure that only features required for classification are calculated, rather than using the entire feature library – as is currently required due to the design of the Weka libraries. For classifiers that use tree structures, this may result in very few features being calculated, depending on the path that the instance takes down the tree. In addition, the process of feature calculation can be greatly improved. For example, common calculations that are used by many features, such as finding the bounding box or length of a stroke, only need to be calculated once for an instance rather than re-calculating for each feature that uses such information. Features that iterate through all strokes in the diagram, for example when finding strokes close by for spatial context features, can be calculated at the same time, rather than iterating through the whole diagram multiple times. Further work on feature selection, such as identifying and removing common aliases in the feature library, may reduce the feature set without compromising recognition accuracy. With these modifications, we believe the time taken to divide strokes using the new dividers would greatly improve.

Although many improvements can be made, this work provides a solid foundation for developing not only text-shape dividers but sketch recognition techniques in general. With the use of DataManager, efficiency and ease of data collection, labelling and dataset generation has been achieved. The comprehensive feature library is a highly valuable input to building accurate recognisers. The systematic analysis of data mining techniques for text-shape division and the results produced for text-shape dividers has established that the use of such techniques are highly successful and can be applied to further sketch recognition problems.

Appendix A: Preliminary Analysis Results

	Classifier	% Correctly Classified (10-fold cross validation)	Notes
1	**Trees.RandomForest**	**95.99**	
2	**Meta.Bagging**	**95.31**	
3	**Functions.MultilayerPerceptron**	**95.02**	
4	Functions.Logistic	94.95	Related to LADTree and LogitBoost
5	Meta.MultiClassClassifier	94.95	Same as Logistic (as we have 2 classes)
6	**Trees.LMT**	**94.85**	
7	Functions.SimpleLogistic	94.70	Related to LADTree and LogitBoost
8	**Functions.SMO**	**94.70**	
9	Meta.ThresholdSelector	94.66	Uses Logistic too
10	Rules.JRip	94.45	
11	Trees.FT	94.14	Can use Logistic functions at inner or leaf nodes
12	Meta.Dagging	94.09	Uses SMO as a base classifier
13	Meta.Decorate	93.97	
14	Trees.J48graft	93.81	
15	Meta.AttributeSelectedClassifier	93.63	
16	Rules.Ridor	93.57	
17	Lazy.Ibk	93.13	K-Nearest Neighbour
18	Meta.ClassBalancedND	93.13	
19	Meta.DataNearBalancedND	93.13	
20	Meta.END	93.13	
21	Meta.ND	93.13	
22	Meta.OrdinalClassClassifier	93.13	
23	Trees.J48	93.13	
24	Rules.DTNB	92.81	
25	Lazy.Ib1	92.78	
26	Meta.FilteredClassifier	91.96	
27	**Trees.LADTree**	**91.94**	
28	**Meta.LogitBoost**	**91.87**	
29	Bayes.BayesNet	91.74	
30	Trees.ADTree	91.35	
31	Meta.RacedIncrementalLogitBoost	90.87	
32	Rules.PART	90.53	
33	Trees.RandomTree	89.91	
34	Meta.AdaBoostM1	89.47	
35	Rules.DecisionTable	89.45	
36	Meta.MultiBoostAB	86.89	Extension of AdaBoost with Decision Stump
37	Misc.VFI	85.93	Vote
38	Meta.EnsembleSelection	77.48	Ensemble
39	Trees.REPTree	77.48	Fast Decision Tree learner

Note: Algorithms listed in bold were chosen for further tuning.

Appendix B: Hybrid Feature Sets

The following is a listing of the feature selection algorithms used to compile the hybrid feature sets and each feature set used for feature selection in Section 6.3.3.

Feature set 1

Feature Selection Algorithms	
Evaluator	**Search Method**
ClassifierSubsetEval	RankSearch
ClassifierSubsetEval	GeneticSearch
ClassifierSubsetEval	BestFirst
WrapperSubsetEval	RankSearch
WrapperSubsetEval	GeneticSearch
ConsistencySubsetEval	RankSearch
ConsistencySubsetEval	GeneticSearch
ConsistencySubsetEval	GreedyStepwise
ConsistencySubsetEval	BestFirst
CfsSubsetEval	GreedyStepwise
CfsSubsetEval	BestFirst

Feature set 2

Feature Selection Algorithms	
Evaluator	**Search method**
ReliefFAttributeEval	Ranker
ChiSquaredAttributeEval	Ranker
SymmetricalUncertAttributeEval	Ranker
OneRAttributeEval	Ranker
InfoGainAttributeEval	Ranker
GainRatioAttributeEval	Ranker
SymmetricalUncertAttributeSetEval	FCBFSearch

Feature set 1

Features	
1	Divider result
2	Entropy
3	Average length of close end point strokes
4	Long side of enclosing rectangle of largest fragment
5	Number of strokes contained
6	Log start time from previous
7	Minimum pressure
8	Number of strokes horizontally close
9	Total time
10	Cos from first to last
11	Length of next stroke
12	Absolute angle
13	Convex hull area ratio
14	Arc fit radius
15	Enclosing rectangle ratio
16	Maximum speed
17	Maximum curvature
18	Average curvature
19	Curvature of previous stroke
20	Perimeter efficiency
21	Density 2
22	Speed to next stroke
23	Curvature of next stroke
24	Aspect
25	Distance from last stroke
26	Distance to next stroke
27	Total intersections
28	Divider closest stroke
29	Cos initial angle
30	Next stroke divider result
31	Previous stroke divider result
32	Number of Bezier cusps
33	Number of mid-point self intersections
34	Number of self intersections
35	Pressure variation
36	Log start time to next
37	Time from last stroke
38	Log length
39	Total angle and length ratio
40	Average density of close strokes
41	Number of strokes on same horizontal plane
42	Log longest side rectangle
43	Number of other intersections (ex self)
44	Minimum speed
45	Density 1
46	Bounding box diagonal length
47	Sin initial angle
48	Direction
49	Bounding box width
50	Number of strokes of similar height

Feature set 2

Features	
1	Divider result
2	Log length
3	Log aspect
4	Long side of enclosing rectangle of largest fragment
5	Bounding box diagonal length
6	Log longest side rectangle
7	Entropy
8	Density 2
9	Log area
10	Bounding box area
11	Bounding box width
12	Bounding box height
13	Length
14	Distance from last stroke
15	Perimeter to area
16	Total angle and length ratio
17	Minimum speed
18	Log time difference to next
19	Distance to next stroke
20	Time from last stroke
21	Log start time to next
22	Openness
23	X difference between strokes
24	Log time difference from previous
25	Time till next stroke
26	Log start time from previous
27	X start point difference
28	Least squares error
29	Density 1
30	Distance from first to last
31	Length ratio
32	Average speed
33	Previous stroke divider result
34	Number of strokes contained
35	Next stroke divider result
36	Number of strokes similar height
37	Y difference between strokes
38	Total time
39	Arc fit radius
40	Width height ratio
41	Number of close strokes
42	Average curvature
43	Y start point difference
44	Average length of close end point strokes
45	Largest fragment length
46	Length of next stroke
47	Perimeter efficiency
48	Thinness ratio
49	Divider closest stroke
50	Absolute curve of largest fragment

Appendix C: R Code for Calculating Tukey's Confidence Intervals

Written by Yong Wang

```
counts = read.csv("counts.csv")
c2 = counts[,-c(2,11:12)]
d1 = c2[1:8*2-1,]
d2 = c2[1:8*2,]

p = as.matrix( (d1 / (d1 + d2))[,2:7] )
n = as.matrix( d1[1,2:7] + d2[1,2:7] )
s = sqrt(colMeans(p * (1-p)))

# s can also be estimated with average covariance included

# Each sample size case

# y   Vector of means
# n   Numbers of observations in all groups
# s   Within-groups sample standard deviation

# Tukey confidence intervals (honestly significant differences)

ci = function(y, n, s, alpha=.95) {
  r = length(y)
  n = rep(n, length=r)
  N = sum(n)
  t(matrix(y,nrow=2,ncol=r,byrow=TRUE) + c(-1,1) * qtukey(alpha, r, N-r) * s / sqrt(N
/ r) / 2)
}

# Example 9.7 from (Ott et al. 2000), page 445

plot.ci = function(y, n, s, alpha=.95, lwd=1) {
  if( is.matrix(y) ) {
    k = nrow(y)
    plot(1:k, ylim=range(y), type="n",
         xlab="Method",ylab="Tukey Confidence Intervals")
```

```
    for(j in 1:ncol(y)) {
      cl = ci(y[,j], n[j], s[j])
      for(i in 1:k) lines(c(i,i)+(j-1)*.03, cl[i,], type="l", col=j+1, lwd=lwd)
      lines(1:8+(j-1)*.03, y[,j], type="l", col=j+1, lwd=lwd)
    }

    m = rowMeans(y)                          # simple means
    lines(1:8-.03, m, type="l")
    nm = ncol(y) / sum(1/n)  # harmonic mean
    sm = sqrt(mean(s^2/n) * nm)
    clm = ci(m, nm*ncol(y), sm)
    for(i in 1:k) lines(c(i,i)-0.03, clm[i,], type="l", lwd=lwd)

    w = n/s^2 / sum(n/s^2)
    wm = rowSums(sweep(y, 2, w, "*"))        # weighted means
    lines(1:8-.03*2, wm, type="l", lwd=lwd, lty=5)
    nm = ncol(y) / sum(1/n)  # harmonic mean
    clwm = ci(wm, nm*ncol(y), sm)
    for(i in 1:k) lines(c(i,i)-.03*2, clwm[i,], type="l", lwd=lwd, lty=1)
  }
  else {
    cl = ci(y, n, s)
    ylim = range(cl)
    k = nrow(cl)
    plot(1:k, ylim=ylim, type="n",
         xlab="Method", ylab="Tukey Confidence Intervals")
    for(i in 1:k) lines(c(i,i), cl[i,], type="l")
  }
  list(means=m, CI.means=clm, weighted.means=wm, CI.weighted.means=clwm)
}

plot.ci(p, n, s)
```

Appendix D: Full Feature Set Listing

Curvature

Feature	Description	Origin
Number of Bezier cusps	Number of bezier cusps.	(Microsoft Corporation 2007; Patel et al. 2007)
Number of polyline cusps	Number of polyline cusps.	(Patel et al. 2007)
$\sum$ |angle at each point|	Sum of the absolute value of the angle at each point of the stroke.	(Rubine 1991)
$\sum$(angle at each point)^2	Sum of the squared value of the angle at each point of the stroke.	(Rubine 1991)
Absolute curve largest fragment	The total absolute curvature of the largest fragment. Fragments are found using ShortStraw (Wolin et al. 2008), as described in section 4.1.	(Bishop et al. 2004)
Angle of bounding box diagonal	Angle of the bounding box diagonal.	(Rubine 1991)
Average curvature	Average curvature (total angle / number of stroke points).	(Paulson et al. 2008b)
Cos from first to last point.	Cosine of the angle between the first and last point of the stroke.	(Rubine 1991)
Cos of initial angle	Cosine of the initial angle of the stroke.	(Rubine 1991)
Curviness	$\sum$ abs value of the angle at each stroke point below a 19^o threshold.	(Long et al. 2000)
Distance from first to last point	Distance from the first point of the stroke to the last point of the stroke	(Rubine 1991)
Is straight line	Identifies strokes that are straight lines	(Willems et al. 2009)
Least squares error	Orthogonal distance squared between the least squares fitted line and the stroke points / stroke length.	(Sezgin et al. 2001; Paulson et al. 2008b)
Max curvature	Maximum curvature of the stroke.	(Paulson et al. 2008b)
NDDE	Normalised distance between direction extremes.	(Paulson et al. 2008a)
Number of cups	Identifies strokes that have a U shape	(Willems et al. 2009)
Number of direction changes	Number of changes in the direction of a stroke.	New
Number of fragments	Number of fragments in a stroke (fragmented according it's to corners). Fragments are found using ShortStraw (Wolin et al. 2008), as described in section 4.1.	(Bishop et al. 2004)
Openness	Distance from the first to last point of the stroke / size of the stroke's bounding box.	(Long et al. 2000)
Overtracing	Total angle / 2π .	(Paulson et al. 2008a)
Sin from first to last point	Sine of the angle between the first and last point of the stroke.	(Rubine 1991)
Sin of initial angle	Sine of the initial angle of the stroke.	(Rubine 1991)
Total angle	Total angle traversed by the stroke.	(Rubine 1991)
Total angle and length ratio	Total angle / stroke length.	(Long et al. 2000)
Total Angle Ratio	Total angle / $\sum$ |angle at each point|.	(Long et al. 2000)

Density

Feature	Description	Origin
Amount of ink inside	Amount of ink inside the strokes bounding box (count the number of points inside the bounding box	(Young 2005)
Density 1	Stroke length / distance between first & last point.	(Long et al. 2000)
Density 2	Stroke length / area of bounding box.	(Long et al. 2000)
Length ratio	Cumulative distance between stroke points/ length from start to end point of a stroke.	Adapted from (Rubine 1991)
Length:perimeter ratio	Stroke length / perimeter of the stroke's convex hull.	(Fonseca et al. 2001; Fonseca et al. 2002)
Point ratio	Number of points in the stroke's convex hull / number of points in the stroke.	(Leung et al. 2002)
Total length/bounding box diagonal length	Length of the stroke divided by the length of the bounding box diagonal.	(Young 2005)
Entropy	See Section 2.1 for a full description of entropy	(Bhat et al. 2009)

Direction

Feature	Description	Origin
DCR	Maximum change in direction / average change in direction.	(Paulson et al. 2008a)
Direction	Direction of the stroke (Eigenvector of the largest Eigen value)	(Bishop et al. 2004)
Eigen value ratio	The largest Eigen value/ smallest Eigen value.	(Bishop et al. 2004)
Largest fragment direction	Direction of largest fragment (eigenvector of the largest Eigen value. Fragments are found using ShortStraw (Wolin et al. 2008), as described in section 4.1	(Bishop et al. 2004)

Intersections

Features	Description	Origin
Number of end point self intersections	Number of self intersections at the endpoints of the stroke.	Adapted from (Qin 2005) (Patel et al. 2007)
Number of other self intersections	Number of self intersections that are not at the stroke's endpoint.	Adapted from (Qin 2005) (Patel et al. 2007)
Number of self intersections	Number of points where the stroke intersects itself.	Adapted from (Qin 2005) (Patel et al. 2007)

Pressure

Feature	Description	Origin
Average pressure	Mean average pressure of the stroke.	Adapted from (Nakai et al. 2002)
Maximum pressure	Maximum pressure value for the stroke.	Adapted from (Nakai et al. 2002)
Minimum pressure	Minimum pressure value for the stroke.	Adapted from (Nakai et al. 2002)
Number of pressure minima	Number of minima in pressure values for the stroke.	(Patel et al. 2007)

Size

Feature	Description	Origin
Arc fit radius	The radius of an arc fitted to the stroke.	(Paulson et al. 2008b)
Aspect	$\|45\pi/180 -$ angle of the bounding box diagonal $\|$.	(Long et al. 2000)
Bounding box area	Area of the bounding box of the stroke.	Adapted from (Fonseca et al. 2002; Hammond et al. 2002)
Bounding box diagonal length	Length of the bounding box diagonal line.	(Rubine 1991)
Bounding box height	Height of the bounding box of the stroke.	Adapted from (Fonseca et al. 2002; Hammond et al. 2002)
Bounding box maximum	The maximum of the stroke's bounding box width and height	(García Martín-Mantero 2010)
Bounding box width	Width of the bounding box of the stroke.	Adapted from (Fonseca et al. 2002; Hammond et al. 2002)
Convex hull area ratio	Ratio of area of convex hull to area of the enclosing rectangle of the stroke.	(Fonseca et al. 2002)
Enclosing rectangle ratio	Ratio of strokes enclosing rectangle width to height.	(Fonseca et al. 2002)
Largest fragment length	Arc length of the stroke's largest fragment. Fragments are found using ShortStraw (Wolin et al. 2008), as described in section 4.1	(Bishop et al. 2004)
Length	Total length of the stroke.	(Rubine 1991)
Log area	Log of the stroke's bounding box area.	(Long et al. 2000)
Log aspect	Log of the aspect feature.	(Long et al. 2000)
Log length	Log of the total length of the stroke.	(Machii et al. 1993; Long et al. 2000)
Log longest side rectangle	Log of the length of the longest side of the stroke's bounding box.	(Machii et al. 1993)
Long side of enclosing rectangle of largest fragment	The longest length of the largest fragment's enclosing rectangle. Fragments are found using ShortStraw (Wolin et al. 2008), as described in section 4.1	(Bishop et al. 2004)
Perimeter efficiency	$2\sqrt{(\pi}$ stroke's convex hull area) / stroke's convex hull perimeter.	(Leung et al. 2002)
Perimeter to area	Ratio of perimeter to area of the stroke's convex hull	(Fonseca et al. 2002)
Thinness ratio	Perimeter^2 of stroke's convex hull / area of stroke's convex hull	(Fonseca et al. 2001; Fonseca et al. 2002)
Width to height ratio	Ratio of the stroke's bounding box width to height.	Adapted from (Fonseca et al. 2002)

Spatial context

Feature	Description	Origin
Curvature		
Average curvature of close end point strokes	Average curvature (using total angle) of other strokes with endpoints close to current stroke.	(Ao et al. 2007)
Average curvature of close strokes	Average curvature (using total angle) of other strokes close to current stroke.	(Ao et al. 2007)
Density		
Average density of close end point strokes	Average density (stroke length / bounding box diagonal length) of other strokes close at end points to the current stroke.	(Ao et al. 2007)
Average density of close strokes	Average density (stroke length / bounding box diagonal length) of other strokes close to the current stroke.	(Ao et al. 2007)
Divider Results		
Divider 2007 closest stroke	Results of our text/shape divider for the closest stroke to the current. (Patel et al. 2007)	New
Intersections		
Number of other intersections	Number of points of intersection of the current stroke with other strokes (excluding self intersections).	Adapted from (Calhoun et al. 2002)
Number of other strokes intersecting	Number of other strokes that intersect the current stroke (excluding itself).	Adapted from (Fonseca et al. 2002; Hammond et al. 2002)
Number of strokes vertically overlapping	The number of strokes vertically overlapping the current stroke.	New
Total number of intersections	Total number of intersections (includes self intersections).	Adapted from (Calhoun et al. 2002)
Total number of strokes intersecting	Number of strokes that intersect the current stroke (including itself)	(Fonseca et al. 2002; Hammond et al. 2002)
Location		
Is contained	The stroke is contained by another stroke.	New
Number of close end point strokes	Number of other strokes whose endpoints are close to end points of the current stroke.	(Ao et al. 2007)
Number of close strokes	The number of other close strokes to the current stroke.	(Ao et al. 2007)
Number of strokes contained	Number of strokes contained inside the current stroke.	New
Number of strokes horizontally close	Number of strokes horizontally close to current stroke.	New
Number of strokes on same horizontal plane	The number of strokes on the same horizontal plane as the current stroke.	New
Number of vertically close strokes	The number of other strokes vertically close to the current stroke.	(Ao et al. 2007)
Smallest distance between strokes from end point	The smallest distance to another stroke from the current stroke's end point.	New
Smallest distance between strokes from start point	The smallest distance to another stroke from the current stroke's start point.	New
Size		
Average length of close end point strokes	Average length of other strokes close to end points of the current stroke.	(Ao et al. 2007)
Average length of close strokes	Average length of other strokes close to the current stroke.	(Ao et al. 2007)
Length of closest stroke	The length of the closest stroke to the current stroke where the closest stroke is found by measuring distance between the middle of the bounding box.	New
Number of strokes of similar height	Number of strokes of similar height to current stroke.	New

Temporal context

Feature	Description	Origin
Curvature		
Curvature of next stroke	Total angle of next stroke.	Adapted from (Rubine 1991) (Ao et al. 2007)
Curvature of previous stroke	Total angle of previous stroke.	Adapted from (Rubine 1991) (Ao et al. 2007)
Density		
Density of next stroke	Length of the next stroke divided by the length of the next stroke's bounding box diagonal.	Adapted from (Rubine 1991) (Ao et al. 2007)
Density of previous stroke	Length of the previous stroke divided by the length of the previous stroke's bounding box diagonal.	Adapted from (Rubine 1991) (Ao et al. 2007)
Divider Results		
Divider 2007 next stroke	Results of our text/shape divider for the next stroke. (Patel et al. 2007)	New
Divider 2007 previous stroke	Results of our text/shape divider for the previous stroke. (Patel et al. 2007)	New
Length		
Length of next stroke	Total length of next stroke.	(Ao et al. 2007)
Length of previous stroke	Total length of previous stroke.	(Ao et al. 2007)
Location/Distance		
Distance from last stroke	Distance between current stroke and previous stroke.	Adapted from (Young 2005) (Patel et al. 2007)
Distance to next stroke	Distance between current stroke and next stroke.	Adapted from (Young 2005) (Patel et al. 2007)
X difference between strokes	Difference in X co-ordinate between current stroke and next.	(Bishop et al. 2004)
X start point difference	Difference in starting X coordinates of current stroke to next stroke.	(Bishop et al. 2004)
Y difference between strokes	Difference in Y co-ordinate between current stroke and next.	(Bishop et al. 2004)
Y start point difference	Difference in starting Y coordinates of current stroke to next stroke.	(Bishop et al. 2004)
Time/Speed		
Log start time from previous	Log of time from start of previous stroke to start of current stroke.	(Bishop et al. 2004)
Log start time to next	Log of time from start of current stroke to start of the next stroke.	(Bishop et al. 2004)
Log time difference from previous	Log of the time between the start of the current and end of the previous stroke.	(Bishop et al. 2004)
Log time difference to next	Log of the time between the end of the current stroke and the start of the next stroke.	(Bishop et al. 2004)
Speed from last stroke	Speed (distance/time) between current stroke and previous stroke.	(Patel et al. 2007)
Speed to next stroke	Speed (distance/time) between current stroke and next stroke.	(Patel et al. 2007)
Time from last stroke	The time between the start of the current stroke and the end of the previous stroke.	(Patel et al. 2007)
Time till next stroke	The time between the end of the current stroke and the start of the next stroke.	(Patel et al. 2007)

Time/speed

Feature	Description	Origin
Number of speed minima	Number of extreme minima in the speed values for the stroke.	Adapted from(Sezgin et al. 2001) (Patel et al. 2007)
Average speed	Mean average speed when drawing the stroke.	Adapted from (Rubine 1991) (Patel et al. 2007)
Maximum speed	Maximum speed when drawing the stroke.	Adapted from (Rubine 1991) (Patel et al. 2007)
Maximum speed squared	Maximum speed of the stroke squared.	(Rubine 1991)
Minimum speed	Minimum speed when drawing the stroke.	Adapted from (Rubine 1991) (Patel et al. 2007)
Total duration	Total duration of the stroke from pen up to pen down.	(Rubine 1991)

Divider results

Feature	Description	Origin
Divider 2007 Result	Results of our text/shape divider on the current stroke. (Patel et al. 2007)	New
Tablet OS text probability	Tablet OS text recogniser probability of the stroke being text.	(Microsoft Corporation 2005) (Patel et al. 2007)

Second parse features

Feature	Description	Origin
Is Arrowhead	Second parse feature to identify misclassified arrowheads.	(Kara et al. 2004; Freeman et al. 2007)

Appendix E: Ethics Application Documents

Department of Computer Science
The University of Auckland
Private Bag 92019
Auckland

Tel: 09 373 7599

PARTICIPANT INFORMATION SHEET

Title: *Development of Techniques for Sketched Diagram Recognition*

Primary Investigator (PhD Student): *Rachel Patel*

To participants:

My name is Rachel Patel, I am a PhD student in the Department of Computer Science at the University of Auckland. I am conducting research into pen input computing with my primary supervisor Beryl Plimmer. I am investigating how computers can support more natural computer interaction. A part of exploring these ideas is involving potential 'ordinary' users in the design, usability testing and evaluation of the prototype applications. This particular project aims to support hand-drawing of diagrams.

In this study we are collecting base data for training our recognition engines.

You are invited to participate in our research and we would appreciate any assistance you can offer us, although you are under no obligation to do so.

Participation involves one visit to our laboratory at The University of Auckland, for approximately 1 hour. If you agree to participate, you may be asked to perform a number of tasks on paper, using a computer via the keyboard or mouse and using a computer with a pen. The tasks will be fully explained and demonstrated. You will be asked to construct diagrams. The changes you make and the time you spend working on each task will be digitally recorded using screen capture software which is non-identifying.

The digital recordings may be used in research reports on this project. Your consent form will be held in a secure file for 6 years, at the end of this time it will be properly disposed of. Your name will not be used in any reports arising from this study. The information collected during this study will be available publicly for other researchers in this field to be used in future analysis and publications and will be kept indefinitely. All of this information will be non-identifying. At the conclusion of the study, a summary of the findings will be available from the researchers upon request.

If you don't want to participate, you don't have to give any reason for your decision. If you do participate, you may withdraw at any time during the session and you can also ask for the information you have provided to be withdrawn at any time until one week after the conclusion of your session, without explanation and without penalty, by contacting me (details below). Although your information will contain no identifying information it will be marked with a numerical ID. You must take note of your ID and provide this if you wish to withdraw your information within the given time period. If you are a student at The University of Auckland choosing not to participate, or to withdraw yourself or your information, your grades or academic relationships with the University or members of staff will not be affected.

This project is funded through the University of Auckland and the Tertiary Education Commission Bright Futures Top Achievers Doctoral Scholarships.

If you agree to participate in this study, please first complete the consent form attached to this information sheet.

Thank you very much for your time and help in making this study possible. If you have any questions at any time you can phone me (3737599 ext 89357) or the Head of Department, Associate Professor Robert Amor (3737599 ext 83068), or you can write to us at:

Department of Computer Science,
The University of Auckland
Private Bag 92019
Auckland.

For any queries regarding ethical concerns, please contact The Chair, The University of Auckland Human Participants Ethics Committee, The University of Auckland, Office of the Vice Chancellor, Private Bag 92019, Auckland. Tel. 3737599 ext 87830.

APPROVED BY THE UNIVERSITY OF AUCKLAND HUMAN PARTICIPANTS ETHICS COMMITTEE on 5 December 2007 for a period of 3 years from 5 December 2007. Reference 2007/ 404.

Department of Computer Science
The University of Auckland
Private Bag 92019
Auckland

Tel: 09 373 7599

CONSENT FORM

This consent form will be held for a period of at least six years

Title: *Development of Techniques for Sketched Diagram Recognition*

Researcher: Rachel Patel

I have been given and understood an explanation of this research project. I have had an opportunity to ask questions and have them answered. I understand that at the conclusion of the study, a summary of the findings will be available from the researchers upon request.
I understand that the data collected from the study will be held indefinitely and will be publicly available for other researchers in this field.
I understand that all of the data collected from the study will be non-identifying.
I understand that I may withdraw myself and any information traceable to me at any time up to one week after the completion of this session without giving a reason, and without any penalty.
I understand that I may withdraw my participation during the laboratory session at any time.
I understand that my grades and relationships with The University of Auckland will be unaffected whether or not I participate in this study or withdraw my participation during it.

I agree to take part in this research by completing the laboratory session.

I **agree/do not agree** digital recordings taken during the session being used research reports on this project.

Signed:

Name:

(please print clearly)

Date:

APPROVED BY THE UNIVERSITY OF AUCKLAND HUMAN PARTICIPANTS ETHICS COMMITTEE on 5 December 2007 for a period of 3 years from 5 December 2007. Reference 2007/ 404.

Appendix F: Usability Study Questionnaire

Complete only This Section before the session

I have used a pen input on a computer	Frequently	Occasionally	A couple of times	Once	Never

---

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
General Questions:					
This exercise was enjoyable					
About the Organization Diagram:					
I understand the task					
This interaction tools helped with my task completion					
About the Graph Diagram:					
I understand the task					
This interaction tools helped with my task completion					
About the Environment:					
Creating the diagram was easy					
Checking and editing the diagram was easy					
I would like to use this method of interaction in the future					

Comments/Recommendations:

Appendix G: Co-Authorship Forms

Appendix H: Permission for use of Copyright Material

References

(2009). "IKVM.NET." 6/15, 2009, from http://www.ikvm.net/

Alvarado, C. and R. Davis (2004). SketchREAD: a multi-domain sketch recognition engine. Proceedings of the 17th annual ACM symposium on User interface software and technology, Santa Fe, NM, USA, ACM Press.23-32

Alvarado, C. and M. Lazzareschi (2007). Properties of Real World Digital Logic Diagrams. 1st International Workshop on Pen-based Learning Technologies, Catania, Italy.1-6

Ao, X., J. Li, et al. (2006). Structuralizing digital ink for efficient selection. Proceedings of the 11th international conference on Intelligent user interfaces. Sydney, Australia, ACM**:** 148-154.

Ao, X., X. Wang, et al. (2007). Structuring and Manipulating Hand-Sketched Diagrams. Sketch Based Interfaces and Modeling (SBIM '07) pre-print edition, Riverside, California, Eurographics

Avola, D., A. D. Buono, et al. (2009). A Novel Online Textual/Graphical Domain Separation Approach for Sketch-Based Interfaces. 2nd International Symposium on Intelligent Interactive Multimedia Systems and Services (KES-IIMSS 2009), Mogliano Veneto, Italy.167-176

Avola, D., F. Ferri, et al. (2008). A Framework for Designing and Recognizing Sketch-Based Libraries for Pervasive Systems. UNISCON, Springer**:** 405-416.

Bailey, B. P. and J. A. Konstan (2003). Are Informal Tools Better? Comparing DEMAIS, Pencil and Paper, and Authorware for Early Multimedia Design. CHI 2003, Ft Lauderdale, ACM.313-320

Berthold, M. R., N. Cebron, et al. (2008). "KNIME: The Konstanz Information Miner." Data Analysis, Machine Learning and Applications. Studies in Classification, Data Analysis, and Knowledge Organization: 319-326.

Bhat, A. and T. Hammond (2009). Using Entropy to Distinguish Shape Versus Text in Hand-Drawn Diagrams. International Joint Conference on Artificial Intelligence (IJCAI '09), Pasadena, California, USA.1395-1400

Bickerstaffe, A., A. Lane, et al. (2007). Developing Domain-Specific Gesture Recognizers for Smart Diagram Environments. GREC-IAPR Workshop on Graphics Recognition, Curitiba, Brazil.145-156

Bishop, C. M., M. Svensen, et al. (2004). Distinguishing Text from Graphics in On-Line Handwritten Ink. Proceedings of the Ninth International Workshop on Frontiers in Handwriting Recognition, IEEE Computer Society**:** 142-147.

Black, A. (1990). "Visible planning on paper and on screen: The impact of working medium on decision-making by novice graphic designers." Behaviour and information technology **9**(4): 283-296.

Blagojevic, R., S. H.-H. Chang, et al. (2010). The Power of Automatic Feature Selection: Rubine on Steroids. Sketch based interfaces and modeling, Annecy, France, Eurographics.79-86

Blagojevic, R., P. Schmieder, et al. (2009). Towards a Toolkit for the Development and Evaluation of Sketch Recognition Techniques. Intelligent User Interfaces (IUI'09) Sketch Recognition Workshop, Florida, USA

Bouckaert, R. R., E. Frank, et al. (2010). "WEKA-Experiences with a Java Open-Source Project." Journal of Machine Learning Research **11**: 2533-2541.

Breiman, L. (1996). "Bagging predictors." Machine Learning **24**(2): 123-140.

Breiman, L. (2001). "Random Forests." Machine Learning **45**(1): 5-32.

Buxton, B. (2007). Sketching User Experiences: Getting the Design Right and the Right Design (Interactive Technologies). San Francisco, Morgan Kaufmann.

Calhoun, C., T. F. Stahovich, et al. (2002). Recognising Multi-Stroke Symbols. AAAI Spring Symposium on Sketch Understanding.15-23

Chang, S. H.-H. (2010). Applying Data Mining for the Recognition of Digital Ink Strokes. Computer Science. Auckland, University of Auckland. **MEng:** 176.

Chang, S. H.-H., B. Plimmer, et al. (2010). Rata.SSR: Data Mining for Pertinent Stroke Recognizers. Sketch-based interfaces and modeling, Annecy, France, Eurographics.95-102

Chen, Q., J. Grundy, et al. (2003). An E-whiteboard application to support early design-stage sketching of UML diagrams. Human Centric Computer Languages and Environments, Auckland, NZ, IEEE.219-226

Chen, Q., J. Grundy, et al. (2008). "SUMLOW: early design-stage sketching of UML diagrams on an E-whiteboard." Software Practice and Experience **38**(9): 961-994.

Chik, V. C. H. (2008). Intelligent Mind-mapping. Computer Science. Auckland, University of Auckland. **MEng:** 119.

Choi, H., B. Paulson, et al. (2008). Gesture Recognition Based on Manifold Learning. Proceedings of the 2008 Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition. Orlando, Florida, Springer-Verlag 247-256.

Chung, R., P. Mirica, et al. (2005). InkKit: A Generic Design Tool for the Tablet PC. CHINZ 05, Auckland, ACM.29-30

Costagliola, G., V. Deufemia, et al. (2005). Sketch Grammars: A Formalism for Describing and Recognizing Diagrammatic Sketch Languages. International Conference on Document Analysis and Recognition (ICDAR '05), Washington, DC, USA, IEEE Computer Society.1226-1231

Damm, C. H., K. M. Hansen, et al. (2000). Tool support for cooperative object-oriented design: Gesture based modelling on and electronic whiteboard. Chi 2000, ACM.518-525

Delaney, A., B. Plimmer, et al. (2010). Recognizing Sketches of Euler Diagrams Drawn with Ellipses. Sixteenth International Conference on Distributed Multimedia Systems, Chicago, USA.305--310

Demšar, J., B. Zupan, et al. (2004). Orange: From Experimental Machine Learning to Interactive Data Mining. Knowledge Discovery in Databases: PKDD 2004, Lecture Notes in Computer Science, Springer Berlin / Heidelberg. **3202/2004:** 537- 539.

Dix, A., J. Finlay, et al. (2004). Human-Computer Interaction. Harlow, England, Prentice Hall.

Efron, B. and R. Tibshirani (1993). An Introduction to the Bootstrap, Chapman and Hall.

Elrod, S., R. Bruce, et al. (1992). "Liveboard: A large interactive display supporting group meetings, presentations and remote collaboration." CHI '92: 599-607.

Field, M., S. Gordon, et al. (2009). "The effect of task on classification accuracy: Using gesture recognition techniques in free-sketch recognition." CAD/GRAPHICS 2009 **34**(5): 499-512.

Fonseca, M. J. and J. A. Jorge (2001). Experimental Evaluation of an on-line Scribble Recognizer. Pattern Recognition Letters.1311–1319

Fonseca, M. J., C. e. Pimentel, et al. (2002). CALI: An Online Scribble Recogniser for Calligraphic Interfaces. AAAI Spring Symposium on Sketch Understanding, IEEE.51-58

Fowler, M. and K. Scott (1999). UML Distilled: A Brief Guide to the Standard Object Modeling Language (2nd Edition) Addison-Wesley Professional.

Frank, E. (2009 - 2010). Personal Communication. University of Waikato, Hamilton.

Freeman, I. and B. Plimmer (2007). Connector Semantics for Sketched Diagram Recognition. AUIC, Ballarat, Australia, ACM.71-78

Friedman, J., T. Hastie, et al. (2000). "Additive Logistic Regression: a Statistical View of Boosting." The Annals of Statistics **28**(2): 337-407.

García Martín-Mantero, Y. (2010). Editor gráfico de diagramas de clases basado en trazos naturales, UCLM (University of Castilla La Mancha), Spain.

Glaser, B. G. and A. L. Strauss (1967). The Discovery of Grounded Theory: Strategies for Qualitative Research. Chicago, Aldine Publishing Company.

Goel, V. (1995). Sketches of thought. Cambridge, Massachusetts, The MIT Press.

Gross, M. (1996). "The Electronic Cocktail Napkin-a computational environment for working with design diagrams." Design Studies **17**(1): pp. 53-69(17).

Grundy, J. C. and J. G. Hosking (2007). Supporting generic sketching-based input of diagrams in a domain-specific visual language meta-tool. IEEE/ACM International Conference on Software Engineering, Minneapolis, USA.282-291

Gütlein, M., E. Frank, et al. (2009). Large-scale attribute selection using wrappers. IEEE Symposium on Computational Intelligence and Data Mining, IEEE.332-339

Hall, M., E. Frank, et al. (2009). "The WEKA data mining software: An update." SIGKDD Explorations **11(1)**: 10-18.

Hammond, T. and R. Davis (2002). Tahuti: A Geometrical Sketch Recognition System for UML Class Diagrams. 2002 AAAI Spring Symposium on Sketch Understanding.59-66

Hammond, T. and R. Davis (2005). "LADDER: A Sketching Language for User Interface Developers." Computer and Graphics, Elsevier **29**: 518--532.

Hastie, T. and R. Tibshirani (1998). Classification by Pairwise Coupling. Advances in Neural Information Processing Systems, Denver, Colorado, United States MIT Press.507-513

Holmes, G., B. Pfahringer, et al. (2002). Multiclass Alternating Decision Trees. Machine Learning: ECML 2002, Springer Berlin / Heidelberg. **2430:** 105-122.

Holte, R. C. (1993). "Very simple classification rules perform well on most commonly used datasets." Machine Learning **11**(1): 63-90.

Hse, H. and A. R. Newton (2004). Sketched Symbol Recognition using Zernike Moments. International Conference on Pattern Recognition, Cambridge, UK.367-370

Hutton, G., M. Cripps, et al. (1997). A Strategy for On-line Interpretation of Sketched Engineering Drawings. Proceedings of the 4th International Conference on Document Analysis and Recognition, IEEE Computer Society.771-775

Jain, A. K., A. M. Namboodiri, et al. (2001). Structure in On-line Documents. Proceedings of the Sixth International Conference on Document Analysis and Recognition, IEEE Computer Society**:** 844-848.

Jansen, A. R., K. Marriott, et al. (2003). Cider: A component-based toolkit for creating smart diagram environments. International Conference on Distributed and Multimedia Systems, Miami.353-359

Johnson, G. (2009). Picturephone: A game for sketch data capture. Intelligent User Interfaces (IUI'09) Workshop on Sketch Recognition, Sanibel Island, Florida

Johnson, G. and E. Y.-L. Do (2009a). Games for sketch data collection. Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling. New Orleans, Louisiana, ACM**:** 117-123.

Johnson, G., M. D. Gross, et al. (2009b). "Computational Support for Sketching in Design: A Review." Foundations and Trends in Human-Computer Interaction **2**(1): 1-93.

Kara, L. B. and T. F. Stahovich (2004). Hierarchical Parsing and Recognition of HandSketched Diagrams. UIST '04, Santa Fe, New Mexico, ACM Press.13 - 22

Kaster, B., E. Jacobson, et al. (2009). SOUSA v2.0: Automatically Generating Secure and Searchable Data Collection Studies. International Workshop on Visual Languages and Computing (VLC 2009), Redwood City, CA, USA.365-368

Keerthi, S. S., S. K. Shevade, et al. (2001). "Improvements to Platt's SMO Algorithm for SVM Classifier Design." Neural Computation **13**(3): 637-649.

Kira, K. and L. A. Rendell (1992). A Practical Approach to Feature Selection. Ninth International Workshop on Machine Learning, Aberdeen, Scotland, United Kingdom, Morgan Kaufmann Publishers, San Francisco, CA.249-256

Kittler, J., M. Hatef, et al. (1998). "On combining classifiers." IEEE Transactions on Pattern Analysis and Machine Intelligence **20(3)**: 226-239.

Kohavi, R. and G. H. John (1997). "Wrappers for feature subset selection." Artificial Intelligence **97(1-2)**: 273-324.

Kononenko, I. (1994). Estimating Attributes: Analysis and Extensions of RELIEF. European Conference on Machine Learning, Catania, Italy.171-182

Kuncheva, L. I. (2004). Combining Pattern Classifiers: Methods and Algorithms, John Wiley and Sons, Inc.

Landay, J. and B. Myers (1995). Interactive sketching for the early stages of user interface design. Chi '95 Mosaic of Creativity, ACM.43-50

Landay, J. and B. Myers (1996). Sketching storyboards to illustrate interface behaviors. CHI '96, Vancouver, BC Canada, ACM.193-194

Landwehr, N., M. Hall, et al. (2005). "Logistic Model Trees." Machine Learning **59**(1-2): 161-205.

Lank, E., J. S. Thorley, et al. (2000). An interactive system for recognizing hand drawn UML diagrams. Proceedings of the Centre for Advanced Studies on Collaborative research. Mississauga, Ontario, Canada, IBM Press**:** 7.

LeBlanc, D. C. (2004). Statistics: Concepts and Applications for Science Jones & Bartlett Publishers.

Leung, W. H. and T. Chen (2002). User-independent retrieval of free-form hand-drawn sketches. Acoustics, Speech, and Signal Processing, 2002. Proceedings. (ICASSP '02), Orlando, Florida.2029-2032

Lin, J., M. W. Newman, et al. (2000). Denim: Finding a tighter fit between tools and practice for web design. Chi 2000, ACM.510-517

Long, A. C., J. A. Landay, et al. (2000). Visual similarity of pen gestures. CHI 2000, The Hague, Amsterdam, ACM.360-367

Machii, K., H. Fukushima, et al. (1993). On-line text/drawings segmentation of handwritten patterns. Document Analysis and Recognition, Tsukuba Science City, Japan.710-713

MacLean, S., D. Tausky, et al. (2009). Tools for the efficient generation of hand-drawn corpora based on context-free grammars. Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling. New Orleans, Louisiana, ACM**:** 125-132.

Meyer, B., K. Marriott, et al. (2009). Intelligent diagramming in the electronic online classroom. Human System Interaction, Catania.174-180

Microsoft Corporation (2005). Microsoft Windows XP Tablet PC Edition Software Development Kit

Microsoft Corporation (2007). "MSDN Stroke.BezierCusps Property." from http://msdn2.microsoft.com/en-us/library/microsoft.ink.stroke.beziercusps.aspx

Microsoft Corporation (2008). "Ink Analysis Overview." 2008, from http://msdn.microsoft.com/en-us/library/ms704040(VS.85).aspx

Mierswa, I., M. Wurst, et al. (2006). YALE: rapid prototyping for complex data mining tasks. 12th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '06), New York, NY, USA.935-940

Minsky, M. and S. Papert. (1969). Perceptrons. Cambridge, MA: MIT Press.

Mochida, K. and M. Nakagawa (2003). Separating drawings, formula and text from free handwriting. International Graphonomics Society (IGS2003), Scottsdale, Arizona.216-219

Mochida, K. and M. Nakagawa (2004). "Separating Figure, Mathematical Formula and Japanese Text from Free Handwriting In Mixed Online Documents." International Journal of Pattern Recognition and Artificial Intelligence **18**(7): 1173 - 1187.

Moran, T. P., P. Chiu, et al. (1997). Pen-Based interaction techniques for organizing material on an electronic whiteboard. 10th Annual Symposium on User Interface Software and Technology, Banff, Canada, ACM.45-54

Moran, T. P., W. van Melle, et al. (1998). Spatial interpretation of domain objects integrated into a freeform electronic whiteboard. UIST '98, San Francisco, CA, ACM SIGSOFT.175-184

Nakai, M., T. Sudo, et al. (2002). Pen Pressure Features for Writer-Independent On-Line Handwriting Recognition Based on Substroke HMM. Proceedings of the 16 th International Conference on Pattern Recognition (ICPR'02) Volume 3, IEEE Computer Society.30220

Niels, R., D. Willems, et al. (2008). The NicIcon database of handwritten icons. 11th International Conference on Frontiers in Handwriting Recognition (ICFHR 2008), Montréal, Canada.296-301

Oltmans, M., C. Alvarado, et al. (2004). ETCHA Sketches: Lessons learned from collecting sketch data. AAAI Fall Symposium on Making Pen-Based Interaction Intelligent and Natural. .134-140

Ott, R. L. and M. T. Longnecker (2000). An Introduction to Statistical Methods and Data Analysis, Duxbury Press.

Patel, R. (2007). Exploring better techniques for diagram recognition. MSc in Computer Science. Auckland, University of Auckland. **MSc:** 146.

Patel, R., B. Plimmer, et al. (2007). Ink Features for Diagram Recognition. 4th Eurographics Workshop on Sketch-Based Interfaces and Modeling Riverside, California, Eurographics.131-138

Paulson, B. and T. Hammond (2008a). PaleoSketch: Accurate Primitive Sketch Recognition and Beautification. Intelligent User Interfaces (IUI '08), New York, USA, ACM Press.1-10

Paulson, B., P. Rajan, et al. (2008b). What!?! No Rubine Features?: Using Geometric-based Features to Produce Normalized Confidence Values for Sketch Recognition VL/HCC Workshop: Sketch Tools for Diagramming, Herrsching am Ammersee, Germany.57-63

Paulson, B., A. Wolin, et al. (2008c). SOUSA: Sketch-based Online User Study Applet. Sketch Based Interfaces and Modeling, Annecy, France, Eurographics.81-88

Pedersen, E. R., K. McCall, et al. (1993). Tivoli: An electronic whiteboard for informal workgroup meetings. CHI '93, ACM.391-398

Peterson, E., T. Stahovich, et al. (2010). Grouping Strokes into Shapes in Hand-Drawn Diagrams. Proc. of the 24th AAAI Conference on Artificial Intelligence (AAAI-10).974-979

Platt, J. (1999). Fast training of support vector machines using sequential minimal optimization. Advances in Kernel Methods - Support Vector Learning.185-208

Plimmer, B. (2004). Using Shared Displays to Support Group Designs; A Study of the Use of Informal User Interface Designs when Learning to Program. Computer Science, University of Waikato. **PhD**.

Plimmer, B. and I. Freeman (2007). A Toolkit Approach to Sketched Diagram Recognition. HCI, Lancaster, UK, eWiC.205-213

Plimmer, B., H. Purchase, et al. (2010). SketchNode: Intelligent sketching support and formal diagramming. OZCHI 2010. Brisbane, ACM**:** 136-143.

Plimmer, B. E. and M. Apperley (2003a). Evaluating a Sketch Environment for Novice Programmers. SIGCHI, Ft Lauderdale, ACM.1018-1019

Plimmer, B. E. and M. Apperley (2003b). Freeform: A Tool for Sketching Form Designs. BHCI, Bath.183-186

Plimmer, B. E. and M. Apperley (2004). INTERACTING with sketched interface designs: an evaluation study. SigChi 2004, Vienna, ACM.1337-1340

Qi, Y., M. Szummer, et al. (2005). Diagram Structure Recognition by Bayesian Conditional Random Fields. Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2, IEEE Computer Society**:** 191-196.

Qin, S. (2005). Intelligent Classification of Sketch Strokes. EUROCON, Serbia & Montenegro, Belgrade, IEEE.1374-1377

Quinlan, R. (1993). "C4.5: Programs for Machine Learning." Machine Learning **16**(3): 235-240.

R Development Core Team (2006). R: A Language and Environment for Statistical Computing. Vienna, Austria, R Foundation for Statistical Computing.

Robnik-Sikonja, M. and I. Kononenko (1997). "An adaptation of Relief for attribute estimation in regression." Fourteenth International Conference on Machine Learning 296-304.

Rodríguez, J. A., G. Sánchez, et al. (2008). "Categorization of Digital Ink Elements Using Spectral Features " Graphics Recognition. Recent Advances and New Opportunities, Lecture Notes in Computer Science **5046/2008**: 181-190.

Romer, A., S. Leinert, et al. (2000). "External Support of Problem Analysis in Design Problem Solving." Research in Engineering Design **12**(3): 144-151.

Rubine, D. H. (1991). Specifying gestures by example. Proceedings of Siggraph '91, ACM.329-337

Rumelhart, D. E., G. E. Hinton, et al. (1986). Learning internal representations by error propagation. Cambridge, MIT Press.

Schmieder, P. (2009). Comparing Basic Shape Classifiers: A Platform for Evaluating Sketch Recognition Algorithms. Computer Science. Auckland, University of Auckland. **MSc:** 177.

Schmieder, P., B. Plimmer, et al. (2009). Automatic Evaluation of Sketch Recognizers. Sketch Based Interfaces and Modelling, New Orleans, USA.85-92

Seniuk, A. and D. Blostein (2009). Pen Acoustic Emissions for Text and Gesture Recognition. 10th International Conference on Document Analysis and Recognition, Barcelona, Spain.872-876

Sezgin, M. and R. Davis (2007). Temporal Sketch Recognition in Interspersed Drawings. Sketch Based Interfaces and Modeling (SBIM '07), Riverside, California, USA.15-22

Sezgin, T. M. and R. Davis (2005). HMM-based efficient sketch recognition. Proceedings of the 10th international conference on Intelligent user interfaces, San Diego, California, USA, ACM Press.281-283

Sezgin, T. M., T. Stahovich, et al. (2001). Sketch based interfaces: early processing for sketch understanding. Proceedings of the 2001 workshop on Perceptive user interfaces, Orlando, Florida, ACM Press.1-8

Shilman, M. and P. Viola (2004). Spatial recognition and grouping of text and graphics. Eurographics Workshop on Sketch-Based Interfaces and Modeling.91-95

Shilman, M., Z. Wei;, et al. (2003). Discerning structure from freeform handwritten notes. Document Analysis and Recognition.60 - 65

Signer, B., U. Kurmann, et al. (2007a). iGesture: A General Gesture Recognition Framework. 9th International Conference on Document Analysis and Recognition (ICDAR), Curitiba, Brazil.954-958

Signer, B., M. C. Norrie, et al. (2007b). iGesture: A Java Framework for the Development and Deployment of Stroke-Based Online Gesture Recognition Algorithms. Zurich, Switzerland, Institute for Information Systems, ETH Zurich.

Snedecor, G. W. and W. G. Cochran (1989). Statistical methods. Iowa, Blackwell Publishing Professional.

Stahovich, T. F., R. Davis, et al. (1995). Turning sketches into working geometry. ASME Design Theory and Methodology.603-611

Sumner, M., E. Frank, et al. (2005). Speeding up Logistic Model Tree Induction. 9th European Conference on Principles and Practice of Knowledge Discovery in Databases, Porto, Portugal, Springer.675-683

Szummer, M. and Y. Qi (2004). Contextual recognition of hand-drawn diagrams with conditional random fields. Ninth International Workshop on Frontiers in Handwriting Recognition (IWFHR).32-37

Tumen, R. S., M. E. Acer, et al. (2010). Feature extraction and classifier combination for image-based sketch recognition. Joint Session of the Seventh Sketch-Based Interfaces and Modeling Workshop and Eighth Symposium on Non-Photorealistic Animation and Rendering. Annecy, France, Eurographics Association**:** 63-70.

Waranusast, R., P. Haddawy, et al. (2009). Segmentation of Text and Non-text in On-Line Handwritten Patient Record Based on Spatio-Temporal Analysis. Proceedings of the 12th Conference on Artificial Intelligence in Medicine: Artificial Intelligence in Medicine. 345-354

Willems, D. and R. Niels (2008). Definitions for features used in online pen gesture recognition, NICI, Radboud University Nijmegen.

Willems, D., R. Niels, et al. (2009). "Iconic and multi-stroke gesture recognition." Pattern Recogn. **42**(12): 3303-3312.

Witten, I. H. and E. Frank (2005). Data Mining: Practical machine learning tools and techniques. San Francisco, Morgan Kaufmann.

Wobbrock, J. O., A. D. Wilson, et al. (2007). Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. User interface software and technology. Newport, Rhode Island, USA, ACM**:** 159-168.

Wolin, A., B. Eoff, et al. (2008). Short Straw: A Simple and Effective Corner Finder for Polylines. Sketch Based Interfaces and Modeling (SBIM '08), Annecy, France, Eurographics.33-40

Wolin, A., D. Smith, et al. (2007). A Pen-based Tool for Efficient Labelling of 2D Sketches. 4th Eurographics Workshop on Sketch-Based Interfaces and Modeling, Riverside, CA.67-74

Yeung, L., B. Plimmer, et al. (2008). Effect of Fidelity in Diagram Presentation. HCI, Liverpool.p. 35-45

Young, M. (2005). InkKit: The Back End of the Generic Design Transformation Tool. Computer Science. Auckland, University of Auckland. **BEng**.

Yu, B. and S. Cai (2003). A domain-independent system for sketch recognition. Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia, Melbourne, Australia, ACM Press.141-146

Zeleznik, R. C., A. Bragdon, et al. (2008). Lineogrammer: creating diagrams by drawing. Proceedings of User interface software and technology. Monterey, CA, USA, ACM**:** 161-170.

Zhang, L. and Z. Sun (2007). "An experimental comparison of machine learning for adaptive sketch recognition." Applied Mathematics and Computation **185**(2): 1138-1148.