Computers & Graphics 35 (2011) 976–991

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

Using data mining for digital ink recognition: Dividing text and shapes in sketched diagrams

Rachel Blagojevic [a,*], Beryl Plimmer [a], John Grundy [b], Yong Wang [a]

[a] University of Auckland, Private bag 92019, Auckland, New Zealand
[b] Swinburne University of Technology, PO Box 218, Hawthorn, Victoria 3122, Australia

ARTICLE INFO

Article history:
Received 3 April 2011
Received in revised form
20 June 2011
Accepted 12 July 2011
Available online 26 July 2011

Keywords:
Sketch tools
Recognition algorithms
Sketch recognition
Pen-based interfaces

ABSTRACT

The low accuracy rates of text–shape dividers for digital ink diagrams are hindering their use in real world applications. While recognition of handwriting is well advanced and there have been many recognition approaches proposed for hand drawn sketches, there has been less attention on the division of text and drawing ink. Feature based recognition is a common approach for text–shape division. However, the choice of features and algorithms are critical to the success of the recognition. We propose the use of data mining techniques to build more accurate text–shape dividers. A comparative study is used to systematically identify the algorithms best suited for the specific problem. We have generated dividers using data mining with diagrams from three domains and a comprehensive ink feature library. 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.

1. Introduction

Hand drawn pen and paper sketches are commonplace for capturing early phase designs and diagrams. Pen and paper offers an unconstrained space suitable for quick construction and allows for ambiguity. With recent advances in hardware such as Tablet PC's, computer based sketch tools offer a similar pen-based interaction experience. In addition, these computer based tools can benefit from the ease of digital storage, transmission and archiving. Recognition of sketches can add even greater value to these tools: the ability to automatically identify elements in a sketch allows us to support tasks such as intelligent editing, execution, conversion and animation of the sketches.

Although a number of sketch tools have been developed, they are yet to achieve general acceptance. One of the outstanding challenges is considerably more accurate recognition. Recognition rates from laboratory experiments are typically in the range of 98–99% and above [1–3]. However, rates achieved in less controlled conditions, where data is not limited to produce optimal performance, are usually much lower, for example with the same algorithms, accuracy rates between 84% and 93% are reported in [3–6]. Furthermore, many of these tools are limited as they are not able to distinguish between drawing elements (shapes) and text strokes in a sketch [1–3]. However, most natural diagrams consist of both writing and drawing as shown in Fig. 1.

While recognition of handwriting is well advanced and there have been many recognition approaches proposed for hand drawn sketches, there has been less attention on the division of text and drawing. People can comprehend writing and drawing seamlessly, yet there is a clear semantic divide that suggests, from a computational perspective, it is sensible to deal with them separately. Text–shape division is a difficult problem as there is a large within-class variation compared with other recognition problems. Several recognisers [7–9], commonly referred to as dividers, have been proposed for this purpose, but recognition rates in realistic situations are still unacceptable. 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. This work draws on a more comprehensive set of ink features and employs a range of data mining techniques that have been systematically selected and tuned in a comparative study to select the most accurate model for text–shape division of sketched diagrams.

The remainder of this paper is organised as follows. Related work in sketched diagram recognition, focusing on text–shape division is presented in the next section. The methodology used for this investigation is outlined in Section 3, followed by a description of our ink feature library and the training data used for our analysis. Section 6 presents our analysis using data mining techniques and

* Corresponding author. Tel.: +64 9 836 2636.
E-mail addresses: rpat088@aucklanduni.ac.nz (R. Blagojevic), beryl@cs.auckland.ac.nz (B. Plimmer), jgrundy@swin.edu.au (J. Grundy), yongwang@auckland.ac.nz (Y. Wang).

0097-8493/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2011.07.002

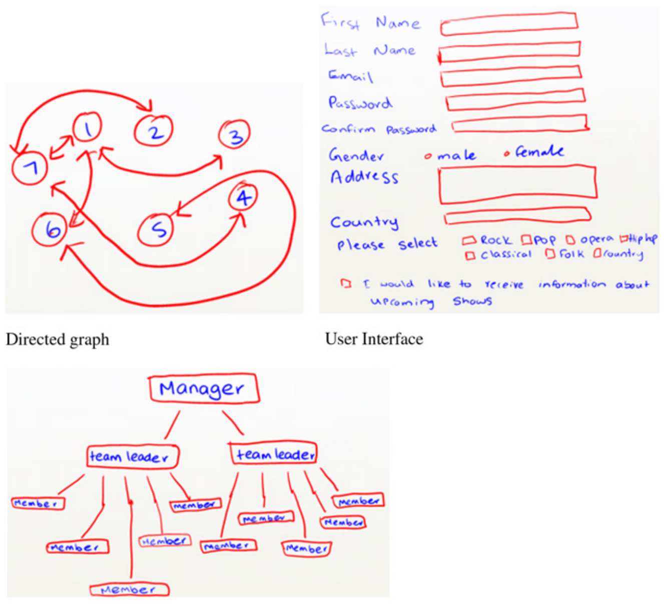


Fig. 1. Training data examples.

the results of the analysis. A comparative evaluation of our best dividers against three existing ones is given in Section 7. We discuss our findings in Section 8 followed by our conclusions and direction for future work.

2. Related work

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.

There are several approaches to sketch recognition. Johnson et al. [10] categorise existing approaches into hard coded algorithms, visual matching, which includes template matching and feature-based recognition, and use of textual descriptions. Due to the large within-class variation in the classes of text and shapes, we believe that feature-based recognition is the most flexible approach and has the most potential for the text–shape divider problem. Feature-based recognition involves the measurement of various ink features and the use of an algorithm to combine these features to produce classifications. Previous text–shape dividers [7–9] have also used a feature-based approach to recognition.

In the area of sketch diagram recognition, many systems focus only on shapes [1–3,11–15]. 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. There have been some attempts at incorporating text–shape division in domain specific recognisers [16,17] and domain independent diagramming tools [18,19]. These systems are predominantly rule-based, using stroke features chosen heuristically to distinguish between text and shapes.

Research in the area of digital ink document analysis for freehand note-taking has explored text and shape division [20–23]. However, as the content of documents is mainly text, these methods hold some bias, which may make them unsuitable for sketched diagrams. In addition, as Bhat and Hammond [7] point out, some of these methods would have difficulty with text interspersed within a diagram. There has also been some work separating Japanese characters from shapes in documents [23,24].

Three reports specifically on domain independent dividers are [7–9]. Although they use a feature-based approach, they have focused their development on one or two algorithms and rely on very limited feature sets.

Bishop et al. [8] developed a feature-based divider that uses local stroke features and spatial and temporal contexts 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. They found that using local features and temporal context were successful. They report classification rates from 86.4% to 97.0% for three classifier model variations.

In our previous work [9] we developed a domain independent divider for shapes and text based on statistical analysis of 46 stroke features. A decision tree was built identifying eight features as significant for distinguishing between shapes and text as shown in Fig. 2. The results on a test set showed an accuracy 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 [25] and InkKit [5,18] dividers, it was able to correctly classify more strokes overall for the test set.

A more recent development in this field has been the use of a feature called Entropy [7] to distinguish between shapes and text. Strokes are first grouped into shapes and words/letters using spatial and temporal proximities. 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 and Hammond [7]) 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 and Hammond [7] and Bishop et al.'s [8] 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

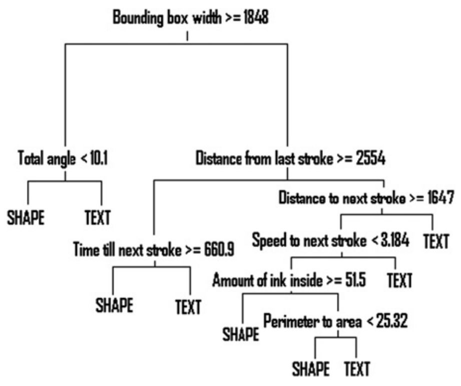

Bounding box width >= 1848
Total angle < 10.1
Distance from last stroke >= 2554
SHAPE
TEXT
Distance to next stroke >= 1647
Time till next stroke >= 660.9
Speed to next stroke < 3.184
TEXT
SHAPE
TEXT
Amount of ink inside >= 51.5
TEXT
SHAPE
Perimeter to area < 25.32
SHAPE
TEXT

Fig. 2. Divider 2007 decision tree produced by Patel et al. [9].

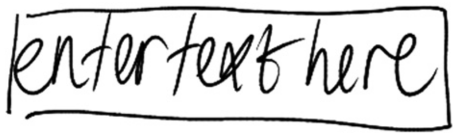


Fig. 3. This diagram shows an example of where stroke grouping could fail.

observed in sketches [26]. We have re-implemented Bhat and Hammond [7] algorithm for our evaluation. However, we do not group strokes as there are many cases where their proposal would fail. For example, in Fig. 3 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. Section 7.1 describes our implementation of this algorithm and our training method as no thresholds are given in [7]. As our evaluation will show, this divider has been trained and tested on limited data and constrained conditions and does not perform at the reported rate of 92.06% on realistic diagrams.

Microsoft has developed recognisers for ink analysis that are now built into its operating systems (from Windows Vista onwards) [27]. 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 [9] we ran a comparative evaluation of an older version of Microsoft's divider [25] 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 [7,8,28] although, judging from the results presented, not to the same extent as the Microsoft divider.

Other recent work in this area [29] has involved using AdaBoost and an extension of our previous feature set [9] to build a text–shape divider as well as grouping strokes after division. Their evaluation shows that it is more accurate than the Microsoft divider but has mixed results against Bhat and Hammond [7] divider.

While others have used various features and algorithms for text–shape division, to our knowledge no systematic analysis of algorithms has been carried out for text–shape division to determine the most suitable algorithm for this problem. We present a comprehensive comparative study of features and algorithms using data mining to select the most accurate model. In particular we are looking at the problem of distinguishing between text and shapes as a first step for recognising sketched diagrams; a fundamental problem required to preserve a non-modal user interface similar to pen and paper.

3. Methodology

This section describes the approach we have used to improve the text–shape dividers for hand-drawn diagrams. We employ data mining techniques to build text–shape dividers. 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 used 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

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. 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 our data collection tool, DataManager [30]. This enables feature measurements to be taken on any diagrams collected with the tool (and other datasets that are converted to DataManager's data format) 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 [31], 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.

3.2. Data collection

Large amounts of data are needed in order to improve the accuracy of sketch recognition algorithms. There is a small amount of digital ink data publicly available [32–34]. However, most of this data does not include text or holds some biases related to the method of data collection, so a new corpus was constructed. Our tool, DataManager [31] was used to collect, label and generate training and testing datasets for our analysis.

From previous experience [29] 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. Further information on the training and testing datasets can be found in Sections 5 and 7.2.

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. DataManager's dataset generator [31] was used here.

Data mining techniques were employed to perform the analysis. Data mining uses machine learning algorithms to search data for patterns [35]; in this case, we are searching for patterns to predict which class a stroke belongs to—text or shape.

Weka [35], 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.

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

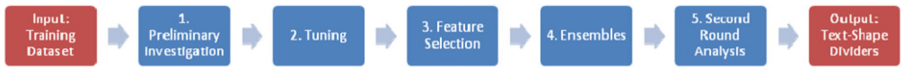


Fig. 4. Process diagram of analysis.

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. We began with a large number of algorithms and systematically reduced this list until the best performing classifiers were found for text–shape division. Taking a standard approach, we used several steps for the analysis process, as shown in Fig. 4.

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 using our dataset. Important parameters were identified so that a range of values could be tested in order to tune each algorithm.

The next stages of analysis involved experimenting with feature selection algorithms and ensembles. Feature selection can reduce the feature library, where non-contributing features are identified and eliminated. The general goal of feature selection is to reduce recognition time, as fewer features are required to be calculated, and to improve accuracy by eliminating bad features. Ensembles combine two or more algorithms into a voting mechanism for classification. Ensembles can result in a higher accuracy than individual algorithms using combinations of strengths and weaknesses that complement each other.

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.

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 [6]. Existing dividers, Entropy [7], Divider 2007 (our divider from previous work) [9] and the Microsoft Ink Analysis Divider [28] 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.

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 improve sketched diagram recognition.

4. Ink features

For recognition to be successful, the features fed into the algorithms must provide good distinguishing characteristics between classes of interest. Without valuable input such as this any classification algorithm would suffer.

Our previous feature set [9] of 46 features has been extended to a more comprehensive library of 114 features for sketch recognition. It has been assembled from previous work in sketch recognition, includes some of our own additions, Entropy [7], and our previous divider [9]. Our previous divider is used for several features: pre-classification of the current stroke, pre-classification of strokes close by (for spatial context), and pre-classification of successive strokes (for temporal context).

In order to better understand the type of things that the features are measuring we have developed the taxonomy shown in Table 1. 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, however it can also be a part of the divider results as it was developed as a one feature divider by Bhat and Hammond [7]. For a more full description of the feature library and taxonomy see [36]. This feature library is also available with full implementation within DataManager [31] from 〈www.cs.auckland.ac.nz/research/hci/downloads〉.

5. Data

For the training set we collected and labelled sketched diagrams from 20 participants using DataManager [31]. Each participant drew three diagrams; a directed graph, organisation chart and a user interface; examples are shown in Fig. 1. We ensured that the data we collected was in the form of full diagrams rather than isolated components, as a previous study [37] showed that this has a significant influence on recognition accuracy. There are a total of 7248 strokes in the training set, with 5616 text strokes and 1632 shape strokes.

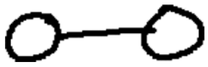
Using this collection of diagrams we generated a dataset of feature vectors for each stroke using DataManager [31]. DataManager's dataset generator function is able to take the diagrams collected and calculate feature vectors based on the implementation of our feature library. DataManager then writes these feature vectors to files compatible with data mining tools for ease of analysis.

6. Data mining analysis

We used several steps to build our text–shape dividers as illustrated in Fig. 4. First we carried out a preliminary analysis of data mining algorithms. We were able to narrow down our study to a small group of algorithms and proceeded to tune these to their optimal parameters. Following this we investigated the use of feature selection and ensembles to improve on our results. Finally we completed a second round of analysis to identify and correct common misclassifications found. This section describes each step of the analysis in more detail.

As mentioned in Section 3.3, Weka (developer version 3.7) [35], an open source data mining tool was used to perform our data analysis, and build, tune and test classifier models for dividers. Weka has a large number of machine learning algorithms, ranging from naïve Bayes, decision trees and rules, to support vector machines and neural nets. These algorithms can be used to perform data mining on

Table 1
Taxonomy of digital ink stroke features.

Category	Number of Features	Description
Curvature	23	These features measure various aspects of a stroke's curvature commonly by calculating angles within the stroke. For example they can show that the line above has a greater curvature than the line below. The curvature of text is often greater than shapes.
Density	8	Density features measure the concentration of points in a stroke. For example the writing has a higher density of points than the rectangle when compared with their bounding box. The density of text is often greater than shapes.
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. The diagram shows intersections at the end and middle of strokes. Text strokes often have more intersections than shapes. 
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. The size of text is usually smaller than shapes.
Spatial context	23	Features measuring spatial context are within the following subcategories: curvature, density, divider results, intersections, location and size. Each subcategory contains measurements for strokes in close proximity to the current stroke. Strokes with similar feature measurements to those that are close by are more likely to be from the same class.
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. Strokes drawn in succession are more likely to be from the same class except when objects are interspersed.
Time/speed	6	These dynamic features include the total, average, maximum and minimum times or speed for a stroke.

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 [38] 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.

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. Several servers were employed for the preliminary investigation and tuning steps. 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 [35]. 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.

6.1. Preliminary analysis

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 [35], 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, although some are able to work well by translating nominal values to numeric. The analysis began with a preliminary investigation of all these algorithms. This involved building classifier models for each algorithm using the training data described in Section 5.

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 improve their results. We were able to narrow the search down to seven algorithms that are likely to gain the best classification accuracy for a divider. When choosing these algorithms we looked at their performance in our preliminary tests; we tried to include a good range of machine learning algorithms; and also consulted with a machine learning expert [38] to help us to identify those that may improve with further tuning.

The chosen classifiers are listed below:

(1) Bagging (with an REPTree base learner) [39];
(2) RandomForest (forest of random trees) [40];
(3) LogitBoost (additive logistic regression) [41];
(4) LADTree (alternating decision tree using the LogitBoost strategy) [42];
(5) LMT (logistic model tree) [43,44];
(6) Multilayer Perceptron (neural network) [45,46];
(7) SMO (support vector machine) [47–49].

The AdaBoost algorithm, used recently by [30] to build a text–shape divider, was included in our preliminary investigation. However, it was found to have a very low accuracy compared with other algorithms tested; it ranked 34th of the 39 algorithms tested.

6.2. Tuning

Using the training dataset of feature vectors generated from the diagrams collected we built dividers by training each classifier. While Weka provides sensible default parameters for most algorithms, some classifiers required tuning to optimise their results for our dataset.

Algorithms are compared here using corrected paired *t*-tests. When using cross validation, samples are not independent. This can cause significant differences to be found by a simple paired *t*-test that do not really exist [35]. Weka's Experimenter interface includes a corrected paired *t*-test that solves this problem by modifying the *t*-statistic used [35] so that significant differences are not over estimated when making comparisons.

6.2.1. Bagging

For our experiments, the Bagging algorithm [39] 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 (a fast decision tree learner) at 10, 100, 500, 1000 and 5000 iterations. Corrected paired *t*-tests ($\alpha=0.05$) showed no significant difference in the results at each level of iterations, so no further tuning was applied. 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 (with a standard deviation for 500 and 1000 iterations of 0.58 and 0.62, respectively).

6.2.2. RandomForest

RandomForest [40] was tuned by varying the number of iterations of the algorithm to 10, 100, 500 and 1000 iterations. This parameter dictates the number of trees that are produced. It was observed that a higher number of iterations produced higher accuracies; therefore additional tests were added with 1500, 2000, 2500, and 3000 iterations. Corrected paired *t*-tests ($\alpha=0.05$) show that there is no significant difference between any of the models. The highest result is produced at 500 iterations where, on average (over ten folds of cross validation), 96.45% (sd: 0.57) of the training dataset is correctly classified into text and shape.

6.2.3. LogitBoost

Two base learners were investigated for LogitBoost [41]: the Decision Stump (an one node decision tree) and the REPTree (a fast decision tree learner). 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 corrected paired *t*-test ($\alpha=0.05$) showed no significant difference between the two at 120 iterations of the algorithm. Based on these results, both trees were further investigated as base classifiers.

To further tune LogitBoost, various values for the number of iterations the algorithm performs and the shrinkage parameter were tested. The number of iterations determines the number of trees that are produced. Unlike other algorithms, LogitBoost can use the shrinkage parameter to avoid over-fitting the 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, and 5000;
- *Shrinkage*: 1.0 (Weka default value) and 0.1.

It was observed from the results of corrected 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, and 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 model with the highest level of accuracy uses a Decision Stump base learner, 5000 iterations, and a shrinkage value of 0.1. On average (over ten folds of cross validation), 96.7% (sd: 0.84) of the training dataset is correctly classified into text and shape. Corrected paired *t*-tests ($\alpha=0.05$) 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. 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 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

We tuned the LADTree [42] by varying the number of iterations of the algorithm to 10, 100, 500, 1000 and 1500 iterations. This parameter determines the size of the LADTree. The number of iterations could not be increased any further due to computational time and memory constraints. Corrected paired *t*-tests

($\alpha=0.05$) 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. The highest result is produced at 1500 iterations where on average 97.48% (standard deviation, sd: 0.52) of the training dataset (using ten-fold cross validation) is correctly divided into text and shapes.

6.2.5. *LMT*

The result of 10-fold cross validation using our training dataset on LMT [43,44] with default parameters is 94.85% (sd: 0.68).

6.2.6. *Multilayer perceptron*

The result of 10-fold cross validation using the training dataset on MultilayerPerceptron [45,46] with default parameters is 95.02% (sd: 0.78).

6.2.7. *SMO*

SMO [47–49] 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. The complexity parameter allows a trade-off between misclassifying some instances by allowing for a wider distance between classes. The gamma parameter of the RBF kernel can be tuned to control the linearity of the mapping done by the RBFKernel. High values of gamma result in almost linear mappings.

To find the best model, the GridSearch function [35] 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).

6.2.8. *Comparison of tuned classifiers*

The best results of ten-fold cross validation for each classifier on the training dataset are shown in Table 2. Corrected 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.

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 recognition time, as a smaller number of feature calculations have to be made. Three methods of feature selection were used: an Attribute Selected Classifier with Wrapper, Attribute Selected Classifier with Relief F and a hybrid method.

6.3.1. *Attribute selected classifier with wrapper*

We chose to use the Attribute Selected Classifier in Weka [35] as this allows feature selection and classifier training to be completed in one process, 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.

In addition, a Wrapper [50] can be used within the Attribute Selected Classifier as the feature selection method. The feature selection method is used to evaluate the accuracy of any feature subset. The wrapper can take any classifier and use it to perform feature selection. The advantage of using the wrapper is that the same machine learning algorithm can be used to evaluate the feature subset and also train the final classifier, therefore we expect good results. For example within the Attribute Selected Classifier we can use the LADTree via the wrapper to find the best feature subset and then we can use this chosen feature subset to train LADTree for our 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 3. The parameters were chosen as optimal from the tests described in Section 6.2.

Table 3
Optimal parameter settings used for feature selection and ensembles.

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 2
Ranking of best results obtained from the tuned classifiers. Those with * are significantly more accurate than the others according to corrected paired *t*-test.

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

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 [51]. 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 et al. [51] report that for problems with few classes, as in our case with two classes, high accuracy is achieved when $k \leq 10$.

We found that all of the Attribute Selected Classifiers were significantly less accurate than the original tuned classifiers without feature selection, according to corrected paired t-tests ($\alpha = 0.05$). The results range from 89.24% (sd: 1.13) to 93.58% (sd: 0.45) of the training dataset correctly classified on average (over 10 folds of cross validation). This suggests that in fact all of the ink features captured do add some value to each classifier. Therefore, reducing the feature set using this feature selection method decreases the accuracy of all of these classification models.

6.3.2. *Attribute selected classifier with Relief F*

In previous work [36] we found Relief F [52–54] to be one of the best of the eight ranking methods tested from Weka's feature selectors. We used the Attribute Selected Classifier 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 [35] to be used. This simply ranks features by evaluating them individually, leaving the overall evaluations to Relief F.

Corrected paired t-tests ($\alpha = 0.05$) show for all classifiers that the original tuned models are not significantly different from the models built with Relief F attribute selection. The results range from 94.85% (sd: 0.68) to 97.12% (sd: 0.54). 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. 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.

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 [55]. 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.

In previous work [36] we conducted a study of numerous feature selection algorithms and found that an optimal number of features is 20. Adding more features does not greatly change the accuracy of an algorithm. 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 the top 20 for both feature sets 1 and 2 are significantly worse than the tuned classifiers with no feature selection according to corrected paired t-tests ($\alpha = 0.05$). 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 for text–shape division.

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 [56,57] 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 corrected paired t-tests on the ten-fold cross validation experiments described in Section 6.2. That ranking is shown in Table 2. Combinations were composed of the top 7 algorithms (all algorithms), the top 6, top 5, top 4, top 3 and top 2. The LogisticRegression [58] model was used as the meta classifier, due to its simplicity. Also, the parameters for each algorithm, obtained from the tuning steps in Section 6.2, have been used here, as described in Table 3.

To further inform the choice of classifier combinations, each individual classifier was tested on another dataset of Entity Relationship (ER) and Process diagrams, obtained from [59], with special attention drawn to their performance on each shape class. Examples from this dataset are shown in Fig. 5. The ER/Process diagram dataset contains a total of 7062 strokes, with 4817 text strokes and 2245 shapes strokes. The results showed that a lower proportion of shapes, between 80% and 89%, are correctly classified than text, which has a correct classification rate between 97% and 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 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 4.

The results show that 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% to 99%. The areas of weakness are in classifying arrows and lines. For arrows, correct classification rates range from a very low 40–64%. 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 overfitted 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. A total of 19 ensemble combinations were tested.

Corrected paired *t*-tests were used to compare the different voting combinations; in particular the ranking function provided by Weka Experimenter [35]. The highest ranked combination is LLSL{LogitBoost, LADTree, SMO, LMT}. The next highest are LLS{LogitBoost, LADTree, SMO} and LadSR{LADTree, SMO, RandomForest}, followed by LLSR{LogitBoost, LADTree, SMO, RandomForest}. 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 4. However, corrected 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 4 show that the greatest area of weakness for all classifiers trialled is with arrows, where recognition rates range from 40% to 64%. 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. They are described in Table 5. These features were found after the initial set had been compiled. The fourth feature, Is Arrowhead, is unique because it acts as 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. The second parse can be run after the use of any classifier model to refine the results. This particular feature was built to identify arrowheads that were misclassified as text by the initial classifier.

The extended feature set, with additional features 1–3 described in Table 5, 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 3 were used for each classifier. The results of ten-fold cross validation are shown in Table 6.

In addition, an LADTree at 1500 iterations was trained, as this produced a higher result for our original feature set. The results of ten-fold cross validation for this LADTree are also shown in Table 6.

Corrected paired *t*-tests show that the LADTree with 1500 iterations is significantly more accurate than all other models. Corrected 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 6) 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 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.

Four conditions are included in the test to highlight the effect of using the extended feature set and second parse feature.

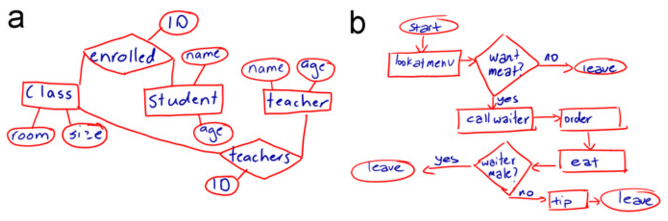


Fig. 5. (a) ER and (b) process diagram test set examples.

Table 4
Summary of testing tuned models with ER/Process diagrams.

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 5
Second round features.

Feature	Description	Origin
(1) Is straight line	This feature identifies straight lines. A line segment is made from the point 1/3 of the way into the stroke to the point 2/3's into the stroke. 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.	[60]
(2) Number of cups	This feature identifies 'U' shapes in strokes, known as cups. 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. 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.	[60]
(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 [9,29] by Garcia [61]. 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. This feature may help solve the problem of identifying vertical connectors.	[61]
(4) Is arrowhead	This feature is used as a second parse of a sketch to determine if a stroke is an arrow. It assumes the arrow is drawn in two strokes: one stroke for the arrowhead and the other for its shaft and that the arrow shaft has already been correctly classified. Firstly, the arrowhead is found by determining if the stroke is in two fragments. Next, the arrow shaft is found with a search for the closest shape stroke to the possible arrowhead. 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. Lastly, the ratio of the length of arrowhead to the whole arrow length must be less than 40%.	[62,63]

Table 6
Results of using 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)

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 7.

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 [64]. The tests show that there is no significant difference between the highest and lowest results for arrows (p-value: 0.08, sd: 0.03), rectangles (p-value: 0.50, sd: 0.01), diamonds (p-value: 0.54, sd: 0.01), ellipses (p-value: 0.42, sd: <0.01), lines (p-value: 0.11, sd: 0.01) and shapes overall (p-value: 0.19, sd: 0.01). 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. For text, z-tests consistently show that the conditions without feature 4 are significantly more accurate than those with feature 4 (p-values ranging from 9.32×10^{-15} to 1.06×10^{-13}, sd: <0.01). In addition, the conditions without feature 4 are not significantly different from each other (p-value: 0.74, sd: <0.01) and the conditions with feature 4 are also not significantly different from each other (p-value: 0.88, sd: <0.01).

A similar pattern is evident in the results for the total percentage of strokes classified correctly, as shown by the p-values in Table 8. This parallel may be because 68% of the total ER/Process diagrams dataset are text strokes.

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 7 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.

6.6. Analysis 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 arrows and lines, 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 (Vote 1) and LLS (Vote 2), 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.

7. Evaluation

The goal of our research is to improve recognition of hand-drawn diagrams through the development of more accurate text–shape dividers using data mining. A systematic investigation of

Table 7
Results of second round features to the original feature set on er/process diagrams dataset with the LADTree (1500 Iterations).

	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 8
P-values for the total % of strokes correctly classified under four conditions (sd: <0.01).

	Original	Extended	Original + feature 4	Extended + feature 4
Original		0.94	<0.01	<0.01
Extended			<0.01	<0.01
Original + feature 4				0.88
Extended + feature 4				

Table 9
New dividers chosen for the evaluation.

Divider	Configuration
LADTree 1	LADTree with 1500 iterations.
LogitBoost	LogitBoost with Decision Stump, shrinkage = 0.1 and 5000 iterations.
Vote 1	Ensemble classifier LLSL with LogitBoost, LADTree, SMO and LMT (see Table 3 for individual algorithm configurations).
Vote 2	Ensemble classifier LLS with LogitBoost, LADTree and SMO (see Table 3 for individual algorithm configurations).
LADTree 2	LADTree with 1500 trained with 2nd round features including second parse

data mining techniques, using a comprehensive ink feature library, has resulted in the construction of several dividers. To evaluate whether or not the 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 this section we describe the implementation of each of the dividers included in the evaluation, the test datasets used for the evaluation and present the final results.

7.1. Divider implementation

In order to run a comparative evaluation of our new models against other dividers we integrated our models into DataManager's Evaluator [6]. 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.

Each of our new dividers (listed in Table 9) were generated and output into .model files using Weka's Explorer interface [35]. 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 [65], 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.

In addition to the new divider models, three existing dividers were integrated: Divider 2007 (from our previous work) [9], the Microsoft Ink Analysis divider [28], and the Entropy divider [7].

Divider 2007 [29] was not re-trained; it was implemented with the same thresholds as the original decision tree shown in Fig. 2. The implementation is in C# .NET so the integration into the Evaluator was straightforward.

Microsoft Ink Analysis [28] is able to distinguish between shapes and handwriting. 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.

When implementing the Entropy divider, it had to be trained as no thresholds were provided by Bhat and Hammond [7]. It was trained on the same training dataset as the new dividers, described in Section 5. The Decision Stump algorithm from Weka [35] 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 [66], a rule based method, and a J48 tree (C4.5 decision tree) [67] gave similar results. We believe this is less than Bhat and Hammond [7] published result because of the training data used.

7.2. Test data

Several datasets were gathered for testing the text–shape dividers. The datasets represent diagrams from various domains. These datasets were intentionally chosen to 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 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. A summary of each dataset used for testing the text–shape dividers is shown in Table 10 and examples of each dataset are shown in Fig. 6.

7.3. Evaluation 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.

Table 10
Summary statistics for each test dataset.

	# 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	[7]
Logic	13	2296	6320	8616	27:73	[34]
Euler	10	413	334	747	55:45	[68]
Total		8230	7817	16,047	51:49	

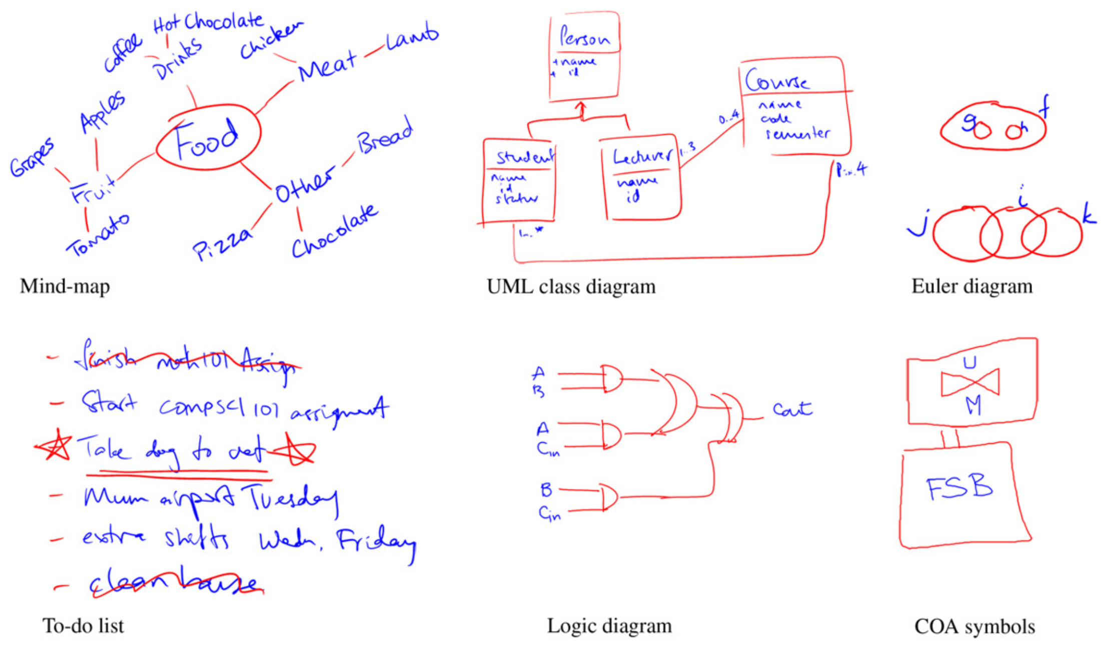


Fig. 6. Test data examples. Shape strokes are shown in red and text strokes in blue. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Tukey's confidence intervals [69] are used here as a method of performing multiple statistical 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 highly accurate.

A graph displaying Tukey's confidence intervals for each dataset and divider is shown in Fig. 7. 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.

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 Fig. 7 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.

The overall accuracy of each divider is calculated in two ways: using a simple average and a weighted average. They are shown in Table 11. 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 Eq. (1). All datasets are given the same weight using this average, regardless of their size.

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 Eq. (2). The weight for a dataset should be proportional to the inverse variance of the mean [70]. The variance is s^2/n: therefore the inverse variance is used as the numerator of the weight equation shown in Eq. (2). 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

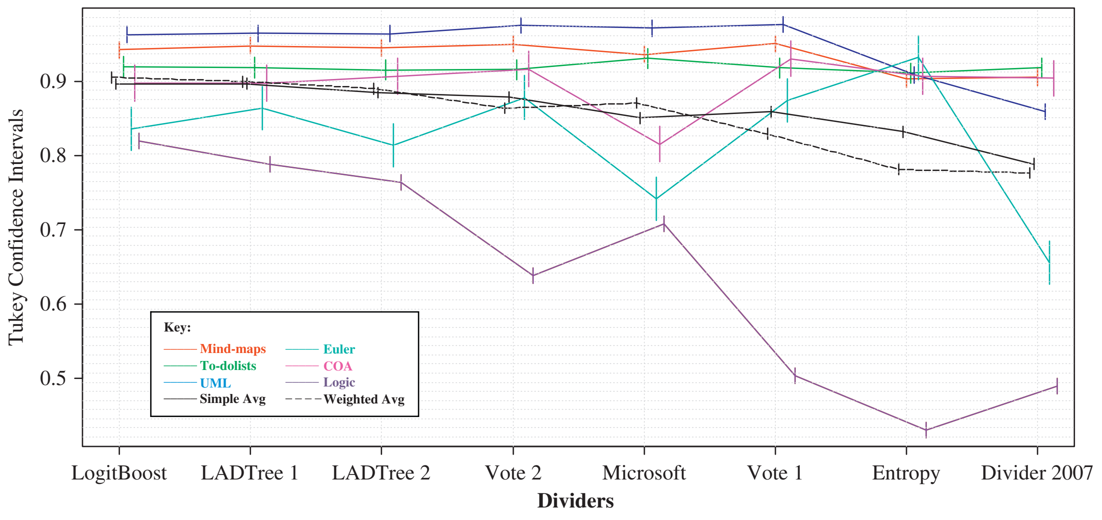


Fig. 7. All results displayed as Tukey's confidence intervals.

Table 11
Simple and weighted averages of classification rates for each divider.

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

multiplied by the weight value for each dataset.

$$\text{simple average} = \sum p/\text{number of datasets} \quad (1)$$

where p is the proportion of strokes correct for each dataset

Eq. (1) **Simple Average Calculation for a Divider's Accuracy**

$$\text{weight} = \frac{n/s^2}{\sum(n/s^2)} \quad \text{weighted average} = \sum(p*\text{weight}) \quad (2)$$

where: n is number of instances in each dataset; s is within group standard deviation; p is the proportion of strokes correct the each dataset; and Σ is the sum overall values for the divider on each dataset

Eq. (2) **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 Eq. (2). 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 Fig. 7 to assist us in analysing the overall significant differences between dividers. Table 12 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).

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 12 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

7.3.1. *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 13 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 1–8 feature

Table 12
Summary of the significant differences shown by Tukey's confidence intervals in Fig. 7. √ significantly more accurate, X significantly less accurate, – not significantly different.

	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 13
Average time (seconds) to classify a stroke.

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

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. 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.

8. Discussion

Dividing text and shapes in sketches is a difficult problem. This is mainly due to the large variation present within each class of text and shape. Previous work in developing dividers is limited as they focused on a small number of features and algorithms.

We have completed a systematic study of a wide range of data mining techniques and have found this to be successful in terms of producing more accurate dividers. 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.

We were surprised by the lack of improvement in accuracy when using feature selection and ensembles. There are several possible reasons for this. 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.

When developing Divider 2007 [29], 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 11). 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 11). 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.

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. [71] 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.

Another possible reason for the low performance of feature selection and ensembles may be because the parameter configurations for the classifiers were set to those with the fast training times rather than the highest accuracy due to long training times. However, the accuracy of the classifiers with the chosen parameter configurations were not significantly different to the classifiers with the highest classification rates based on the training data; the one exception was for LADTree. These settings still may have had a negative effect on the accuracy obtained by feature selection with these classifiers.

Chang [72] 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. 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 inner 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.

The use inner voting mechanisms in our chosen algorithms may have also meant that using outer ensemble combinations had no significant effect on the accuracy level already obtained.

Others [71,72] 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 [72], the number of classes in each test dataset range from three to six basic shapes. Schmieder's study [59] 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.

The high accuracy of the results we have obtained by combining data mining techniques with our extensive feature library to build dividers demonstrates the effectiveness of this approach. We believe that other digital ink recognition problems would also benefit from a similar study of data mining techniques.

We chose to train and test on diagrams of different domains to create a general diagram divider. Each diagram domain has its own syntax, semantics and mix of drawing shapes. Given the difference between the training 10-fold validation values and the test results, it may be worthwhile to data mine and train a divider for each diagram domain or include a much wider range of diagrams in the training set. In addition, we could also look specifically at dividing documents. The Microsoft divider may perform better on documents as there is a higher proportion of text present.

Considering diagram and notes recognition from a wider perspective there are three possible approaches to the recognition: bottom-up, top-down or a combination of both. A bottom-up approach begins the recognition process at the primitive stroke level. Typically this is 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. We believe this approach has better potential for eager recognition—i.e. one does not need to wait for the diagram to be complete for recognition. We believe accurate, eager recognition is a precursor to a better user experience, for example intelligent editing support for sketch tools.

We believe 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. The use of shape contexts [73] and Image Deformation Model (IDM) features [74] may also add valuable information to the recognition process. These additions provide direction for future work.

9. Conclusion

We have assembled a comprehensive library of computable ink features; they measure features of individual strokes and relationships between strokes that are spatially or temporally adjacent. Using these features and diagrams from three different domains we undertook an extensive review of data mining algorithms. Seven of the most effective algorithms were selected for further tuning, which improved their performance. We then explored whether limiting the number of features by using feature selection could produce comparable results and found that reducing the feature set reduced performance. We also explored combining algorithms in an ensemble with voting strategies, this provided no significant gain. In addition, we performed a second round of analysis to try to correct common misclassifications.

Our two best dividers, LADTree and LogitBoost, are significantly more accurate than all other algorithms evaluated. A comparative evaluation of these dividers against three others shows that the new dividers outperform the others with a clear statistically significant difference. The success of our new dividers demonstrates the effectiveness of considering a wide range of ink features and using data mining techniques for sketch recognition development.

Acknowledgements

Thanks to Associate Professor Eibe Frank for expert advice on using data mining techniques. This research is partly funded by Microsoft Research Asia and Royal Society of New Zealand, Marsden Fund.

References

[1] Rubine DH. Specifying gestures by example. In: Proceedings of Siggraph '91, ACM; 1991.
[2] Paulson B, Hammond, T. PaleoSketch: accurate primitive sketch recognition and beautification. In: Proceedings of Intelligent User Interfaces (IUI '08), ACM Press, New York, USA; 2008.
[3] Wobbrock JO, Wilson AD, Li Y. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In: Proceedings of the User Interface Software and Technology. ACM, Newport, Rhode Island, USA; 2007. p. 159–68.
[4] Plimmer B. Using shared displays to support group designs; a study of the use of informal user interface designs when learning to program, In: Computer Science, University of Waikato; 2004.
[5] Young M. InkKit: The Back End of the Generic Design Transformation Tool, Computer Science, University of Auckland, Auckland; 2005.
[6] Schmieder P, Plimmer B, Blagojevic R. Automatic evaluation of sketch recognizers. In: Proceedings of the Sketch Based Interfaces and Modelling, New Orleans, USA; 2009.
[7] Bhat A, Hammond T. Using entropy to distinguish shape versus text in hand-drawn diagrams. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI '09), Pasadena, California, USA; 2009.
[8] Bishop CM, Svensen M, Hinton GE. Distinguishing text from graphics in on-line handwritten ink. In: Proceedings of the Ninth International Workshop on Frontiers in Handwriting Recognition, IEEE Computer Society; 2004. p. 142–7.
[9] Patel R. et al. Ink features for diagram recognition. In: Proceedings of the Fourth Eurographics Workshop on Sketch-Based Interfaces and Modeling. Riverside, California, Eurographics; 2007.

[10] Johnson G, et al. Computational Support for Sketching In: Design: A Review. Foundations and Trends in Human–Computer Interaction 2009;2(1):1–93.
[11] Fonseca MJ Pimentel Ce Jorge JA. CALI: an online scribble recogniser for calligraphic interfaces. In: Proceedings of the AAAI Spring Symposium on Sketch Understanding: IEEE; 2002.
[12] Yu B, Cai S. A domain-independent system for sketch recognition. In: Proceedings of the First International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia. Melbourne, Australia, ACM Press; 2003.
[13] Leung WH, Chen T. User-independent retrieval of free-form hand-drawn sketches. In: Proceedings of the Acoustics, Speech, and Signal Processing, 2002 (ICASSP '02). Orlando, Florida; 2002.
[14] Szummer M, Qi Y. Contextual recognition of hand-drawn diagrams with conditional random fields. In: Proceedings of the Ninth International Workshop on Frontiers in Handwriting Recognition (IWFHR); 2004.
[15] Qi Y, Szummer M, Minka TP. Diagram structure recognition by Bayesian conditional random fields. In: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05). Vol. 2, IEEE Computer Society; 2005. pp. 191–6.
[16] Lank E, Thorley JS, Chen SJ-S. An interactive system for recognizing hand drawn UML diagrams. In: Proceedings of the Centre for Advanced Studies on Collaborative research. IBM Press, Mississauga, Ontario, Canada; 2000. p. 7.
[17] Hammond T, Davis R. Tahuti: a geometrical sketch recognition system for UML class diagrams. In: Proceedings of the 2002 AAAI Spring Symposium on Sketch Understanding; 2002.
[18] Plimmer B, Freeman I. A toolkit approach to sketched diagram recognition. In: Proceedings of the HCI. Lancaster, UK, eWiC; 2007.
[19] Zeleznik RC, et al. Lineogrammer: creating diagrams by drawing. In: Proceedings of User interface Software and Technology. ACM, Monterey, CA, USA; 2008. pp. 161–70.
[20] Shilman M, et al. Discerning structure from freeform handwritten notes. In: Proceedings of the Document Analysis and Recognition; 2003.
[21] Jain AK, Namboodiri AM, Subrahmonia J. Structure in on-line documents. In: Proceedings of the Sixth International Conference on Document Analysis and Recognition, IEEE Computer Society; 2001. pp. 844–8.
[22] Ao X, et al.. Structuralizing digital ink for efficient selection. In: Proceedings of the 11th International Conference on Intelligent User Interfaces. ACM, Sydney, Australia; 2006. pp. 148–54.
[23] Machii K, Fukushima H, Nakagawa M. On-line text/drawings segmentation of handwritten patterns. In: Proceedings of the Document Analysis and Recognition. Tsukuba Science City, Japan; 1993.
[24] Mochida K, Nakagawa M. Separating drawings, formula and text from free handwriting. In: Proceedings of the International Graphonomics Society (IGS2003). Scottsdale, Arizona; 2003.
[25] Microsoft Corporation, Microsoft Windows Tablet, XP. PC Edition Software Development Kit; 2005.
[26] Sezgin M, Davis R. Temporal Sketch Recognition in Interspersed Drawings. In: Proceedings of the Sketch Based Interfaces and Modeling (SBIM '07). Riverside, California, USA; 2007.
[27] Microsoft Corporation. Ink Analysis Overview. 2008 [cited 2008; Available from: 〈http://msdn.microsoft.com/en-us/library/ms704040(VS.85).aspx〉.
[28] Patel R. Exploring better techniques for diagram recognition. MSc in Computer Science, University of Auckland, Auckland; 2007. p. 146.
[29] Peterson E, et al. Grouping strokes into shapes in hand-drawn diagrams. In: Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI-10); 2010.
[30] Blagojevic R, et al. A Data Collection Tool for Sketched Diagrams. in Sketch Based Interfaces and Modeling. Annecy, France: Eurographics; 2008.
[31] Glaser BG, Strauss AL. The Discovery of Grounded Theory: Strategies for Qualitative Research. Chicago: Aldine Publishing Company; 1967.
[32] Oltmans M, Alvarado C, Davis R. ETCHA sketches: lessons learned from collecting sketch data. In: Proceedings of the AAAI Fall Symposium on Making Pen-Based Interaction Intelligent and Natural; 2004.
[33] Hse H, Newton AR. Sketched symbol recognition using Zernike moments. In: Proceedings of the International Conference on Pattern Recognition. Cambridge, UK; 2004.
[34] Alvarado C, Lazzareschi M. Properties of real world digital logic diagrams. In: Proceedings of the First International Workshop on Pen-based Learning Technologies. Catania, Italy; 2007.
[35] Witten IH, Frank E. Data Mining: Practical machine learning tools and techniques.2nd Edition ed. San Francisco: Morgan Kaufmann; 2005.
[36] Blagojevic R, Chang SH-H, Plimmer B. The power of automatic feature selection: rubine on steroids. In: Proceedings of the Sketch Based Interfaces and Modeling. Annecy, France, Eurographics; 2010.
[37] Field M, et al. The effect of task on classification accuracy: Using gesture recognition techniques In: free-sketch recognition. CAD/GRAPHICS 2009 2009;34(5):499–512.
[38] Frank E. Personal Communication. Hamilton: University of Waikato; 2009–2010.
[39] Breiman L. Bagging predictors. Machine Learning 1996;24(2):123–40.
[40] Breiman L. Random forests. Machine Learning 2001;45(1):5–32.
[41] Friedman J, Hastie T, Tibshirani R. Additive logistic regression: a statistical view of boosting. The Annals of Statistics 2000;28(2):337–407.
[42] Holmes G, et al. Multiclass Alternating Decision Trees. In: Proceedings of the Machine Learning, ECML 2002. Springer Berlin/Heidelberg; 2002. p. 105–22.
[43] Landwehr N, Hall M, Frank E. Logistic model trees. Machine Learning 2005;59(1–2):161–205.
[44] Sumner M, Frank E, Hall M. Speeding up Logistic Model Tree Induction. In: Proceedings of the Ninth European Conference on Principles and Practice of Knowledge Discovery in Databases. Porto, Portugal, Springer; 2005.
[45] Rumelhart DE, Hinton GE, Williams RJ. Learning internal representations by error propagation. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, ed. D.E.a.M. In: Rumelhart JL, editor. Foundations, vol. 1. Cambridge: MIT Press; 1986.
[46] Minsky M, Papert. S. Perceptrons. Cambridge: MA.MIT Press; 1969.
[47] Hastie T, Tibshirani R. Classification by pairwise coupling. In: Proceedings of the Advances in Neural Information Processing Systems. Denver, Colorado, United States MIT Press; 1998.
[48] Keerthi SS, et al. Improvements to Platt's SMO algorithm for SVM classifier design. Neural Computation 2001;13(3):637–49.
[49] Platt J. Fast training of support vector machines using sequential minimal optimization. In: Proceedings of the Advances in Kernel Methods—Support Vector Learning; 1999.
[50] Kohavi R, John GH. Wrappers for feature subset selection. Artificial Intelligence 1997;97(1–2):273–324.
[51] Gütlein M, et al. Large-scale attribute selection using wrappers. In: Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining. IEEE; 2009.
[52] Robnik-Sikonja M, Kononenko I. An adaptation of Relief for attribute estimation in regression. In: Proceedings of the Fourteenth International Conference on Machine Learning; 1997. pp. 296–304.
[53] Kononenko I. Estimating attributes: analysis and extensions of RELIEF. In: Proceedings of the European Conference on Machine Learning. Catania, Italy; 1994.
[54] Kira K, Rendell LA. A Practical Approach to Feature Selection. In: Proceedings of the Ninth International Workshop on Machine Learning. Aberdeen, Scotland, United Kingdom: Morgan Kaufmann Publishers, San Francisco, CA; 1992.
[55] Paulson B, et al. What!?! No Rubine features?: using geometric-based features to produce normalized confidence values for sketch recognition In: Proceedings of the VL/HCC Workshop: Sketch Tools for Diagramming. Herrsching am Ammersee, Germany; 2008.
[56] Kuncheva LI. Combining Pattern Classifiers: Methods and Algorithms. John Wiley and Sons, Inc.; 2004.
[57] Kittler J, et al. On combining classifiers. IEEE Transactions on Pattern Analysis and Machine Intelligence 1998;20(3):226–39.
[58] Le Cessie S, Van Houwelingen JC. Ridge Estimators in Logistic Regression. Applied Statistics 1992;41(1):191–201.
[59] Schmieder P. Comparing basic shape classifiers: a platform for evaluating sketch recognition algorithms. In Computer Science, University of Auckland, Auckland; 2009 p. 177.
[60] Willems D, Niels R. Definitions for Features Used in Online Pen Gesture Recognition. NICI, Radboud University Nijmegen; 2008.
[61] García Martín-Mantero Y. Editor gráfico de diagramas de clases basado en trazos naturales. Spain: UCLM (University of Castilla La Mancha); 2010.
[62] Freeman I, Plimmer B. Connector semantics for sketched diagram recognition. In: Proceedings of the AUIC. Ballarat, Australia. ACM; 2007.
[63] Kara LB, Stahovich TF. Hierarchical parsing and recognition of handsketched diagrams. In: Proceedings of the UIST '04. Santa Fe, New Mexico: ACM Press; 2004.
[64] LeBlanc DC. Statistics: Concepts and Applications for Science. Jones & Bartlett Publishers; 2004. 382.
[65] IKVM.NET. 2009 [cited 6/15/2009; Available from: 〈http://www.ikvm.net/〉.
[66] Holte RC. Very simple classification rules perform well on most commonly used datasets. Machine Learning 1993;11(1):63–90.
[67] Quinlan R. C4.5: programs for machine learning. Machine Learning 1993;16(3):235–40.
[68] Delaney A, et al. Recognizing sketches of Euler diagrams drawn with ellipses. In: Proceedings Sixteenth International Conference on Distributed Multimedia Systems. Chicago, USA; 2010.
[69] Ott RL, Longnecker MT. 5 ed.An Introduction to Statistical Methods and Data Analysis, 1184. Duxbury Press; 2000.
[70] Snedecor GW, Cochran WG. Statistical methods. Iowa: Blackwell Publishing Professional; 1989.
[71] Tumen RS, Acer ME, Sezgin TM. Feature extraction and classifier combination for image-based sketch recognition, In: Joint Session of the Seventh Sketch-Based Interfaces and Modeling Workshop and Eighth Symposium on Non-Photorealistic Animation and Rendering, Eurographics Association. Annecy, France; 2010. pp. 63–70.
[72] Chang SH-H. Applying data mining for the recognition of digital ink strokes. In Computer Science, University of Auckland, Auckland; 2010. p. 176.
[73] Oltmans M. Envisioning sketch recognition: a local feature based approach to recognizing informal sketches. Cambridge, MA, USA: Massachusetts Institute of Technology; 2007.
[74] Ouyang TY, Davis R. A visual approach to sketched symbol recognition, In: Proceedings of the 21st International Joint Conference on Artifical intelligence. Morgan Kaufmann Publishers Inc. Pasadena, California, USA; 2009. pp. 1463–8.