EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling (2010)
Marc Alexa & Ellen Yi-Luen Do (Guest Editors)

Rata.SSR: Data Mining for Pertinent Stroke Recognizers

Samuel Hsiao-Heng Chang, Beryl Plimmer and Rachel Blagojevic

University of Auckland, Computer Science, Auckland, NZ

ABSTRACT

While many approaches to digital ink recognition have been proposed, most lack flexibility and adaptability to provide acceptable recognition rates across a variety of problem spaces. Time and expert knowledge are required to build accurate recognizers for a new domain. This project uses selected algorithms from a data mining toolkit and a large feature library, to compose a tailored software component (Rata.SSR) that enables single stroke recognizer generation from a few example diagrams. We evaluated Rata.SSR against four popular recognizers with three data sets (one of our own and two from other projects). The results show that it outperforms other recognizers on all tests except recognizer and data set pairs (e.g. PaleoSketch recognizer and PaleoSketch data set) – in these cases the difference is small, and Rata is more flexible. We hence demonstrate a flexible and adaptable procedure for adopting existing techniques to quickly generate accurate recognizers without extensive knowledge of either AI or data mining.

Categories and Subject Descriptors (according to ACM CCS): I.7.5 [Document and Text Processing]: Graphics recognition and interpretation, I.2.10 [Artificial Intelligence]: Shape, I.5.2 [Pattern Recognition]: Classifier design and evaluation.

1. Introduction

Accurate recognition of hand-drawn input is a basic requirement for computer-based sketch tools to reach their potential. Many of the existing recognizers work well for the specific context for which they were designed, but lack flexibility and extensibility. This project examines existing data mining techniques build within WEKA [HFH*09] to compose context specific recognizers. Our particular interest is in diagrams; however there are many different types of diagrams each with different types of components, and syntactic and semantic rules. Our approach is to automatically compose an accurate stroke (gesture) recognizer from examples. This allows accurate stroke recognition results to be passed to the other parts of the recognition process such as joining related strokes and deciphering semantics.

One of the main approaches to ink recognition is to compute features of the ink and use these features to discriminate between different classes of strokes. Many research projects use this approach and improve the accuracy by selecting features and fixing the threshold of each feature statistically [PPGI07] or heuristically [YC03]. While such an approach can improve results for a particular context, it is time consuming and the resulting recognizer is inflexible. These 'hard coded' [JGHD09] recognizers require significant work to recognize new shapes, because appropriate ink features and new thresholds need to be found manually. As there are essentially an infinite number of diagrams, designing a recognizer for each is impractical.

An alternative approach is to support a range of common shape types to allow more flexibility [FPJ02, PH08]; however, including shapes that are not required is likely to reduce accuracy [FPJ02]. Yet another approach is template matching, where processed input strokes are compared with given templates on the pixel data to find the similarities [Gro94, KS04, WWL07]. New shapes can be simply added by specifying new templates. However because this approach relies mainly on the pixel data, it does not fully exploit the rich temporal data contained in digital ink.

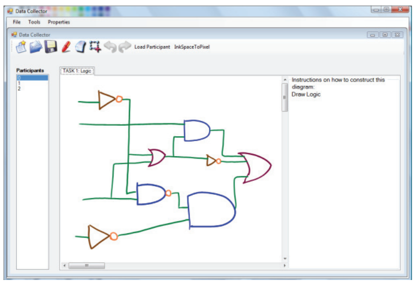
Machine learning techniques that automatically find relationships between features can result in extensible recognizers which are capable of utilizing rich feature sets [Rub91, WNGV09]. This is a promising approach that avoids the disadvantages of hard-coded and pattern matching recognizers. However, there are two major limitations to current diagram recognition research using this approach: first, the number of features used in each project has been limited; second, there is no guarantee that the machine learning algorithms employed are the best because most projects have focused on one or two algorithms.

In this project we use WEKA [HFH*09], a data mining tool which provides many data mining algorithms, to explore the performance of different algorithms using a large set of computable ink features. A set of well performing algorithms were tuned to their best configurations and from this set four algorithms are combined in an ensemble to provide an accurate, trainable recognizer. The recognizer is packaged as Rata.SSR, to allow non-experts to generate single stroke recognizers through a user interface in DataManager [BPGW08]. The generated recognizers can then be used in other software applications via a simple API. We have deliberately scoped this project to simple gestures, ignoring joining and/or splitting which are a usual part of basic shape recognizers, so that we can accurately measure the single stroke recognition success rates.

The rest of the paper is organized as follows. First a small scenario of use describes how a user would build a recognizer with Rata.SSR. Section 3 summarizes related literature on diagram ink recognition. Section 4 presents the construction of Rata.SSR. We then present the result of our evaluation. Finally the paper ends with the discussion and conclusions.

2. Scenario of Use

We will start with a small example to show how a single stroke recognizer for any type of diagram can be quickly generated without much effort. Imagine one wants a logic diagram recognizer. The first step is to collect a number of example sketches in DataManager [BPGW08], as shown in Figure 1. Approximately 10 examples of each class are required – this can be achieved by asking four people to each draw a diagram that includes three examples for each class. The strokes in these sketches must then be labeled with their shape class name. From this a feature data set is generated by DataManager using the labeled strokes.



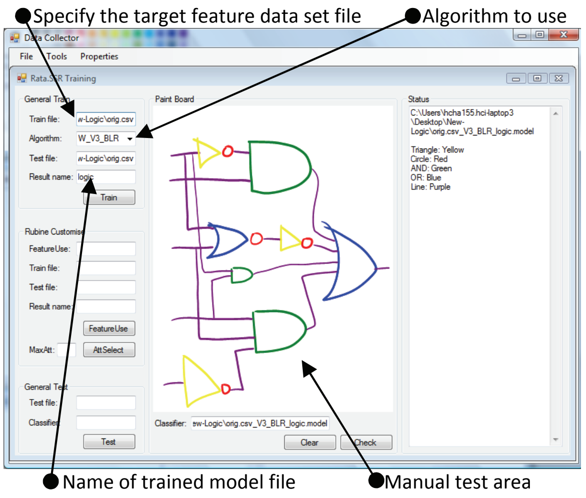
Instructions on how to construct this diagram:
Draw Logic

Figure 1: Sample logic diagram collected in DataManager

Using the interface in Figure 2 the user selects the feature data set file, the algorithm (the default is recommended) and a place to save the recognizer model file. The generation of the model takes just a few minutes. It can then be checked in the manual test panel. To incorporate the model into a program, the DLLs, a model file and just two lines of code are required, (1) to load the model and (2) to recognize a stroke – there are several overloads for the classify method to cater for different requirements.

```
c = ClassifierCreator.GetClassifier("...//rata.model");  (1)
string result = c.classifierClassify(targetStroke);        (2)
```

Our experiments presented below show that one can expect recognition rates over 95% for a wide range of different types of diagram using Rata.SSR as simply as we have described above.



Train file: w-Logic\orig.csv
Algorithm: W_V3_BLR
Test file: w-Logic\orig.csv
Result name: logic
FeatureUse:
Train file:
Test file:
Result name:
MaxAtt:
Test file:
Classifier:
Classifier: ew-Logic\orig.csv_V3_BLR_logic.model

C:\Users\hcha155.hci-laptop3
\Desktop\New-
Logic\orig.csv_V3_BLR_logic.model

Triangle: Yellow
Circle: Red
AND: Green
OR: Blue
Line: Purple

Figure 2: DataManager interface to Rata.SSR

The design is flexible. New computable ink features can be added to DataManager. Instead of creating more shape classes than necessary, for each given problem the recognizer is trained for only the required shape classes. Furthermore, because the process is done by data mining, very little human work is required. In addition, the adoption of WEKA allows extendable algorithms from data mining experts, and optimizing opportunities for expert users.

3. Related Work

Obtaining information about the digital ink strokes is always the key to recognition; the way this information is deduced decides the mechanism used in these systems. Some use similar approaches to image processing, matching shape templates against input, for instance [Gro94, KS04, WWL07]. Others hard-code the threshold of ink features for each shape, such as [CSKK02, FPJ02, SSD01, YC03]. While a third group also use features, they combine the features with machine learning algorithms to train the recognizers, for example [FPJ02, HEP*08, Rub91].

Template matching approaches are similar to many image processing techniques: a user creates a number of example shapes which are used as templates. These templates are constructed by standardizing the number of points, rotating to a standard position, and scaling to a standard size. Data to be recognized is manipulated using the same process and then pixel matched to the templates. Examples of recognizers using this approach are [Gro94, KS04, WWL07]. Although these recognizers are extendable, they do not utilize the full information of the ink strokes and the rotation makes it difficult to differentiate some classes, for example rectangles from diamonds. Reported recognition rates vary from 87% [KS04] to 99.02% [WWL07].

Both hard-coded and trainable recognizers use computable features of the ink (such as stroke length and curvature), as opposed to pixel data used by template matchers. Hard-coded recognizers [FPJ02] apply fixed thresholds to the different features to differentiate the classes of interest. For example a closed shape such as a circle would have the start and finish points 'close together'. The thresholds have been arrived at either heuristically [SSD01, YC03] or by statistical methods [FPJ02, PPGI07]. While such an approach is effective in distinguishing lines and arcs in segmentation problems [CSKK02, SSD01, YC03], they have limited flexibility for application to more complex recognition tasks. Systems built this way are cost ineffective: a lot of effort is required to extend the number of supported shapes [AVK93, FPJ02].

Training based approaches support flexibility by converting ink data to features, and applying machine learning techniques to find relationships [FPJ02, HEP*08, Rub91]. Although many training approaches exist, few have applied a large number of features, except [WNGV09] which has 758 features (although these are mostly recombination of a base set of 48 features). The other part of the training-based approach is the machine learning algorithms employed. Most studies have reported on the use of one or two algorithms. Reported recognition rates for trainable recognizers are in the range 95.1% [FPJ02] to 99.2% [WNGV09]. This approach is promising, yet not fully explored. A similar area, multi-media machine learning, reports many successful applications. They suggest the application of rich feature sets [BSS04, VA05] and comparing different algorithms to rank the effectiveness of each [CSJ00, LZ04, Tay08].

Data mining tools such as WEKA [HFH*09] and RapidMiner [Rap10] provide many data mining algorithms. A large variety of features have been used in different ink recognition projects, and most are assembled into the feature library in DataManager [BPGW08]. In this project we combine the feature library of DataManager with the algorithms in WEKA to compose a fully trainable recognizer that performs as well as the best of other techniques, yet is a good deal more flexible.

4. Our approach

In order to explore multiple recognition algorithms and strategies, for the purposes of combining these into an ensemble, we have used the WEKA data mining tool [HFH*09]. First the full range of suitable WEKA algorithms was trialed and a set of well performing algorithms were selected for optimization (section 4.2). The optimization process, explained in section 4.3, used feature data from the three different data sets described in section 4.1. After ranking (4.4) the optimized algorithms were used together with the data sets to explore ensemble strategies (4.5), again using WEKA algorithms. From this process an ensemble of four tuned algorithms form the core of Rata.SSR. Finally an interface to WEKA from DataManager and a wrapper for Rata.SSR have been developed so that it can be used as described in Section 2 above.

4.1 Data sets for data mining

Feature data is required to run an initial trial on the algorithms; they are also necessary for the optimization and the ensemble building process. For the initial identification of possible algorithms a small simple data set of graph drawings was used. For the more critical algorithm optimization and ensemble, three diagram sets were collected from 20 participants. As at this stage we are concentrating on single stroke recognition, the participants were asked to draw each component in a single stroke. A summary of the data in each is shown in Table 1.

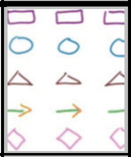
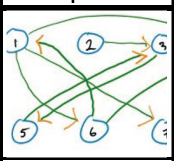
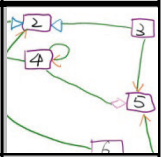
	ShapeData	GraphData	ClassData
Example			
Collection method	Isolated	In-situ	In-situ
Rectangle	80	-	170
Ellipse	80	146	-
Triangle	80	-	57
Line	80	172	215
Arrowhead	80	173	96
Diamond	80	-	60
Total	**480**	**491**	**598**

Table 1. *Data sets used for algorithm optimization*

The data sets are roughly differentiated by the number of shape types and the collection method. A variety of data sets should ensure the optimized algorithms can be applied to different diagram types.

These data sets were collected and labeled in DataManager [BPGW08]. From the labeled sketches DataManager can compute ink features, with the current version it is capable of generating 114 features [BCP10]. These features measure aspects of individual strokes such as curvature, size and density, and spatial and temporal relationships to other strokes in the diagram. They can be calculated in 0.087 seconds for each stroke. Data Manager's feature library is available online [Bla09].

4.2 Algorithm exploration

WEKA [HFH*09] was selected to supply the data mining algorithms: it is an open source tool which provides many data mining techniques. As WEKA is developed to support different data mining problems, each algorithm exposes settings that can be changed to tune it.

WEKA includes many algorithms and not all are suitable for ink feature data. We analyzed all algorithms provided by WEKA, with their default settings, on a simple data set. Among those algorithms, forty stood out as having higher accuracy than the others (greater than 90%). This number needed to be reduced for more detailed analysis. Although we could pick the top performing ones, the fact that a recognizer performed well on this simple data set does not guarantee its performance with other data sets. Furthermore, because we planned to explore combining algorithms with ensemble techniques, algorithms with strength in different aspects were desirable. In the end nine algorithms were selected which had good performance and differed in their underlying mechanisms. They were Bagging (BAG), Bayesian Network (BN), Ensembles of Nested Dichotomies (END), LogitBoost Alternating Decision Tree (LAD), LogitBoost (LB), Logistic Model Trees (LMT), Multilayer Perceptron (MLP), Random Forest (RF) and Sequential Minimal Optimization (SMO), each as implemented in WEKA [HFH*09].

4.3 Algorithm Optimization

Each algorithm in WEKA contains several settings which can be adjusted to alter the nature of that algorithm. The optimization conducted on each algorithm was a five step tuning process:

- Get the base performance from default settings
- Optimize each setting independently
- The optimized settings were combined to produce an optimized algorithm
- A series of evaluations between the optimized algorithm and the base algorithm, to select the more promising one
- An attribute selection process to further tune the algorithm – although this proved not to be useful

In the first step the data sets were used to train and test each algorithm with the default settings using 10 fold cross validation. The results for this are shown in Table 2 column 'Default'. Next we looked at the settings available for each algorithm in turn. WEKA makes different settings available depending on the algorithm: for example, users can specify the number of trees for Random Forest, or the number of iterations LogitBoost should run. Settings were considered individually; default values were used with the exception of the one being analyzed. To settings being studied, both true and false cases were applied if they are binary, and numerical settings were tested with a series of different values. The settings were applied to each data set independently. For each setting, the accuracies generated by changing its value were compared, and the value which returned the highest accuracy at the lowest cost was taken as the optimal value. In most cases the optimal setting was the same or had the same trend across data sets. If a setting had different effects on different data sets, the average of the data sets was taken. The optimized algorithm is that with all the individual settings at their optimal value.

We did not consider effect of different combinations of settings because of practical time and computational constraints. While each individual setting performs well with its optimal values, combining optimal values does not necessarily give the best results. Hence for each optimized algorithm we repeated the 10 fold cross validation (Table 2 column 'Opt').

While 10 fold cross validation can reduce the effect of over-fitting, it is not perfect for diagram recognition. Although over-fitting of training examples is prevented, because cross validation randomly selects training examples from the data set, each participant could participate in training data as well as testing data. Such situations are too optimistic for 'out of the box' recognizers where users are different from the ones who provided training data. Additionally, we are interested in the association between the numbers of training examples versus the resulting accuracy of a classifier. Based on these considerations two splitting experiments, random splitting and ordered splitting, were conducted. For each experiment, the data is split into training and testing. A 10% splitting indicates 10% of the data was selected for training and the remaining 90% for testing. Nine different splits were chosen, from 10% to 90% with 10% intervals.

Random splitting selects training examples randomly from the input data. To remove the noise the average of ten rounds is taken. Although a participant can still appear in both training and testing, this experiment shows the relationship between the number of training examples and the accuracy. Ordered splitting selects training examples from the start of the data set. For example, with a data set of 500 strokes, 10% splitting will take the first 50 strokes as training examples, while the rest become testing examples. Because our data sets were organized in the order of participants, and the numbers of strokes drawn by each of the 20 participants is similar, we can assume that each 10% in ordered splitting is equivalent to two participants, which ensures the training examples are from different participants who presented testing examples. The average results for the random and ordered splitting results are presented in Table 2. We noted that all algorithms had increased accuracy with more data, and all reached their maximum accuracy with less than 50% of the data.

The final step was an attribute selection experiment. As all 114 features were used, we were concerned about computation time for on-line, real-time recognition. Although it takes only 0.087 seconds to calculate on a desktop with an Intel® Core™2 Duo Processor E8400 and 4GB of RAM, we reasoned that ineffective features should be discarded; furthermore, the addition of new features in the future will most likely increase the calculation time. Wrapper subset evaluator, implemented in WEKA, is a suitable approach for attribute selection and it has been successfully applied to diagram recognition [PH08]. We compared the attribute selected algorithms with their default version, with 20% random splitting. The results are presented in Table 2 column 'Attribute Selection'. Two algorithms had marginally improved accuracy; however, the training time increased significantly for all algorithms. We hence decided not to apply attribute selection in this study.

4.4 Algorithm Ranking

In order to rank the algorithms we combined results from the different evaluations described above. In most cases we used the optimized algorithm; however the optimized SMO algorithm performed worse than its default version, so the default was used in the ranking. In addition we used the default version of LogitBoost as it performed almost equally with the optimized version in the 10 fold, and the default version performed better in the splitting tests.

As noted above, the splitting experiments suggested al-

	10 fold			Random splitting average						Ordered splitting average						Attribute Selection		Rank
				10%-90%			50%-90%			10%-90%			50%-90%					
	Default	Opt	R	Default	Opt	R	Default	Opt	R	Default	Opt	R	Default	Opt	R	Default	AttSel	
BN	97.5	98.6	1	96.6	97.9	1	97.1	98.4	1	94.7	96.3	1	95.2	97.5	1	95.9	94.0	1.0
RF	97.9	98.4	2	96.4	97.7	2	97.6	98.2	2	93.2	95.4	2	94.1	96.1	4	95.0	93.8	2.3
LAD	96.4	98.6	1	95.2	97.6	3	96.2	98.1	3	91.2	94.3	3	94.2	96.4	2	--	--	2.4
LB	98.4	98.7	2	96.9	94.7	4	98.0	98.2	4	93.1	92.3	4	96.3	96.9	3	95.8	92.8	3.4
LMT	98.2	98.4	2	95.7	96.1	6	97.2	97.3	7	91.4	91.5	5	95.0	95.2	6	95.3	93.0	4.9
MLP	98.4	98.4	2	95.9	96.0	7	97.6	97.6	5	90.5	90.7	7	95.3	95.3	5	93.8	94.2	5.3
END	97.3	97.9	4	95.2	96.2	5	96.8	97.4	6	90.1	91.0	6	93.6	94.6	7	93.8	92.2	5.4
SMO	98.2	98.0	3	95.4	94.4	8	97.1	96.0	8	90.1	88.9	8	93.8	93.1	8	93.3	93.7	6.8
BAG	96.1	96.4	5	94.6	95.3	9	95.9	96.3	9	89.8	89.8	9	92.1	92.5	9	93.8	92.6	8.0
Voting	99.4	99.6		97.9	98.4		99.0	99.3		96.7	97.4		97.8	98.3		--	--	--

Table 2. *Algorithm results and rankings (mean of tests over the three data sets described in section 4.1)*

gorithms have better performance with more training data. Reasoning that in real world usage more training examples will be available, rankings from splitting experiments with training data from 50% to 90% are included separately. However since they are using the same information as used by Random Splitting (RS) and Ordered Splitting (OS), we decided not to treat RS50 and OS50 as equal weight. The ranking was calculated by applying a nominal ranking score (as shown in Table 2) to each algorithm for each experiment, and applying the following formula:

$$Average\ Ranking = \frac{10fold + RS + OS + 0.5(RS50 + OS50)}{4}$$

Bayesian Network demonstrates the best overall performance, as shown in Table 2. LogitBoost shows an interesting case: although it generally performs well, it produces a comparatively poorer performance with the full splitting experiments. Investigation suggested that this is because it tends to over-fit with less training examples.

4.5 Ensemble

Two ensemble strategies were explored: voting and stacking. Voting combines classifiers by averaging the probability estimates of each class and making decisions on the highest possible classification. Stacking is more sophisticated: it applies two levels of classification by using a meta-classifier to classify the results returned by each contributing algorithm. As the meta-classifier should be relatively simple [WF05], we tested Zero R, Naïve Bayes, J48, and Simple Cart; among which Naïve Bayes has the best performance. Surprisingly, the results show that voting has higher accuracy (about 1% more). Because voting also requires less training time, we decided to focus on applying it as the ensemble strategy.

Different combinations of the base nine algorithms were explored to maximize the recognition rate and minimize the computation time – as nine algorithms have 511 possible combinations it was impractical to do an exhaustive search. After exploring a variety of strategies we applied the following steps using 10-fold cross validation:

1. Find the best **number** of algorithms by starting with all algorithms and progressively removing the lowest ranked algorithms. By comparing all the results, we found four contributing algorithms to be optimum.
2. Swap the worst performing of the four algorithms with a lower ranked algorithm and compare the results. If it improves the recognition rate retain it.
3. Repeat 2 for all algorithms.

We found that the voting by averaging the results with the combination of BN, LAD, MLP and LMT makes the best ensemble. A statistical Z-test was performed between this ensemble and the best performing algorithm, BN. On their average of all optimized results, it was confirmed that the difference was statistically significant, with a standard error of 0.003 and p-value of 0.011.

4.6 Data Manager Interface and Software Component

As our goal is to have a recognizer that can be used by other programs, we built a component which provides a simple interface connecting DataManager and WEKA, and also allows for the use of WEKA generated algorithms in C# programs. There are three motivating factors for this: WEKA is a complex tool that takes some time to learn, and by simplifying the interface non-experts will be able to use our configured WEKA algorithms; recognizers generated in WEKA are not immediately consumable by other programs; and finally WEKA is a Java tool which does not natively support C# sketch tools, so we required that it be encapsulated within a C# Java interface.

Although training only requires a list of features, because these features are collected and labeled through DataManager, an interface was built within DataManager for training the recognizers. This interface is shown in Figure 2. The user specifies the feature and output file locations and selects the algorithm from a dropdown list. Currently the list contains the ensemble described in section 4.5 and the optimized individual algorithms from section 4.3, hence users can ignore the configuration details. This interface makes it easy for users to collect and label data, and generate a recognizer all from the one tool. It also hides the complexity of WEKA.

For the Java – C# compatibility we applied IKVM.NET [Fri09] to translate the WEKA library from its native JAR file package into a C# compatible DLL.

The file produced from the DataManager training interface is a standard WEKA model file. It is a serialized java object which contains an exact copy of a trained recognizer including all the thresholds and settings; for example, for a tree algorithm it would have the feature and threshold value for each node in the tree. In the case of an ensemble, it contains the ensemble configuration and the configurations of all the contributing algorithms.

These recognizers can be used through Rata.SSR. It is the bridge between recognizers and programs which need recognition capability. It is implemented as a DLL for ease of transfer and use. The individual WEKA recognizers act like components, with Rata.SSR as the central recognition engine which can generate these components as well as use them. Individual recognizers or application programs do not have to provide code for functionality like feature calculation, as they are all implemented within Rata.SSR.

While Rata.SSR is implemented to ensure novice users can use it, it is also important to consider advanced users with data mining knowledge. The preset configurations for the algorithms are currently optimized for our experiment data presented in Table 1; although they promise a high recognition rate, for individual diagrams the configurations can be further tuned. The open source nature of WEKA allows it to be extended by data mining experts. New WEKA algorithms can then be used in Rata.SSR by updating the WEKA.dll. Additionally, experts can use the WEKA explorer interface to customize their recognizers. Because we unified the file structure of the exported classifiers of Rata.SSR and WEKA, the customized WEKA classifier can be loaded by Rata.SSR and used to recognize C# strokes. This configuration enables users to make further classification for individual classifiers, as well as allowing the use of all algorithms provided by WEKA.

5. Evaluation

To evaluate Rata.SSR we have compared it with four other recognizers using one of our own data sets and two data sets that have been used to evaluate other recognizers.

The other recognizers used in the evaluation are: $1 [WWL07], Rubine [Rub91] (as implemented in InkKit [PF07]), PaleoSketch [PH08], and CALI [FPJ02]. $1 and Rubine are trainable recognizers so any data set can be used. PaleoSketch and CALI, however, are hardcoded so to be fair we must consider their effectiveness on the classes that we can reasonably map between the data sets used in the evaluation and what they can recognize.

The data sets used in the evaluation are: our own FlowChart data set, a $1 data set, and a PaleoSketch data set.

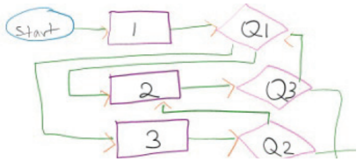
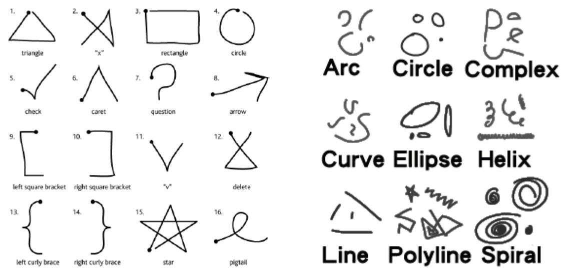


Figure 3: *Example diagram from the Flowchart data set with strokes color coded by class.*

The flowchart data set (Figure 3) was collected at the same time and from the same participants as the data set described in section 4.1. From the 20 participants there are 683 strokes which broken down by class are: rectangle 99, ellipse 42, line 242, arrow-head 239 and diamond 61. They were collected and labeled in DataManager [Bla09].



$1 data set from [WWL07] PaleoSketch data set

Figure 4: *Data sets from other research*

The $1 data set downloaded from [WWL09] has 16 different classes (Figure 4). It has been collected as isolated shapes, in a similar way to our ShapeData, from 11 participants. There are 16 shape classes with 330 examples per class. We converted the data to the DataManager format for the evaluation.

The PaleoSketch data set with 9 classes (Figure 4) was provided by [PH08]. It was also collected as isolated shapes and has data from 20 participants, with each drawing approximately 10 examples of each shape. The shape classes are: arc, circle, complex, curve, ellipse, helix, line, poly-line, and spiral. Complex and poly-line are catchall classes that do not represent any particular visual element - for example the adjacent star and random shape in Figure 3 are both categorized as 5 line polygons. There are some corruptions in this file that we have not managed to identify – we have 50% of the data for the evaluation. This provided ample examples for accurate results.

For the trainable recognizers, Rata.SSR, $1 and Rubine, we split each data set in half, trained on one half and tested on the other, and then reversed the sets for another round. There were small differences between the results for each round, which are averaged for each data set.

Evaluating the fixed recognizers, CALI and PaleoSketch, presented some problems - the raw tests when we did very simple mappings of obviously related shapes (such as ellipses and circles) resulted in very poor recognition rates 37.5% and 50.68% respectively. For these recognizers we also report success rates for those shapes that appear in both the data set and recognizer. The FlowChart data set can be fully evaluated as there are matching classes for each class in CALI and PaleoSketch - however they do classify some shapes as classes that do not exist in the flowchart, for example spirals. For the $1 data set, we removed Check, LeftCurly, Pigtail, RightCurly and Star for PaleoSketch, using only Caret, Circle, LeftCurly, Rectangle, RightCurly, v and Triangle for CALI. For the PaleoSketch data set, CALI can only handle Arc, Circle, Ellipse and Line; and for other recognizers, because of the variable nature of the PaleoSketch complex and poly-line classes we also report results with these two classes removed.

The evaluation is conducted with the evaluator implemented within DataManager [SPB09]. The results are shown in Table 3. In most cases Rata.SSR outperforms all of the other algorithms. The exceptions are the $1 algorithm on the $1 data set, and PaleoSketch on the PaleoSketch data. The $1 - $1 evaluation achieves 1.11% better result than Rata.SSR. All the trainable recognizers suffer from the catchall poly-line classes in the raw PaleoSketch data set – although Rata.SSR does achieve 89.9%. Tests without these classes show Rata.SSR performing at a very similar rate to PaleoSketch and outperforming the other trainable recognizers. Most of the Rata.SSR errors remaining are between curve and arc, removing one of these classes or combining them increases Rata.SSR to 98.4% and PaleoSketch to 99.2%.

	FlowChart	$1		PaleoSketch		Avg
		All	Part	All	Part	
Rata.SSR	98.7	97.1	--	89.9	94.9	96.9
$1	82.8	98.3	--	78.9	89.8	90.3
Rubine	93.3	95.7	--	41.2	46.1	78.4
CALI	85.2	37.5	85.1	42.2	95.0*	88.4
PaleoSketch	92.0	50.7	71.4	95.7	98.3	87.2

Table 3. *Evaluation results for recognizers against three data sets (All indicates all shape classes are used and Part means some are removed as described. Note CALI on PaleoSketch Part used even less class than the others)*

6. Discussion

In this project we use algorithms from an existing machine learning library, and with careful analysis, to configure an ensemble recognizer that is accurate and flexible. The first step was to explore a wide range of trainable algorithms using a large feature set and three sets of diagram data.

When exploring WEKA we found that there were many algorithms that gave good results for ink feature data.

However, we could not fully explore all of them; we chose nine that performed well and represented a variety of different AI techniques. While this was not an exhaustive exploration of available AI algorithms it is more comprehensive than has been reported elsewhere.

The nine selected algorithms were then individually tuned using three data sets. WEKA offers many tuning attributes for each algorithm. Computational constraints meant we could not try combinations of attributes; instead we found the optimal value for each attribute and then combined all optimum values. Retesting the optimized algorithms showed that most had improved about 1%, but two had worse performance. For those two we carried the original configuration through to the next phases.

Further optimization of the individual algorithms may be possible by tuning the combination of settings. We believe these values are related to the nature of the input data. More data sets and a lot of computational power are needed for further analysis. An area worth exploration is ways to dynamically find the best configuration for each data set.

We found the attribute selection algorithms in WEKA had very limited effect on the accuracy. It is likely this is because the algorithms are already applying attribute selection type behavior such as a tree structure or a voting mechanism. Comparatively SMO and Multilayer Perceptron had better performance with attribute selection – they applied neither tree structures nor voting mechanisms. The improvement, however, is marginal. Considering the lengthened training time, in most situations we believe the application of attribute selection is not required. However, while recognition time with the current 114 features is within real-time requirements, if more features are added, minimizing the number of features to be calculated would be sensible.

The ensemble proved effective with a 1% improvement in recognition rate compared to the best performing algorithm. To build the ensemble we tried voting and stacking, and found that the voting algorithms consistently perform better. It may be because the meta-algorithm used in stacking is not effective; however, four different meta-algorithms were used and none returned better result than voting. We speculate the reason is because the algorithms to be combined are all strong algorithms, and the errors they generate are not caused by their inability in a whole area, but due to noise in the data. Thus even when assigning the best performing algorithm for a section, misclassification would still occur. Furthermore, trying to find the best algorithm may result in over-fitting. In comparison, voting is more robust because it considers the probability returned by different algorithms which may filter the noise. As the algorithms we selected are all reasonably accurate, the probability based voting resulted in improved accuracy.

Interestingly, the combination of the best algorithms does not produce the best performing ensemble. This shows different algorithms have different strengths. The experiment data shows that Bagging, the worst performing algorithm we used, sometimes can correctly recognize shapes which the top performing Bayesian Network cannot.

To evaluate the optimized algorithms we explored 10 fold cross validation, random splitting and ordered splitting. We favor ordered splitting as we believe that this more closely emulates 'out of the box' recognizers – both random splitting and 10 fold cross validation may have participant participating in both training and testing data, and cause optimistic result due to drawing style learning.

Both splitting experiments can demonstrate the relationship between the number of training examples and the accuracy. Most algorithms have over 90% accuracy with 20% of our data as training examples. The accuracy of both experiments increased with more training examples, hitting maximum performance with 40-50% of our data sets.

From these results we suggest that as few as 10 examples per class are sufficient to train any of these algorithms to 90% accuracy and 50 examples will give close-to-optimal performance. Individualizing training data is likely to further improve the resulting accuracy.

While there are still areas where further exploration is possible we evaluated our best Ensemble against four other recognizers using three data sets. Of particular note is the performance of Rata.SSR against $1 on the $1 data set and Rata.SSR against PaleoSketch on the PaleoSketch data set. These data sets were collected to test the respective algorithms, with the capabilities of that algorithm in mind. Rata.SSR performed almost as well as these recognizers on their associated data sets and outperformed them on the other data sets.

Both $1 and PaleoSketch performed well on their own data sets. $1 is a trainable recognizer which has the potential to recognize different types of data. However its recognition approach ignores much of the rich spatial and temporal information that is available for digital ink. PaleoSketch has hard coded modules for each shape it can recognize. While it is possible to provide more shape types to cover all possible shapes [PH08], such implementation can also lead to a decrease in performance as the increase in classes provides a higher potential for mistakes to occur [Rub91]. Furthermore, although the heuristic based approach is easy to reason and program, we contend that when the underlying relationships are more complex they may not be human observable.

When measuring the evaluation results we were generous to the non-trainable recognizers PaleoSketch and CALI. The advantage of trainable recognizers is evident when one considers the widely varied performance of non trainable algorithms against the different data sets.

Rata.SSR's performance is due not only to the strength of its algorithms, but also to the wide range of features which capture the characteristics of the classes more comprehensively than in the other recognizers. It may be hard to add these relationships into hard coded or template matching approaches; however, they can be easily encapsulated into features and used with unmodified training algorithms. With our implementation, new features can be easily added into DataManager, which can be immediately used with the WEKA algorithms.

Rata.SSR has been developed and evaluated in the context of diagram recognition, as this is our principle area of interest. Rata.SSR as presented here is also likely to produce a very good gesture recognizer for functional gestures in the context of touch screen technology. The techniques may also be useful for the following stages of diagram recognition such as joining or splitting strokes and discerning relationships between basic shapes and components.

7. Conclusions

In this study we undertook an extensive evaluation of a wide range of data mining algorithms for recognizing digital ink. We examined both individual algorithms and ensemble strategies using a rich feature set. The Rata.SSR recognizer created as a result of this exploration is a software component which allows recognizers to be easily generated and used. Our comparative evaluation shows Rata.SSR to be flexible and accurate. In addition the results presented in Tables 2 and 3 provide some benchmarks against which other similar algorithms can be measured.

Acknowledgements

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

References

[AK93] APTE A., VO V., KIMURA T. D.: Recognizing multistroke geometric shapes: an experimental evaluation. *In Proc. UIST '93* (1993), pp. 121-128.

[BCP10] BLAGOJEVIC R., CHANG S. H.-H., PLIMMER B.: The Power of Automatic Feature Selection: Rubine on Steroids. . *In Proc. SBIM '10*, (2010), in press

[BSS04] BASILI R., SERAFINI A. ,STELLATO A.: Classification of musical genre: a machine learning approach. *In Proc. ISMIR '04* (2004)

[Bla09] BLAGOJEVIC R. DataManager. (2009), obtained from http://www.cs.auckland.ac.nz/research/hci/downloads/

[BPGW08] BLAGOJEVIC R., PLIMMER B., GRUNDY J. ,WANG Y.: A data collection tool for sketched diagrams. *5th Eurographics Conference on Sketch Based Interfaces and Modelling*, (2008)

[CSKK02] CALHOUN C., STAHOVICH T. F., KURTOGLU T. ,KARA L. B.: Recognizing Multi-Stroke Symbols. *AAAI Spring Symposium - Sketch Understanding*, (2002), pp. 15-23.

[CSJ00] CONNELL S. D., SINHA R. M. K. ,JAIN A. K.: Recognition of unconstrained on-line Devanagari characters. *in Proc. 15th ICPR*, (2000), pp. 368-371.

[FPJ02] FONSECA M. J., PIMENTEL C. ,JORGE J. A.: Cali: An online scribble recognizer for calligraphic interfaces. *AAAI Spring Symposium* (2002), pp. 51-58.

[Fri09] FRIJTERS J.: IKVM.NET (Version0.40.0.1). Available from http://www.ikvm.net/. (2009),

[Gro94] GROSS M. D.: Recognizing and interpreting diagrams in design. *Proceedings of the workshop on Advanced visual interfaces*, (1994), pp. 88 - 94.

[HFH*09] HALL M., FRANK E., HOLMES G., PFAHRINGER B., REUTEMANN P. ,WITTEN I. H.: The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, 1, 11 (2009)

[HEP*08] HAMMOND T., EOFF B., PAULSON B., WOLIN A., DAHMEN K., JOHNSTON J. ,RAJAN P.: Free-sketch recognition. putting the chi in sketching. *CHI '08 extended abstracts on Human factors in computing systems*, (2008), pp. 3027-3032.

[JGHD09] JOHNSON G., GROSS M. D., HONG J. ,DO E. Y.-L.: Computational Support for Sketching in Design: A Review. *Foundations and Trends® in Human–Computer Interaction*, 1, 2 (2009), pp. 1-93.

[KS04] KARA L. B. ,STAHOVICH T. F.: Hierarchical Parsing and Recognition of Hand-Sketched Diagrams. *UIST '04*, (2004), pp. 13-22.

[LZ04] LAVIOLA J. ,ZELEZNIK R.: MathPad 2: A System for the Creation and Exploration of Mathematical Sketches. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2004),* 3, 23 (2004), pp. 432-440.

[PPGI07] PATEL R., PLIMMER B., GRUNDY J. ,IHAKA R.: Ink features for diagram recognition. *Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling*, (2007), pp. 131-138.

[PH08] PAULSON B. ,HAMMOND T.: PaleoSketch: accurate primitive sketch recognition and beautification. *IUI '08*, (2008), pp. 1-10.

[PF07] PLIMMER B. ,FREEMAN I.: A toolkit approach to sketched diagram recognition. *Proceedings of HCI 2007* (2007), pp. 205-213.

[Rap10] RAPID-I RapidMiner. 2010, http://rapid-i.com/

[Rub91] RUBINE D.: Specifying gestures by example. *Proceedings of the 18th ACM SIGGRAPH Computer Graphics*, (1991), pp. 329-337.

[SPB09] SCHMIEDER P., PLIMMER B. ,BLAGOJEVIC R.: Automatic evaluation of sketch recognizers. *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling,* (2009), pp. 85-92.

[SSD01] SEZGIN T. M., STAHOVICH T. ,DAVIS R.: Sketch Based Interfaces: Early Processing for Sketch Understanding. *International Conference on Computer Graphics and Interactive Techniques*, (2001)

[Tay08] TAY K. S.: Improving digital ink interpretation through expected type prediction and dynamic dispatch. *Pattern Recognition (ICPR)*, (2008), pp. 1-4.

[VA05] VOGT T. ,ANDRE E.: Comparing feature sets for acted and spontaneous speech in view of automatic emotion recognition. *Multimedia and Expo (ICME)*, (2005), pp. 474-477.

[WNGV09] WILLEMS D., NIELS R., GERVEN M. V. ,VUURPIJL L.: Iconic and multi-stroke gesture recognition. *Pattern Recogn.*, 12, 42 (2009), pp. 3303-3312.

[WF05] WITTEN I. H. ,FRANK E. Data Mining: Practical Machine Learning Tools and Techniques. *In Morgan Kaufmann*, (2005)

[WWL09] WOBBROCK J. O., WILSON A. D. ,LI Y. $1 Unistroke Recognizer. (2009), http://depts.washington.edu/aimgroup/proj/dollar/

[WWL07] WOBBROCK J. O., WILSON A. D. ,LI Y.: Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. *UIST '07*, (2007), pp. 159 - 168

[YC03] YU B. ,CAI S.: A domain-independent system for sketch recognition. *GRAPHITE '03*, (2003), pp. 141 - 146