Building Digital Ink Recognizers using Data Mining: Distinguishing Between Text and Shapes in Hand Drawn Diagrams

Rachel Blagojevic[1, 1], Beryl Plimmer[1, 3], John Grundy[2, 2], Yong Wang[1, 4]

[1]University of Auckland, Private bag 92019, Auckland, New Zealand

[2]Swinburne University of Technology, PO Box 218, Hawthorn, Victoria, Australia 3122

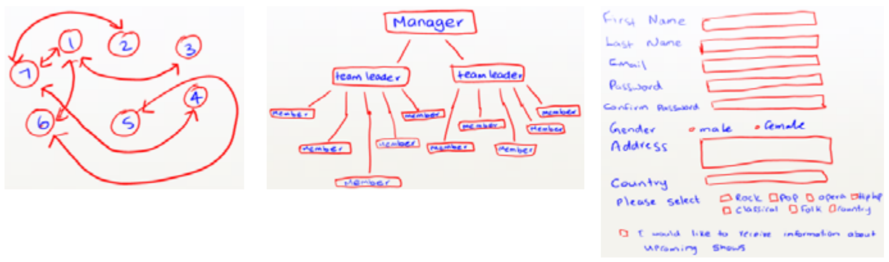
[1]rpat088@aucklanduni.ac.nz, [2]jgrundy@swin.edu.au, {[3]beryl@cs., [4]yongwang@}auckland.ac.nz,

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. The choice of features and algorithms is critical to the success of the recognition, yet heuristics currently form the basis of selection. We propose the use of data mining techniques to automate the process of building text-shape recognizers. This systematic approach identifies the algorithms best suited to the specific problem and generates the trained recognizer. We have generated dividers using data mining and training with diagrams from three domains. The evaluation of our new recognizer on realistic diagrams from two different domains, against two other recognizers shows it to be more successful at dividing shapes and text with 95.2% of strokes correctly classified compared with 86.9% and 83.3% for the two others.

Keywords: Sketch tools, recognition algorithms, sketch recognition, pen-based interfaces.

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 allow 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.



a) Directed graph b) Organization chart c) User interface

Fig. 1. Example sketched diagrams for training set

A number of sketch tools have been developed, however 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% to 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 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]. Most natural diagrams consist of both writing and drawing as shown in figure 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. Several recognizers [7-9], commonly referred to as dividers, have been proposed for this purpose, but recognition rates in realistic situations are still unacceptable. Limited investigation of machine learning for text-shape division has been found to be effective [8]. This work extends [8] by drawing on a larger set of ink features and using a range of data mining techniques systematically selected and tuned.

2 Background

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.

The majority of recognizers rely on information provided by various measurements of the digital ink strokes (digital ink is represented as a vector of x, y points, each point has a time and possible pressure attribute) [1, 2, 10], as well as specific algorithms to combine and select the appropriate features.

In the area of sketched diagram recognition many systems focus only on shapes [1-3]. There have been some attempts at incorporating text-shape division in domain specific recognizers [11, 12] and domain independent diagramming tools [10, 13]. 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-shape division [14-19]. However as the content of documents is mainly text these methods hold some bias which 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 [18, 20].

Three reports specifically on dividers are [7-9]. Bishop et al [8] use local stroke features and spatial and temporal context within an HMM to distinguish between text and shape strokes. They found that using local features and temporal context was 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. 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 and InkKit dividers, it was able to correctly classify more strokes overall for the test set.

A more recent development in this field is the use of a feature called entropy [7] to distinguish between shapes and text. First strokes are grouped into shapes and words/letters and then stroke points are re-sampled for smoothing. The angle between every point and its adjacent points in the stroke group is calculated. Each angle from the stroke group is matched to a dictionary containing a different alphabet symbol to represent a range of angles. This results in a text string representation of each stroke group. Using Shannon's entropy formula (as cited by Bhat et al [7]) they sum up the probabilities of each letter in the string to find the entropy of that group. This value of entropy is higher for text than shapes as text is more "information dense" than shapes. They report that 92.06% of data which it had training examples for were correctly classified. For data the divider had not been trained on it had an accuracy of 96.42%, however only 71.06% of data was able to be classified. We have re-implemented this algorithm for our evaluation. 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.

The choice of features and algorithms is critical to the success of the recognition, yet heuristics currently form the basis of selection. Given that features provide such value as input to recognition algorithms, a feature set should be chosen carefully using statistical or data mining techniques. While others have used some data mining techniques [8, 15] to the best of our knowledge no one has done a comprehensive analysis of algorithms. We present below a comprehensive comparative study of features and algorithms to select the most accurate model. In particular we are looking at the problem of distinguishing between text and shapes as a first step to recognizing sketched diagrams; a fundamental problem required to preserve a non-modal user interface similar to pen and paper.

3 Our Approach

In order to use data mining techniques to build classifiers we first compiled a comprehensive feature library which is used in conjunction with our training set of diagrams to generate a training dataset. We investigated a wide range of data mining algorithms before focusing on seven that were producing the most promising results. These seven algorithms and the training dataset were used to build new dividers.

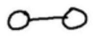
3.1 Features

Our previous feature set [9] of 46 features has been extended to a more comprehensive library of 115 stroke 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).

Many researchers have developed features that measure similar attributes. In order to give the reader some sense of the types of features we have categorized the feature library into ten categories, summarized in table 1.

This feature library is available with full implementation within DataManager [21] from www.cs.auckland.ac.nz/research/hci/downloads.

Table 1. Summary of stroke feature categories.

1. Curvature (e.g. the line above has a greater curvature than the line below).	**6. Pressure** (measure the pressure applied to the screen when drawing a stroke. Pressure is dependent on the capabilities of the hardware).
2. Density (e.g. the text has larger density of points than the shape).	**7. Size**
3. Direction (this is related to the slope of the stroke).	**8. Spatial context** (with sub categories: curvature, density, divider results, intersections, location and size).
4. Divider Results (these features provide the results of text/shape divider algorithms).	**9. Temporal context** (with sub categories: curvature, density, divider results, length, location/distance and time/speed).
5. Intersections (e.g. the diagram shows intersecting strokes). 	**10. Time / speed** (includes total, average, maximum and minimum times or speed).

3.2 Dataset

For the training set we have collected and labeled sketched diagrams from 20 participants using DataManager [21]. Each participant has drawn three diagrams; a directed graph, organization chart and a user interface e.g. figure 1. 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 have generated a dataset of feature vectors for each stroke using DataManager. DataManager's dataset generator function is able to take the diagrams collected and calculate feature vectors based on the implementation of our feature library.

3.3 Building Classifiers

Weka (developer version 3.7) [22], an open source data mining tool, has a large number of machine learning algorithms that can be used to perform our data analysis and build, tune and test classifier models for dividers. We found that 60 of the algorithms within Weka were possibly suited to the divider problem. We began our analysis with a preliminary investigation of all these algorithms. This involved building classifier models for each algorithm using the training data. Some clearly performed better than others while some needed tuning of their specific parameters to optimize their results. Upon discussion of the preliminary investigation we were able to narrow the search down to seven algorithms that are likely to gain the best classification accuracy for a divider[1].

The chosen classifiers are: Bagging [23](with an REP tree base learner), LADTree [24](alternating decision tree using the LogitBoost strategy), LMT [25](logistic model tree), LogitBoost [26](additive logistic regression with Decision Stump or REP tree base learner), MultilayerPerceptron [22] (neural network), RandomForest [27](forest of random trees) and SMO [28](support vector machine). 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 optimize their results.

For Bagging [23] we tuned the algorithm by varying the number of bagging iterations that the algorithm performs. This parameter is indicative of the number of trees that can be produced. The default value for this in Weka is 10 iterations. We ran an experiment using 10-fold cross validation for Bagging with REPTree (a fast decision tree learner) at 10, 100, 500, 1000 and 5000 iterations. Paired t-tests (α=0.05) showed no significant difference in the results at each level of iterations. The highest result is shown in table 2, this was produced at 500 and 1000 iterations.

To tune the LADTree [24] we varied the number of iterations of the algorithm to 10, 100, 500 and 1000 iterations. We were unable to increase the number of iterations to greater than 1000 due to time and memory constraints. This algorithm takes a long time to train therefore we chose to run the experiment with 5-fold cross validation as opposed to 10-fold. Paired t-tests (α=0.05) showed that the LADTree with 500 and 1000 iterations were significantly more accurate than the others. There was no significant difference between the LADTrees with 500 and 1000 iterations. The highest result shown in table 2 was produced at 1000 iterations.

The default parameters in Weka for LMT [25] are sensible for this problem and did not require tuning. The result of 10-fold cross validation using our training dataset on LMT with default parameters is shown in table 2.

To begin tuning LogitBoost [26] we ran a preliminary 10-fold cross validation experiment to see if there were any significant differences in using Decision Stump or

[1] Thanks to Eibe Frank for his advice on the selection of algorithms.

REPTree as a base classifier for LogitBoost. A paired t-test (α=0.05) showed no significant difference between the two at 120 iterations of the algorithm. Based on these results we decided to continue to investigate both trees as base classifiers.

To further tune LogitBoost we varied the number of iterations the algorithm performs and also the shrinkage parameter. Shrinkage is a parameter that can be tuned to avoid overfitting the LogitBoost model to the training dataset. When a classifier is overfitted it reduces the likelihood of the model retaining the same level of accuracy, achieved with training data, on a new test dataset. Small values for shrinkage reduce overfitting. We ran experiments using 10-fold cross validation for LogitBoost with the following options: base classifier as a Decision Stump or REPTree; number of iterations at 10, 100, 500, 1000 or 5000; shrinkage at 1.0 (Weka default value) or 0.1.

Using all combinations of the above options resulted in 20 models for LogitBoost, 10 for each base classifier. For LogitBoost, the model that had the highest level of accuracy was with a Decision Stump base classifier, 5000 iterations and a shrinkage value of 0.1, the result for this model is shown in table 2. Paired t-tests (α=0.05) showed that it was significantly better than all other Decision Stump models except two that were not significantly different; they had a shrinkage value of 1.0 and number of iterations set at 1000 and 5000. When compared with the REPTree models, it was only significantly better than the REPTree model with 10 iterations at a shrinkage value of 0.1, for all others there was no significant difference.

Table 2. Best results obtained from selected classifiers

Classifier	% Correctly classified (10-fold cross validation)
LADTree	**97.49*** (5-fold)
LogitBoost	**96.70***
RandomForest	96.45
SMO	96.41
Bagging	95.67
MultilayerPerceptron	95.02
LMT	94.85

The default parameters in Weka for MultilayerPerceptron [22] are sensible for this problem and did not require tuning. The result of 10-fold cross validation using the training dataset on MultilayerPerceptron with default parameters is shown in table 2.

To tune RandomForest [27] we varied the number of iterations of the algorithm to 10, 100, 500 and 1000 iterations. We were unable to increase the number of iterations to greater than 1000 due to memory constraints. Paired t-tests (α=0.05) showed no significant difference between any of the models. The highest result shown in table 2 was produced at 500 iterations.

SMO [28] is a more complicated classifier to tune. There are two parameters that can be tuned; the complexity value of SMO and the gamma value of the RBF kernel used by SMO. To find the best model we used the GridSearch function in Weka which allows you to optimize two parameters of an algorithm 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. We found the optimal value for complexity was 100, with a gamma value of 0.1. The results of 10-fold cross validation on SMO for our training data is shown in table 2.

The best results of 10-fold cross validation (except LADTree which was 5-fold) for each classifier on our training set is shown in table 2. Paired t-tests (α=0.05) 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.

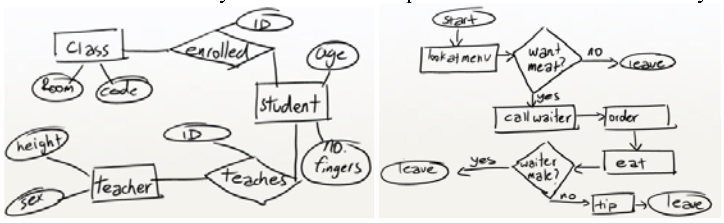
3.4 Implementation

In order to run a comparative evaluation of our two new models against other dividers we integrated our models into DataManager's Evaluator [6]. We also integrated our old divider [9] and implemented the Entropy divider [7].

The Entropy divider had to be trained as no thresholds were provided by [7]. We trained it on the same data as our new dividers using 10-fold cross validation with the decision stump algorithm from Weka [22] to find an optimal threshold. We chose the decision stump algorithm as this generates a decision tree with one node, essentially producing one decision based on the Entropy feature. The 10-fold cross validation reported that 85.76% of the training data was correctly classified; other algorithms such as OneR, a rule based method, and a J48 tree (C4.5 decision tree) showed similar results. Our divider developed from previous work [9] was not re-trained; it was implemented with the same thresholds as the original decision tree.

4 Evaluation

In order to test the accuracy of these dividers on data that they are not trained on we used a new set of diagrams from different domains to the training set. The test set was composed of ER and process diagrams (see figure 2) collected from 33 participants who drew one diagram from each domain. The participants were asked to construct the diagrams from text descriptions so that they are realistic in individual drawing. There are a total of 7062 strokes in our test set which is similar in size to our training set. There are 4817 text strokes and 2245 shape strokes. Table 3 shows the results for each divider on the test set of diagrams. LADTree is the most accurate of the four tested with 95.2% correctly classified closely followed by LogitBoost at 95.0%. The Entropy divider is the least accurate at a rate of 83.3%. It is clear that entropy has a large bias towards text as only 50.5% of the shapes in the test set are correctly



a)ER diagram b) Process diagram

Fig. 2. Example sketched diagrams for test set

Table 3. % Correctly classified for each divider.

Divider	% Correct	% Text	% Shapes
LADTree	95.2	98.3	88.5
LogitBoost	95.0	98.1	88.4
Old Divider	86.9	93.1	73.5
Entropy	83.3	98.7	50.5

classified. Our previous divider is slightly more accurate than Entropy; however its bias towards text is not as extreme. In fact the results show that all dividers classify text much more accurately than shapes.

5 Discussion

The high accuracy of the results we have obtained by using data mining techniques to build dividers demonstrates the effectiveness of this approach. We believe that other recognition problems would also benefit from a similar study of data mining techniques. However there is still room for improvement in these divider algorithms.

In terms of tuning, for all the algorithms where we varied the number of iterations we found that a high number of iterations usually resulted in significantly better results. We could tune these further by increasing the number of iterations for some algorithms however we are constrained by time and memory. Although, these constraints are for training, once the classifier is trained the memory requirements are minimal and actual classification time on instances is very fast in all cases.

We can also study the common types of failures that occur with recognition, in particular for shapes as they are the main source of misclassification. Data mining these misclassified strokes could identify features that may help correctly distinguish them. Studying error cases may also lead to the identification of new features that account for these misclassified shapes.

Feature selection strategies may also contribute to recognizer improvement. This involves using feature selection algorithms to isolate the features that perform well. When training an algorithm insignificant features can have a negative effect on the success of classification algorithms [22] therefore careful feature selection is a very important step to developing recognition techniques. We were surprised that 100+ features were employed by our top two dividers and speculate that some features are redundant or detrimental. Redundant features will only slow execution time whereas our concerns are with features that have a negative effect. Further exploration of feature selection strategies could identify features that should be excluded.

Combining different classifiers into a voting system is also worthy of investigation. Classifiers predictions can be weighted according to their performance and combined to produce one overall classification for an instance [22]. We are yet to investigate whether the different algorithms have a large number of common failures. If they all fail on the same cases then voting is not useful. For future work we plan to investigate the main cause of failures that occur for the original seven algorithms and identify what proportions are common between them.

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 (~ 2.3%), it may be worthwhile to data mine and train a divider for each diagram domain.

6 Conclusion

We have built seven new dividers using data mining techniques to distinguish between text and shapes in hand drawn diagrams. The two best dividers, LADTree and LogitBoost, are able to correctly classify 95.2% and 95.0% respectively of a test set that they have received no training for. A comparative evaluation of these dividers against two others shows that the new dividers clearly outperform the others. The success of our new dividers demonstrates the effectiveness of using data mining techniques for sketch recognition development.

7 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.

8 References

1. Rubine, D.H. *Specifying gestures by example*. in *Proceedings of Siggraph '91*. 1991: ACM.
2. Paulson, B. and T. Hammond. *PaleoSketch: Accurate Primitive Sketch Recognition and Beautification*. in *Intelligent User Interfaces (IUI '08)*. 2008. New York, USA: ACM Press.
3. Wobbrock, J.O., A.D. Wilson, and Y. Li, *Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes*, in *User interface software and technology*. 2007, ACM: Newport, Rhode Island, USA.
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*. 2004, University of Waikato.
5. Young, M., *InkKit: The Back End of the Generic Design Transformation Tool*, in *Computer Science*. 2005, University of Auckland: Auckland.
6. Schmieder, P., B. Plimmer, and R. Blagojevic. *Automatic Evaluation of Sketch Recognition*. in *Sketch Based Interfaces and Modelling*. 2009. New Orleans, USA.
7. Bhat, A. and T. Hammond. *Using Entropy to Distinguish Shape Versus Text in Hand-Drawn Diagrams*. in *International Joint Conference on Artificial Intelligence (IJCAI '09)*. 2009. Pasadena, California, USA.
8. Bishop, C.M., M. Svensen, and G.E. Hinton, *Distinguishing Text from Graphics in On-Line Handwritten Ink*, in *Proceedings of the Ninth International Workshop on Frontiers in Handwriting Recognition*. 2004, IEEE Computer Society.
9. Patel, R., B. Plimmer, et al. *Ink Features for Diagram Recognition*. in *4th Eurographics Workshop on Sketch-Based Interfaces and Modeling* 2007. Riverside, California: Eurographics.

10. Plimmer, B. and I. Freeman. *A Toolkit Approach to Sketched Diagram Recognition*. in *HCI*. 2007. Lancaster, UK: eWiC.
11. Lank, E., J.S. Thorley, and S.J.-S. Chen, *An interactive system for recognizing hand drawn UML diagrams*, in *Proceedings of the Centre for Advanced Studies on Collaborative research*. 2000, IBM Press: Mississauga, Ontario, Canada.
12. Hammond, T. and R. Davis. *Tahuti: A Geometrical Sketch Recognition System for UML Class Diagrams*. in *2002 AAAI Spring Symposium on Sketch Understanding*. 2002.
13. Zeleznik, R.C., A. Bragdon, et al., *Lineogrammer: creating diagrams by drawing*, in *Proceedings of User interface software and technology*. 2008, ACM: Monterey, CA, USA.
14. Shilman, M. and P. Viola. *Spatial recognition and grouping of text and graphics*. in *EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling*. 2004.
15. Shilman, M., Z. Wei;, et al. *Discerning structure from freeform handwritten notes*. in *Document Analysis and Recognition*. 2003.
16. Jain, A.K., A.M. Namboodiri, and J. Subrahmonia, *Structure in On-line Documents*, in *Proceedings of the Sixth International Conference on Document Analysis and Recognition*. 2001, IEEE Computer Society.
17. Ao, X., J. Li, et al., *Structuralizing digital ink for efficient selection*, in *Proceedings of the 11th international conference on Intelligent user interfaces*. 2006, ACM: Sydney, Australia.
18. Machii, K., H. Fukushima, and M. Nakagawa. *On-line text/drawings segmentation of handwritten patterns*. in *Document Analysis and Recognition*. 1993. Tsukuba Science City, Japan.
19. Microsoft Corporation, *Ink Analysis Overview*. 2008 cited 2008; Available from: http://msdn.microsoft.com/en-us/library/ms704040(VS.85).aspx.
20. Mochida, K. and M. Nakagawa. *Separating drawings, formula and text from free handwriting*. in *International Graphonomics Society (IGS2003)*. 2003. Scottsdale, Arizona.
21. Blagojevic, R., B. Plimmer, et al. *A Data Collection Tool for Sketched Diagrams*. in *Sketch Based Interfaces and Modeling*. 2008. Annecy, France: Eurographics.
22. Witten, I.H. and E. Frank, *Data Mining: Practical machine learning tools and techniques*. 2nd Edition ed. 2005, San Francisco: Morgan Kaufmann.
23. Breiman, L., *Bagging predictors.* Machine Learning, 1996. **24(2)**: p. 123-140.
24. Holmes, G., B. Pfahringer, et al., *Multiclass alternating decision trees.* ECML, 2001: p. 161-172.
25. Landwehr, N., M. Hall, and E. Frank, *Logistic Model Trees.* Machine Learning, 2005. **95(1-2)**: p. 161-205.
26. Friedman, J., T. Hastie, and R. Tibshirani, *Additive Logistic Regression: a Statistical View of Boosting*. 1998, Stanford University.
27. Breiman, L., *Random Forests.* Machine Learning, 2001. **45(1)**: p. 5-32.
28. Platt, J. *Machines using Sequential Minimal Optimization*. in *Advances in Kernel Methods - Support Vector Learning*. 1998.