Margin trees: a data structure for relationship classification in handwritten mathematical expressions

Scott MacLean George Labahn

Cheriton School of Computer Science, University of Waterloo

Nov. 5, 2013

Abstract

We develop a new data structure called a *margin tree* for multi-label classification, and apply it to the problem of classifying geometric relationships in handwritten mathematical expressions. A margin tree query returns an approximation of the distribution of class labels near the query point, permitting higher-level recognition systems to resolve or ignore any ambiguity as appropriate, based on context. The effectiveness of margin trees is demonstrated through comparison to several alternative classification methods.

1 Introduction

Many useful software systems exist which work with mathematical content, both for presentation (e.g., LaTeX, MathML), and computation (e.g., Maple, Mathematica). However, the text-based syntax used to input mathematical expressions to such systems is often error-prone, verbose, and otherwise difficult to use for non-experts. Recently, researchers have taken renewed interest in developing techniques for inputing mathematics via handwriting so that users of these systems might draw their input as they do with pen and paper, thereby avoiding the difficulties that arise with expression linearization and conversion to each system's particular syntax.

A technique commonly used for recognizing a handwritten math expression is to model the symbols as nodes linked by edges representing geometric relationships such as vertical or horizontal adjacency, superscript/subscript positining, and containment. This approach arises implicitly when two-dimensional grammars are used for recognition [2, 3, 8, 14], but it is also used independently of grammar-based techniques [4, 11, 13]. Correct recognition of these relationships is important because the spatial arrangement of symbols in a math expression contributes to the expression's semantic content (e.g., two symbols drawn as A^B indicate exponentiation, while two symbols drawn as AB indicate multiplication.)

Most existing techniques for classifying these geometric relationships make use of ad hoc scoring functions [2, 4, 8, 13]. Thanks the recent availability of several large data sets containing ground-truthed handwritten math expressions [9, 6], it is now possible to design a more data-driven approach. This work does just that by designing and training from data a discrimative Bayesian model for classifying the geometric relationships present in mathematical writing.

A common pattern in Bayesian probabilistic modelling is to devise two sets of variables: *observed* and *hidden*. In practical use, the observed variables typically correspond to aspects of some (potentially noisy, ambiguous, or otherwise unreliable) input which can be empirically measured, while the hidden variables correspond to an idealized system state driving the model.

In many problems, the goal is to sample the observed variables $O_1, \ldots, O_n$ at some moment in time, and to deduce the (most likely) values of the hidden state variables $S_1, \ldots, S_m$. Given appropriate prior distributions on the O_i and S_i, Bayes' theorem admits two possible strategies for accomplishing this goal: either the *generative* approach $P(O_1, \ldots, O_n \mid S_1, \ldots, S_m)$ or the *discriminative* approach $P(S_1, \ldots, S_m \mid O_1, \ldots, O_n)$.

In this work, we use the discriminative approach, leaving us with the problem of determining how the state variables are distributed given several observed values. We treat this as a multi-dimensional search problem and develop a data structure we call a "margin tree" which combines the notions of k-d trees and maximum-margin classifiers. A margin tree can be queried for the probability distribution of a variable (a discrete state variable in our case, either scalar or vector) in the neighbourhood of a point in $\mathbb{R}^k$ (the collection of observed variables), which we take to be an estimate of $P(S_1, \ldots, S_m \mid O_1, \ldots, O_n)$.

Other researchers have considered this combination of k-d trees and max-margin classifiers as well, but with quite different motivations and problem domains. Ram, Lee, and Gray [10] find the maximum-margin classifier and use it as a splitting plane at each recursive step of k-d tree construction. The resulting trees are used to solve approximate nearest neighbour queries with hard time limits by returning the nearest neighbour found so far when the limit expires. Tibshirani and Hastie [12] apply a similar idea to a multi-label classification scenario. At each recursive tree construction step, they partition the remaining training labels into two groups, maximizing the margin between the groups.

There are important differences between the problems these researchers have addressed and our own. In the work of Ram, Lee, and Gray, the data was unlabeled, so the maximum-margin classifier was free to split the data between any pair of points, so long as the distance between them was maximal. In the multi-label classification scenario of Tibshirani and Hastie, it is assumed that there is little enough overlap between classes that returning a single answer to a classification query will give sufficiently high accuracy.

In the relation classification problem, each data point is labeled with a relation name, and the data points corresponding to different relations overlap significantly. Our strategy is therefore to perform a range search around a query point and to return the categorial distribution of labels in that neighbourhood, rather than the single label of the nearest neighbour. In the context of math recognition, a higher-level recognition system (e.g., a grammar-based parser) can use contextual information to resolve ambiguities and select a particular label from the distribution.

The next section describes our Bayesian model in detail, and Section 3 defines the margin tree data structure used to compute model probabilities and develops the requisite training and query algorithms. The resulting relation classifier is evaluated against a number of alternative approaches in Section 4, and Section 5 concludes the paper with some observations and ideas for future research.

2 Formalizing the relationship classification problem

In online math recognition problems, the input is typically taken to be a set of ink strokes, each of which is an ordered sequence $(x_1, y_1), \ldots, (x_n, y_n)$ of points in the plane (the number of points n need not be the same for different strokes). Given two sets of input strokes, the relationship classification problem is to determine which, if any, of a set of spatial relationships applies between the sets. We will take this set to be $\{\rightarrow, \nearrow, \searrow, \downarrow, \odot\}$, where the arrows indicate the orientation of the relationship, and $\odot$ indicates containment.

For example, consider the expression shown in Figure 1. A correct relation classifier will identify the relation $\nearrow$ between the x and 2 in the x^2 subexpression, the relation $\odot$ between the $\surd$ and the

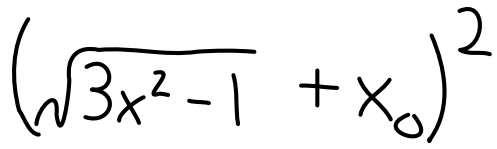

Figure 1: A math expression with many nested relationships.

$3x^2 - 1$, the relation $\rightarrow$ between the $\sqrt{3x^2 - 1}$ and $+$, and between the $+$ and x_0 (amongst other subexpression pairs), and so on.

2.1 A discriminative model for the relationship classification problem

The input to the classification problem consists of two sets of input strokes $U = \{u_1, \ldots, u_n\}$ and $V = \{v_1, \ldots, v_n\}$, as well as two subexpression labels s_U, s_V for U and V, respectively, taken from the alphabet $\Sigma = \Sigma_0 \cup \{\text{GEN}, \text{SYM}, \text{EXPR}\}$, where Σ_0 is the set of available math symbols (e.g. $a, 6, \gamma, +, \int$), EXPR indicates a multi-symbol math expression, SYM indicates a single symbol, and GEN is a generic label which may be applied to any subexpression.

We wish to determine which relation in $R = \{\rightarrow, \nearrow, \searrow, \downarrow, \odot\}$ applies to U and V. To do so, we define one (hidden) random state variable X over R, and six (observed) *feature* variables.

The feature variables are of two varieties: four geometric variables $F_\ell, F_r, F_t, F_b \in \mathbb{R}$ and two semantic variables S_U, S_V distributed over Σ.

Let $\ell(U), r(U), t(U), b(U)$ denote the left-, right-, top-, and bottom-most coordinates of the stokes in U, and similarly for V, and define

$$
\begin{aligned}
f_\ell &= \frac{\ell(V) - \ell(U)}{N}, \\
f_r &= \frac{r(V) - r(U)}{N}, \\
f_b &= \frac{b(V) - b(U)}{N}, \\
f_t &= \frac{t(V) - t(U)}{N},
\end{aligned}
$$

where $N = \max\{r(u) - \ell(u), b(u) - t(u)\}$ normalizes for the scale of the ink. Note that we take the y-axis to be oriented downward so that $b(*) \geq t(*)$.

In the discriminative Bayesian setting, we assume independence between the two varieties of variables and take

$$
\begin{aligned}
X^* = \text{argmax}_{r \in R} \, & P\left(X = r \mid F_\ell = f_\ell, F_r = f_r, F_t = f_t, F_b = f_b, S_U = s_U, S_V = s_V\right) \\
& \times P\left(S_U = s_U, S_V = s_V \mid F_\ell = f_\ell, F_r = f_r, F_t = f_t, F_b = f_b\right) \\
& \times P\left(F_\ell = f_\ell, F_r = f_r, F_t = f_t, F_b = f_b\right)
\end{aligned}
$$

to be the relation that applies between U and V.

As a short form, write this equation as

$$
X^* = \text{argmax}_{r \in R} \, P\left(X = r \mid F, S\right) P\left(S \mid F\right) P\left(F\right),
$$

where S is understood to be a pair from Σ and F a vector in $\mathbb{R}^4$.

2.2 Probability evaluation as a look-up problem

Through several data collection projects [6, 7], we have acquired annotated training data from which the values of X, F, and S may be extracted for each relation appearing in the ground-truth. The extraction process yields a database of tuples of the form $(r, s_U, s_V, f_\ell, f_r, f_t, f_b)$. Adopting a frequentist view, each of the three factors in the product may be treated as a look-up problem. For example, to compute $P(f_\ell, f_r, f_t, f_b)$, one looks up how many tuples in the database match in the appropriate dimensions and normalizes by the overall size of the database. Likewise, to compute $P(s_U, s_V \mid f_\ell, f_r, f_t, f_b)$, one counts the tuples matching on all six variables and divides by the number matching just the f_*.

The primary difficulty with this approach – and the motivation for the development of the margin tree structure – is that the four variables comprising F are continuous. They cannot be looked up directly, as the probability of finding a matching tuple is infinitesimal. Instead, a discretization strategy must be employed such that the f_* components of a query tuple are considered to match those in a database entry if they lie in some neighbourhood around f_*, giving an approximation of the true probability.

The role of margin trees is to organize the tuple database in such a way that it is efficient to query it for these discretized probabilities, but also that these queries return reasonably accurate approximations.

3 Margin trees

3.1 k-d trees

Margin trees are a machine-learning-inspired variant of the k-d tree data structure. k-d trees themselves are an extension of binary trees to k-dimensional data. Given a set of points from $\mathbb{R}^k$, Algorithm 1 constructs a k-d tree:

Algorithm 1 k-d tree construction

Require: input point set $P = \left\{p^{(1)}, \ldots, p^{(n)}\right\}$ as above, current tree depth D, and a set $S \subseteq \{1, \ldots, k\}$ of splitting dimensions
  if $n = 1$ **then**
    return leaf node with point $p^{(1)}$
  $d \leftarrow D \mod |S|$
  $s \leftarrow$ the dth-smallest element in S
  $M \leftarrow$ median of $\left\{p_s^{(i)} : i = 1, \ldots, n\right\}$
  create an internal node N with split value M
  recurse with $P = \{p \in P : p_s \leq M\}, D = D + 1$ to find left child of N
  recurse on $P = \{p \in P : p_s > M\}, D = D + 1$ to find right child of N
  return N

Algorithm 1 splits the input points at the median point along the first splitting dimensions, then splits each of those halves at their median points along the second splitting dimension, and so on, forming a tree structure of the split points. Algorithm 2 demonstrates how to find the tree node covering a query point q.

To see how this applies to our probability computations, recall that the data set contains 7-tuples of the form $(r, s_U, s_V, f_\ell, f_r, f_t, f_b)$. Consider constructing a k-d tree for these tuples, splitting along the four geometric variables f_*. Then, passing a query point $\hat{F} \in \mathbb{R}^4$ to Algorithm 2, an

Algorithm 2 k-d tree cell search

Require: k-d tree node N, query point $q = (q_1, \ldots, q_k)$, current tree depth D, and a set $S \subseteq \{1, \ldots, k\}$ of splitting dimensions
  if N is a leaf node **then**
    return N
  $d \leftarrow D \mod |S|$
  $s \leftarrow$ the dth-smallest element in S
  if $q_s \leq$ the split value for N **then**
    recurse on the left child of N with $D = D + 1$
  else
    recurse on the right child of N with $D = D + 1$

approximation to $P\left(\hat{F}\right)$ can be obtained at any tree node N visited by the algorithm by counting the number of children rooted at N and dividing by the number of nodes in the entire tree, with the discretization scale becoming finer and finer as the search procedure descends into the tree. Similarly, an approximation to $P\left(s_U, s_V \mid \hat{F}\right)$ can be obtained at a visited node N by counting the number of children rooted at N for which s_U, s_V appear in the appropriate tuple dimensions, then dividing by the number of nodes in the subtree rooted at N.

Every node in the k-d tree thus provides an approximation of the probability distributions in which we are interested for the hypercube neighbourhood that the node covers. This observation is one of the two key ideas on which the margin tree data structure is based.

3.2 Cluster preservation via the margin

The second key idea behind margin trees addresses a potential mismatch between the nature of the data being represented and the data partitioning strategy used in the k-d tree algorithms. By splitting the data at the median point, those algorithms achieve optimal runtime efficiency, and cycling through the tuple dimensions provides a simple default behaviour. However, this strategy is not always best for generating accurate function approximations.

Intuitively, we expect each of the tuples in our database to fall into one of several clusters, each associated with a particular combination of relation and semantic variables. I.e., we expect most of the tuples arising from a $\rightarrow$ relation between symbols 2 and a to be near one another, and rather different from those arising from a $\downarrow$ relation between symbols x and —, which form their own cluster. Some of the clusters might overlap significantly; for example, the expressions b^x and px have similar bounding box profiles. But nevertheless many clusters will be quite distinct.

This distinction is ignored by the standard k-d tree algorithms above, which can lead to clusters being split across different tree nodes. For example, the leftmost graph of Figure 2 shows two clusters, coloured orange and blue. The standard median-based k-d tree construction algorithm (Alg. 1) would split the blue cluster across both halves of the data, as shown in the central graph of the Figure. A more appealing subdivision of the data which preserves the structure of its clusters is shown in the right-most graph.

Preserving the cluster-based structure of the data segregates distinct clusters into different tree nodes. Since the tree node split points determine the neighbourhoods in which we compute different approximations of the probability distributions of interest, this segregation effectively divides up the search space into neighbourhoods that more accurately reflect the natural groupings of data points, rather than the arbitrary neighbourhoods obtained by splitting the data in half.

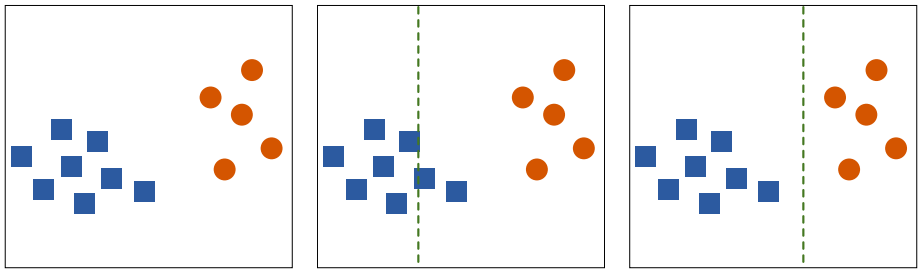

Figure 2: Data clusters and splitting possibilities

To accomplish this cluster separation, we rely on a basic idea of machine learning: margin maximization. Given two sets of data points X and Y, and a curve dividing all the points in X from those in Y, the margin is taken to be the minimal distance from any data point to the curve. Choosing the curve with the largest margin yields the "best" classifier. (A bit more formally, the classifier has the smallest generalization error: under certain theoretical conditions, it makes the fewest errors when classifying previously-unseen data.)

Therefore, rather than splitting the data at the median point along an alternating dimension, we shall split the data in such a way that the margin is maximized at each step in the algorithm. This does not strictly constitute a maximum-margin classifier since we are not explicitly segregating all of the distinctly-labeled data into distinctly-labeled clusters. However, we are subdividing the data in such a way that a query point (f_ℓ, f_r, f_t, f_b) is likely to be mapped to a tree node containing data from exactly the clusters (i.e., the relation and symbol labels r, s_U, s_V) to which the query point is most likely to below, and not data from some other clusters which were included in the tree node by chance.

3.3 Margin tree algorithms

Having described the intuition behind margin trees: hierarchical max-margin-based partitioning of approximation neighbourhoods, we can now give explicit algorithms for constructing (Algorithm 3) and querying (Algorithm 4) a margin tree.

Algorithm 3 Margin tree construction

Require: An input point set $P = \{p^{(1)}, \ldots, p^{(n)}\}$, and a set $S \subseteq \{1, \ldots, k\}$ of splitting dimensions
  if $n = 1$ **then**
    return leaf node with point $p^{(1)}$
  Choose a splitting dimension $d \in S$ and a split point s such that $p_d^{d,(s+1)} - p_d^{d,s}$ is maximal, where $p^{i,j}$ is the jth point in P when ordered along dimension i
  $M \leftarrow \left(p_d^{d,s} + p_d^{d,(s+1)}\right)/2$
  Create an internal node N with split dimension d and point $p^{d,s}$
  Recurse with $P = \{p \in P : p_d < M\}$ to find left child of N
  Recurse on $P = \{p \in P : p_d > M\}$ to find right child of N
  return N

Algorithm 4 demonstrates computation of a conditional query matching the form required for our distributions. In it, the notation size(N) denotes the number of margin tree nodes rooted at N (including N itself).

Algorithm 4 Margin tree query

Require: A k-d tree node N, query point $q = (q_1, \ldots, q_k)$, a set $S \subseteq \{1, \ldots, k\}$ of splitting dimensions, and a set $Q \subseteq \{1, \ldots, k\} \setminus S$ of query dimensions
  if size$(N) < C_1$ **then**
    return "insufficient data"
  if size$(N) < C_2$ **then**
    Let m be the number of nodes rooted at N with data points matching q on all query dimensions
    return $m/$size(N)
  Let d be the splitting dimension for N
  if $q_d \leq$ the split value for N **then**
    Recurse on the left child of N
  else
    Recurse on the right child of N

The constants C_1 and C_2 in Algorithm 4 control the granularity of probability approximations. If a tree node was built from fewer to C_1 training samples, it is deemed too sparsely populated and not used. Cells constructed from between C_1 and C_2 samples are deemed to be sufficiently fine-grained and well-populated, and are used for probability estimation. Cells containing more than C_2 samples are considered too coarse-grained and are recursively searched for more finely-grained approximation neighbourhoods.

3.4 The problem of outliers

As constructed by Algorithm 3, margin trees do not handle outlier data points appropriately. Suppose the set of n input points includes a point p which lies far from the rest of the points along dimension d. Then the large gap between p and the other points will cause the algorithm to partition p into its own tree node splitting the data along dimension d, and will recurse on the remaining $n - 1$ points. This construction causes a significant slowdown in the query algorithm, which must search through chains of linearly-structured nodes arising from outliers before reaching nodes that correspond to true data clusters.

Outliers may be filtered out from the input data, but such filtering becomes unreliable in high-dimensional spaces, and we did not have success experimentally with this approach.

Instead, we adjust the split selection criteria in Algorithm 3 so that it balances between choosing the max-margin and median points. In particular, we replace the maximization of $p_d^{s,(s+1} - p_d^{d,s}$ with the maximization of a weight function $w(s/n, p^{d,s}, p^{d,s+1}, d)$. Based on experimentation, we chose

$$w(x, p^{(1)}, p^{(2)}, d) = \omega\left(p^{(1)}, p^{(2)}\right)\left(1 - 4x^2\right)\left(p_d^{(2)} - p_d^{(1)}\right).$$

The third factor is the same as in Algorithm 3. The second weights each potential split point based on its position in the sorted point lits; the median point is assigned the highest weight, and weights tail off to zero at the lists extremities. The first factor $\omega(p^{(1)}, p^{2)})$ is another weight function measuring the difference in the non-query variables (i.e., for our application, the clustering variables r, s_U, s_V):

$$\omega(p^{(1)}, p^{(2)}) = \frac{1 + \sum_{i \notin S} 1 - \delta\left(p_i^{(1)}, p_i^{(2)}\right)}{1 + k - |S|}.$$

ω assigns higher weights to pairs of points belonging to "more different" clusters, and lower weights to points belonging to the same cluster.

The weight function w thus prefers to split the data at the maximum margin position, by weighting in favour of large gaps from different data clusters. But it balances this preference against a second preference to split the data at the median point for runtime efficiency.

4 Experimental results

To evaluate the effectiveness of margin trees in the context of relationship classification, we compared the algorithms from the previous section against several alternatives:

1. the existing fuzzy membership functions in the MathBrush math recognizer, as described in previous work [8],
2. a grid-based discretization strategy, described below, and
3. a standard k-d tree implementation.

4.1 Grid-based discretization strategy

We included grid-based discretization in our evaluation because it is a simple and commonly-used strategy for approximating probability distributions. The training process we followed is as follows:

1. Fix a number N of grid cells per dimension. (We used $N = 6$ based on experiments, giving a total of $6^4 = 1296$ grid cells for our four-dimensional data points (f_ℓ, f_r, f_t, f_b).)
2. Divide the training set into outliers and non-outliers. (We used Tukey's outlier test [5].)
3. Divide the inner $N - 2$ grid cells in each dimension equally-spaced between the minimum and maximum values appearing in non-outlier data points in each dimension. Reserve the two outer grid cells in each dimension for outliers.
4. Assign each input point to the grid cell containing it.

Then query points q are simply mapped to the grid cell containing them, and the probability distributions of interest are estimated similarly to in the k-d tree case by counting up the number of training samples in the cell that match on some attributes of interest, and dividing by the total number of relevant training samples.

4.2 Methodology and results

To obtain training data for our experiment, we divided the data from our 2009 collection study [6] roughly in half randomly, and designated one half for training. Two sets of testing data were used: the second half from the random split of the 2009 data, as well as the data collected in our 2011 study [7] (Tablet PC data only). The experiment was replicated five times with a different random partitioning of the 2009 data into training and testing halves for each replication.

Table 1 reports the percentage of relations classified correctly by the four methods on the 2009 data set, averaged over the five replications. Table 2 reports similar results for the 2011 data set.

Method	↗	→	↘	↓	⊙	Total
Fuzzy	84.02	91.72	69.31	95.95	98.79	90.50
Grid	90.86	97.89	63.52	98.12	92.27	95.03
k-d tree	76.95	99.15	47.80	98.31	97.58	93.96
Margin tree	87.64	98.96	60.89	98.61	97.20	95.58

Table 1: Experimental results on 2009 data set.

Method	↗	→	↘	↓	⊙	Total
Fuzzy	92.32	81.25	48.87	94.63	89.78	82.62
Grid	94.09	95.51	55.98	96.14	87.88	93.02
k-d tree	84.55	98.43	40.78	97.66	96.93	93.01
Margin tree	94.97	97.54	50.14	98.93	91.09	94.53

Table 2: Experimental results on 2011 data set.

4.3 Analysis

Overall, the three probabilistic methods performed significantly better than the rule-based fuzzy implementation. This improvement was, to a large degree, due to incorporating information about the prior probabilities of each relation. Indeed the most common relations (→ and ↓) are recognized much more accurately by the probabilistic methods, while the accuracy gap is much smaller, or even reversed, for the other three less-common relations.

The k-d tree method in particular seems to be overly biased toward → and ↓, obtaining very high classification accuracy for those two relations, but much weaker results on ↗ and ↘, which are typically most confused with →. This difference in accuracy between the relations is unsurprising given the observation in Section 3.2 that the data partitioning strategy used to construct the k-d tree will split data clusters across tree nodes. As such, it is expected that the most common relations will be represented in most tree nodes.

The grid-based method was surprisingly effective for such a straightforward method. It is influenced by the prior distribution of relations, as it explicitly counts training examples, but the partitioning of data points into grid cells is determined a priori and is not influenced by the data itself. (Unlike the tree-based methods where the hierarchical subdivision of the data is itself determined by the data.) This isolation of the partitioning strategy from the prior distribution helped to keep the classification accuracy of ↗ and ↘ higher than for the tree-based methods.

The margin tree combines both of these properties. It is influenced by the data's prior distribution and thus attains high classification accuracy for → and ↓, but its margin-based data partitioning strategy yields significantly higher classification rates for ↗ and ↘ than for the standard k-d tree. Additionally, the margin tree structure appears to generalize more readily to samples written by new writers: its accuracy reduced the least out of all methods when classifying 2011 data when trained on 2009 data.

The accuracy rate for the ↘ relation is poor across all data sets and classification methods. This is likely due to the large amount of feature overlap with → and the fact that → is much more common; thus the optimal strategy in the case of ambiguous data is to classify an input as →, causing proportionally more errors on ↘. With a richer data set, the geometry of various symbol combinations could be modeled in more detail, leading to more accurate recognition.

5 Conclusions

Margin trees are multi-dimensional binary trees that use a margin-maximization strategy for choosing data partitionining points. They are specifically designed to compute probabilities of the form $P(x_1, \ldots, x_n \mid y_1, \ldots, y_m)$, where the x_i are categorical variables and the y_i are continuous measurements.

Margin trees attain a significant increase in relation classification accuracy over the existing fuzzy relation classifier used in MathBrush, and are more accurate overall than the other probabilistic methods we evaluated. They effectively balance the prior distribution of the training data with its particular cluster-oriented structure.

In the future, we may extend the existing margin tree algorithms to use the kernel trick so that data may be partitioned by non-linear boundaries [1]. Additionally, it would be convenient to develop an on-line update algorithm which, given one or more training sample, efficiently adjusted an existing margin tree data structure to accomodate the new samples. By doing so, the probability distribution approximations could be continuously updated with new training samples as the user writes. Finally, a more formal theoretical analysis considering the training and generalization error of margin trees, as well as how to optimally select the parameters C_1 and C_2 in Algorithm 4 may yield insights leading to further improved accuracy.

References

[1] M. Aizerman, E. Braverman, and L. Rozonoer, *Theoretical foundations of the potential function method in pattern recognition learning*, Aotomation and Remote Control **25** (1964), 821–837.

[2] Francisco Álvaro, Joan-Andreu Sánchez, and José-Miguel Benedí, *Recognition of printed mathematical expressions using two-dimensional stochastic context-free grammars*, Proc. of the Int'l. Conf. on Document Analysis and Recognition, 2011, pp. 1225–1229.

[3] Ahmad-Montaser Awal, Harold Mouchere, and Christian Viard-Gaudin, *Towards handwritten mathematical expression recognition*, Document Analysis and Recognition, International Conference on (Los Alamitos, CA, USA), IEEE Computer Society, 2009, pp. 1046–1050.

[4] Utpal Garain and B. Chaudhuri, *A corpus for ocr research on mathematical expressions*, Int. J. Doc. Anal. Recognit. **7** (2005), no. 4, 241–259.

[5] David Hoaglin, Frederick Mostaller, and John Tukey (editors), *Understanding robust and exploratory data analysis*, John Wiley & Sons, 1983.

[6] S. MacLean, G. Labahn, E. Lank, M. Marzouk, and D. Tausky, *Grammar-based techniques for creating ground-truthed sketch corpora*, Int'l. J. Document Analysis and Recognition **14** (2011), 65–74.

[7] S. MacLean, D. Tausky, G. Labahn, E. Lank, and M. Marzouk, *Is the ipad useful for sketch input?: a comparison with the tablet pc*, Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling, SBIM '11, 2011, pp. 7–14.

[8] Scott MacLean and George Labahn, *A new approach for recognizing handwritten mathematics using relational grammars and fuzzy sets*, International Journal on Document Analysis and Recognition (IJDAR) **16** (2013), no. 2, 139–163.

[9] Harold Mouchère, Christian Viard-Gaudin, Dae Hwan Kim, Jin Hyung Kim, and Utpal Garain, *Icfhr 2012 - competition on recognition of on-line mathematical expressions*, Proc., Int'l Conf. Frontiers of Handwriting Recognition, 2012, pp. 1–6.

[10] Parikshit Ram, Dongryeol Lee, and Alexander G. Gray, *Nearest-neighbor search on a time budget via max-margin trees.*, Proc., SIAM Int'l Conf. on Data Mining, 2012, pp. 1011–1022.

[11] Yu Shi, HaiYang Li, and F.K. Soong, *A unified framework for symbol segmentation and recognition of handwritten mathematical expressions*, Document Analysis and Recognition, Ninth International Conference on, 2007, pp. 854–858.

[12] Robert Tibshirani and Trevor Hastie, *Margin trees for high-dimensional classification*, J. Machine Learning Research **8** (2007), 637–652.

[13] H.-J. Winkler, H. Fahrner, and M. Lang, *A soft-decision approach for structural analysis of handwritten mathematical expressions*, Proc. International Conference on Acoustics, Speech and Signal Processing, 1995, pp. 2459–2462.

[14] R. Yamamoto, S. Sako, T. Nishimoto, and S. Sagayama, *On-line recognition of handwritten mathematical expression based on stroke-based stochastic context-free grammar*, The Tenth International Workshop on Frontiers in Handwriting Recognition, 2006.