A Practical Approach for Writer-Dependent Symbol Recognition Using a Writer-Independent Symbol Recognizer

Joseph J. LaViola, Jr., *Member*, *IEEE*, and Robert C. Zeleznik

Abstract—We present a practical technique for using a writer-independent recognition engine to improve the accuracy and speed while reducing the training requirements of a writer-dependent symbol recognizer. Our writer-dependent recognizer uses a set of binary classifiers based on the AdaBoost learning algorithm, one for each possible pairwise symbol comparison. Each classifier consists of a set of weak learners, one of which is based on a writer-independent handwriting recognizer. During online recognition, we also use the n-best list of the writer-independent recognizer to prune the set of possible symbols and, thus, reduce the number of required binary classifications. In this paper, we describe the geometric and statistical features used in our recognizer and our all-pairs classification algorithm. We also present the results of experiments that quantify the effect incorporating a writer-independent recognition engine into a writer-dependent recognizer has on accuracy, speed, and user training time.

Index Terms—Handwriting recognition, AdaBoost, writer dependence, writer independence, pairwise classification, real-time systems.

1 Introduction

With the increasing popularity of pen-based computers, accurate and robust handwriting recognition is becoming a critical part of many pen-based applications. The choice of recognition engine—writer-independent or writer-dependent—has a significant impact beyond just recognition quality. Writer-independent systems provide the benefit that end users can simply step up to the application and start writing without any awareness of or special interaction with the underlying recognizer. Alternatively, writer-dependent systems maximize recognition accuracy by requiring that end users first provide some form of structured input in order to tailor the recognition engine to the user. The choice of recognition engine also has a significant impact on application design, since writer-independent systems are typically harder to customize to a specialized symbol set (for example, mathematical symbols or users who have alternate techniques for entering the same symbol). For example, the Microsoft Handwriting Recognizer [18] is designed to allow an application designer to choose a specific symbol set from a predefined list; however, the designer cannot make any modifications to a chosen set (for example, restricting the letter "Z" to Z or $\mathbb{Z}$), nor can additional symbols such as $\int$ and $\sum$ be added to the set.

Given the functional advantages of writer-dependent recognition, we were interested in the hypothesis that the primary deficiency of writer-dependent recognition, extensive training time on the part of the user, could largely be addressed by bootstrapping the training process with a writer-independent recognizer. In essence, we want to leverage the extensive a priori training typically done during the development of a writer-independent recognizer. Thus, our objective in this work is to quantify the effect incorporating a writer-independent recognizer would have on writer-dependent recognition accuracy as a function of the number of training samples used per symbol.

Consequently, we developed a representative symbol recognizer that, for simplicity of implementation, uses a set of binary classifiers, based on AdaBoost [34], as part of an all-pairs recognition algorithm. We use the Microsoft handwriting recognizer as weak learners in each of the pairwise classifiers. This naive all-pairs strategy was chosen specifically for its simplicity and accuracy. Runtime efficiency was not a primary focus of our work, although we did exploit an important speedup opportunity arising from the observation that the Microsoft handwriting recognizer has the correct symbol in its n-best list over 99 percent of the time, despite its overall accuracy of just over 91 percent (see Section 4). By using the n-best list during a preprocessing step to reduce the number of possible symbol candidates, we were able to prune the number of classifiers needed for each input symbol. Interestingly, this pruning step not only improves runtime speed but also improves recognition accuracy.

In the next section, we discuss related work on symbol recognition. Section 3 describes our recognition algorithm, including the features used in our weak learners and how we incorporate the Microsoft handwriting recognizer into our writer-dependent recognition engine. Section 4 presents the results of our recognition experiments and discusses their implications. Finally, Section 5 presents conclusions.

2 Related Work

There has been a significant amount of work in developing both writer-dependent and writer-independent symbol

- *J.J. LaViola Jr. is with the School of Electrical Engineering and Computer Science, University of Central Florida, Engineering 3—Harris Center, Orlando, FL 32816-2362. E-mail: jjl@cs.ucf.edu.*
- *R.C. Zeleznik is with the Department of Computer Science, Brown University, Box 1910, Providence, RI 02912. E-mail: bcz@cs.brown.edu.*

Manuscript received 21 Oct. 2005; revised 7 June 2006; accepted 4 Jan. 2007; published online 8 Feb. 2007.
Recommended for acceptance by D. Lopresti.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number TPAMI-0562-1005.
Digital Object Identifier no. 10.1109/TPAMI.2007.1109.

recognition systems, and there has been a variety of different algorithmic approaches for doing so [6], [29], [39]. For example, one of the first approaches to symbol recognition was to break symbols up into zones based on the symbol's bounding box. The sequence of zones traversed by the stylus was used to identify the symbol [9], [12]. Similarly, pen motion based on direction sequences has been used with lookup tables to recognize symbols [15], [30]. Smithies et al. [37] and Rubine [32] both use a variety of statistical and geometric features (angle and quadrant histograms, aspect ratio, stroke length, and others) as input to a K-Means classifier and a simple linear classifier, respectively. Classification algorithms include template matching [8], [27], [28], decision trees [3], [19], neural networks [11], [24], hidden Markov models (HMMs) [20], [21], [41], support vector machines [2], Gaussian classifiers [26], and principal component analysis [10]. In addition to these approaches, AdaBoost has been utilized for symbol recognition in conjunction with neural networks [35] and Viola-Jones filters [36].

Although there has been a significant amount of work done in symbol recognition, to the best of our knowledge, no one has attempted to incorporate a robust writer-independent recognition engine such as the Microsoft handwriting recognizer into a writer-dependent scheme for reducing the training set and improving accuracy and speed. However, there has been work on adapting writer-independent symbol recognizers to a particular user's writing style. Subrahmonia et al. [38] use an HMM-based writer-dependent system that adapts a particular set of writer-independent character models to a writer. Connell and Jain [7] take a slightly different approach by using writer-independent writing style models (lexemes) to identify styles present in a particular writer's training data. These models are updated using the writer's data and writer-independent models replace lexemes with inadequate training samples. Other writer adaptation approaches are discussed in [5].

There has also been work on managing the variability in writing style between different writers and within the same writer at different times through allograph modeling. Biem [4] uses the Minimum Classification Error criterion to reduce error rates using multiple allographs per character. Other approaches for dealing with different writing and printed character styles can be found in [25], [31], [33]. The limitation with these approaches is that the symbol recognition engine designer may not have these independent writing style models, the writer-independent training data, or access to the inner workings of an independent recognizer. With our approach, the output of the writer-independent recognition engine (the n-best list) is the only information required. Thus, the writer-independent recognizer can be treated as a black box, making it simpler to use and integrate into a writer-dependent recognition engine.

3 Pairwise AdaBoost Recognition and the Microsoft Handwriting Recognizer

Of the many possible learning algorithms and classifiers used in symbol recognition, we chose to use AdaBoost in conjunction with simple weak learners based on different statistical and geometric features (see Section 3.1). AdaBoost, developed by Freund and Schapire [13] helps to improve recognition accuracy by combining simple learning algorithms, is relatively simple to implement, and is easily extensible. Another important design decision we made was whether to use a multiclass extension of AdaBoost or to use a set of pairwise classifiers (for example, multiple binary classifiers) and then combine their results to generate a decision. Based on the work of Hastie and Tibshirani [17] and Friedman [14], we chose a pairwise classification approach where each pair is compared to each other and use a "max-wins" rule to make a classification.

3.1 Statistical and Geometric Features

The main input to our symbol recognizer is not a symbol's digital ink strokes but rather features calculated from them. A stroke is defined as a sequence of points $s = p_1 p_2 \dots p_n$ in the xy-plane, where $p_i = (x_i, y_i)$, $1 \leq i \leq n$, p_1 is the pen-down point, p_n is the pen-up point, and n is the number of points in the stroke. The features we use describe symbols numerically and are designed to create boundaries between them, so one symbol can be discriminated from another in feature space. Of the 14 different types of features we use in our recognizer, the first nine are taken from various papers [23], [32], [37], with the last five developed from our own observations.

3.1.1 Symbol Strokes

Each symbol contains a number of strokes. If we assume that users write consistently (that is, they always write a given symbol with the same number of strokes), then this is one of the few features we can count on to disambiguate certain symbols from others. Therefore, we can break up the number of possible symbols into groups before doing any training. For example, if a user writes an "x" with two strokes, we can initially disregard any symbols that have only one stroke. This approach lets us break up our symbol recognizer into a set of recognizers on the basis of how many strokes the symbol contains.

3.1.2 Cusp Features

Cusps are defined as points at which two branches of a curve meet such that the tangents of each branch are equal [40]. In other words, cusps represent locations of high curvature or discontinuity in a stroke. Cusps are good discriminators between smooth and jagged symbols: For example, the letter "m" can have two cusps (depending on how it is written), whereas the letter "0" has none. In addition to the number of cusps, we also compute the minimum and maximum distances between cusps and the stroke endpoints. These two features are used to help discriminate between strokes with cusps in close proximity to each other and ones with cusps far apart.

3.1.3 Aspect Ratio

A symbol's aspect ratio is defined as the ratio of the width to the height of its bounding box. Aspect ratios are good discriminators between tall and wide symbols. For example, in general, the letter "b" is much taller than the letter "w."

3.1.4 Intersection Features

Stroke intersection points are locations at which a stroke intersects itself. These self-intersections occur in symbols with loops such as a "2" or "8" (depending on how they are

written) and, thus, they make good discriminators between symbols with and without loops. Self-intersections can also occur when users write over their ink when making a symbol such as a "b" or "d." As with cusps, we calculate the minimum and maximum distances between self-intersections and the stroke endpoints.

3.1.5 Two-Dimensional Point Histogram

A 2D point histogram gives us a distribution of point locations within a symbol's bounding box. We break up the bounding box into an n_{row} by m_{col} grid (we use a 3×3 grid) and count the number of points in each subbox. The number of points in each subbox is then divided by the total number of points in the symbol. Since certain symbols have their points concentrated at certain locations within their bounding boxes, this histogram can be a good discriminator. In addition, it can also be a good discriminator when one symbol has a concentration of points in a subbox and another symbol has no points in that subbox. An example would be the letter "c" and the number "7."

3.1.6 Angle Histogram

The angle histogram is similar to the 2D point histogram except we use the angles between the symbol's stroke segments and the x-axis. For each stroke in the symbol, we define a vector $\vec{v_j} = p_i - p_{i-1}$ for $2 \leq i \leq n$ and $1 \leq j \leq n-1$. Given a vector $\vec{x} = (1, 0)$, we compute the angle as follows:

$$\theta_j = \arccos\left(\vec{x} \cdot \frac{\vec{v_j}}{\|\vec{v_j}\|}\right). \quad (1)$$

Each θ_j is stored in a bin depending on its value; we use a total of eight bins, breaking up the angles into 45-degree segments. Finally, each bin is divided by the number of angles. The angle histogram is a good symbol discriminator because many symbols have different angular constructions. For example, a "{" and a "3" are usually written in opposing directions, making their angle histograms different. As with the 2D point histogram, in some cases, one symbol may have a concentration of angles in one direction (between 0 and 45 degrees), and another symbol may have none at all, making for a good discrimination metric.

3.1.7 First and Last Distance

The first and last distance feature is simply the distance between the first and last points in a stroke $\|p_n - p_1\|$. If a symbol has more than one stroke, an average of the distances is used. Symbols such as "b" and "o" often start and end in a similar location meaning their first and last distance is small compared with symbols such as $\int$, "j," and "y."

3.1.8 Arc Length

Arc length is the length of a stroke and is defined as

$$l = \sum_{i=2}^{n} \|p_i - p_{i-1}\|. \quad (2)$$

If a symbol has more than one stroke, then we sum all the arc lengths from each. Many different symbols have varying arc lengths, so this is a powerful symbol discrimination feature.

3.1.9 Fit Line Feature

The fit line feature determines whether strokes are straight lines. This feature finds a least-squares approximation to a line using principal components and then uses this approximation to find the distance of the projection of the stroke points onto the approximated line. The closer this distance is to zero, the straighter the stroke. This feature works very well for symbols with straight and curvy strokes and also handles the subtlety of strokes like "(," "1," and ")."

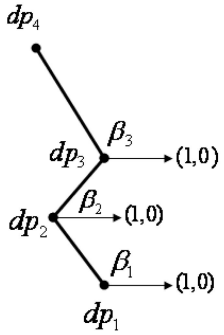

Fig. 1. Dominant point features all derive from the angles between the vector (1, 0) and the vectors defined by consecutive dominant points in the stroke.

3.1.10 Dominant Point Features

The dominant point features are a set of four angle-based features calculated using dominant points instead of stroke points. "Dominant points in strokes" are defined as the key points in a stroke, including the local extrema of curvature, the starting and ending points of a stroke, and the midpoints between these points [23]. Thus, the dominant points in a stroke are a subset of the points in that stroke. We found that dominant points provide enough information to extract these angle-based feature values while avoiding the extra variation found when computing angles using all of the stroke points.[1]

To calculate the dominant point features, we first compute a sequence of angles β_j between the x-axis and the vectors spanned by consecutive dominant points in the stroke (see Fig. 1). These angles are calculated using (1), except $\vec{v_j} = dp_i - dp_{i-1}$ for $2 \leq i \leq m_{dp}$, where dp is a dominant point and m_{dp} is the total number of dominant points.

The first feature is the maximum angle:

$$a_{max} = \max_j \beta_j. \quad (3)$$

The second feature, the average angle deviation a_{dev}, is calculated by first computing the differences ϕ_k between consecutive β_js, where $\phi_k = \beta_j - \beta_{j-1}$ for $2 \leq j \leq m_{dp} - 1$. Then,

$$a_{dev} = \frac{1}{m_{dp} - 2} \sum_{k=1}^{m_{dp}-2} \phi_k. \quad (4)$$

The third feature is the straight line ratio, calculated by counting the number of angle differences ϕ_k that are less than $|\epsilon|$ degrees (we use 3 degrees) and dividing by $m_{dp} - 2$. The last feature is the number of zero crossings, the number of times consecutive ϕ_ks go from negative to positive or positive to negative. The maximum angle, average angle

1. We could have used dominant points for the angle histogram, but in this case, we wanted as much information about a symbol's angular fluctuations as possible.

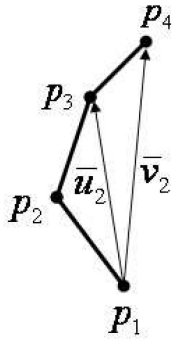

Fig. 2. One triangle in the stroke area calculation. The vectors $\vec{u_2}$ and $\vec{v_2}$ are used to compute the area of the triangle defined by points p_1, p_3, and p_4.

deviation, and number of zero crossings were all designed to discriminate between symbols with varying angular patterns, whereas the straight line ratio was developed to discern symbols that have straight lines from those that have a higher curvature.

3.1.11 Stroke Area

The stroke area feature is designed to discriminate between symbols that are roughly straight and those that are curved in some way. This feature is especially important when dealing with symbols such as "1," "(," and ")." A "1" written as a vertical line has little or no stroke area, whereas "(" and ")" have larger stroke areas. The stroke area is the area defined by vector pairs created with the initial stroke point and each of the remaining stroke points. This approach breaks up the stroke into triangles where the stroke area is simply the sum of the areas of each triangle (see Fig. 2).

To compute the stroke area, we define vectors $\vec{u_i} = p_{i+1} - p_1$ and $\vec{v_i} = p_{i+2} - p_1$ for $1 \leq i \leq n-2$. Then, the stroke area is given as follows:

$$s_{area} = \sum_{i=1}^{n-2} \frac{1}{2} (\vec{u_i} \times \vec{v_i}) \cdot \mathrm{sgn}(\vec{u_i} \times \vec{v_i}), \tag{5}$$

where $\vec{u_i} \times \vec{v_i}$ is a scalar and $\times$ represents the 2D cross product.[2] For symbols with more than one stroke, we take the average of the stroke areas.

3.1.12 Side Ratios

The side ratio features are based on the observation that the first and last points in a stroke have variable locations with respect to a symbol's bounding box. For example, a "c" has starting and ending points far from the left side of its bounding box, whereas $>$ s starting and ending points are close to the left side of its bounding box. Therefore, the starting and ending locations of a symbol can act as a good symbol discriminator. These features are calculated by taking the x-coordinates of the first and last points of a stroke, subtracting them from the left side of the symbol's bounding box (that is, the bounding box's leftmost x value) and dividing them by the bounding box width. With multistroke symbols, the averages of these ratios are taken.

2. If $\vec{u_i}$ and $\vec{v_i}$ are the first and second rows of a 2×2 matrix, the 2D cross product is simply the determinant of the matrix.

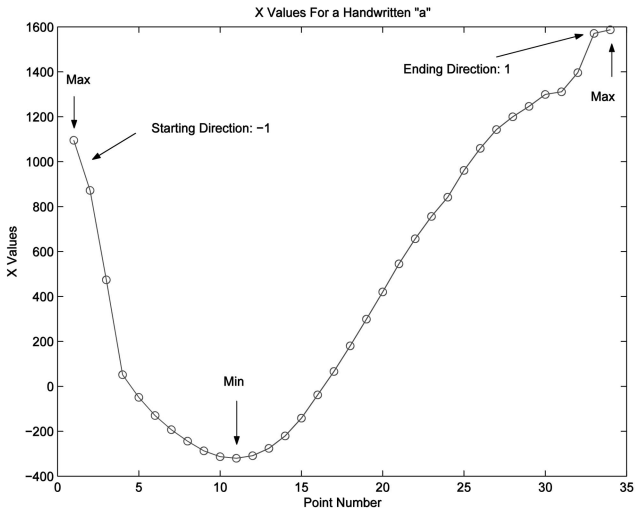

Fig. 3. The x values for the handwritten symbol "a." In this case, the two maximums are the first and last x values.

3.1.13 Top and Bottom Ratios

The top and bottom ratio features are similar to the side ratio features. In this case, the y-coordinate of the first and last points of a stroke is subtracted from the top of the symbol's bounding box (that is, the bounding box's topmost y value), and then, these values are divided by the bounding box height. With multistroke symbols, the averages of these ratios are taken.

3.1.14 Min and Max Features

The min and max features are designed to extract information from the x and y components of a stroke separately. We calculate 10 of these features; x and y versions of the number of local minima, the number of local maxima, the starting direction, the ending direction, and the length between the last direction change and the last stroke point.

As an example, Fig. 3 shows the x values of a handwritten letter "a." A distinguishing characteristic for many symbols is the direction of the strokes points as they are written along the x-axis. Thus, we can extract the number of direction changes that are made for a given symbol along the x-axis by calculating the differences $d_1 = x_i - x_{i-1}$ and $d_2 = x_i - x_l$ for $2 \leq i \leq n$, where x_l is the last x value where a direction change occurred.[3] The x values are iterated from 2 to n, and the number of local minima is then the number of times d_1 is negative and d_2 is positive, and the number of local maxima is the number of times d_1 is positive and d_2 is negative. In Fig. 3, there is one local minimum and two local maxima. Two features that are good discriminators for a variety of symbols are the starting and ending directions for a stroke along the x-axis. If the stroke's starting x values move along the $+x$ direction, a $+1$ is assigned to the starting direction, and if they move along the $-x$ direction, a -1 is assigned to the starting direction. The same process is used for the ending direction. In Fig. 3, the starting direction is -1 and the ending direction is $+1$. Finally, the length between the last x value and the last direction change is found. This feature is useful for discriminating symbols with different lengths between the last part of their strokes

3. x_l is initially set to the first x value in the stroke.

and tends to have more use for when dealing with a stroke's y values. Note that the y versions of these features are found similarly. These features are similar to the dominant point features described above, but they take a different approach because they do not use angle information.

3.2 The Pairwise AdaBoost Classifier

AdaBoost [34] takes a series of weak or base classifiers and calls them repeatedly in a series of rounds on the training data to generate a sequence of weak hypotheses. Each weak hypothesis has a weight associated with it that is updated after each round, based on its performance on the training set. A separate set of weights are used to bias the training set so that the importance of incorrectly classified examples are increased. Thus, the weak learners can focus on them in successive rounds. A linear combination of the weak hypotheses and their weights are used to make a strong hypothesis for classification.

More formally, for each unique symbol pair, our algorithm takes as input training set $(\vec{x}_1, y_1), \ldots, (\vec{x}_m, y_m)$, where each $\vec{x}_i$ represents a feature vector containing J features. Each y_i labels $\vec{x}_i$ using label set $Y = \{-1, 1\}$, and m is the total number of training samples. Since we are using a pairwise approach, our algorithm needs to train all unique pairs of symbols. For each unique pair, our AdaBoost algorithm is called on a set of weak learners, one for each element of the feature set described in Section 3.1. For example, with the 2D point histogram feature, nine weak learners are used, one for each part of the 3×3 grid. Thus, there are a total of 47 weak learners in our initial formulation.[4] We chose this approach because we found, based on empirical observation, that our features can discriminate between different symbol pairs effectively. We wanted the features to be the weak learners rather than having the weak learners act on the features themselves. Thus, each weak learner C_j uses the jth element in the $\vec{x}_i$ training samples, which is noted by $\vec{x}_i(j)$ for $1 \leq j \leq J$.

3.2.1 Weak-Learner Formulation

We use weak learners that employ a simple weighted distance metric, breaking $(\vec{x}_1, y_1), \ldots, (\vec{x}_m, y_m)$ into two parts corresponding to the training samples for each symbol in the symbol pair. Assuming that the training samples are consecutive for each symbol, we separate $(\vec{x}_1, y_1), \ldots, (\vec{x}_m, y_m)$ into $(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)$ and $(\vec{x}_{n+1}, y_{n+1}), \ldots, (\vec{x}_m, y_m)$ and define $D^1(i)$ for $i = 1, \ldots, n$ and $D^2(i)$ for $i = n+1, \ldots, m$ to be the training weights for each symbol. Note that in our formulation, D^1 and D^2 are the training weights calculated in the AdaBoost algorithm (see Section 3.2.2).

For each weak learner C_j in each feature vector $\vec{x}_i(j)$ in the training set, the weighted averages are then calculated as

$$\mu_{j1} = \frac{\sum_{k=1}^{n} x_k(j) D^1(k)}{\sum_{l=1}^{n} D^1(l)} \tag{6}$$

and

$$\mu_{j2} = \frac{\sum_{k=n+1}^{m} x_k(j) D^2(k)}{\sum_{l=n+1}^{m} D^2(l)}. \tag{7}$$

These averages are used to generate the weak hypotheses used in the AdaBoost training algorithm. If a given feature value for a candidate symbol is closer to μ_{j1}, the candidate is labeled as 1; otherwise, the candidate is labeled as -1. If the feature value is an equal distance away from μ_{j1} and μ_{j2}, we simply choose to label the symbol as 1.[5] Note that it is possible for the results of a particular weak classifier to obtain less than 50 percent accuracy. If this occurs the weak learner is reversed so that the first symbol receives a -1 and second symbol receives a 1. This reversal lets us use the weak learner's output to the fullest extent.

3.2.2 AdaBoost Algorithm

For each round $t = 1, \ldots, T * J$, where T is the number of iterations over the J weak learners, the algorithm generates a weak hypothesis $h_t : X \rightarrow \{-1, 1\}$ from weak learner C_j and the training weights $D_t(i)$, where $j = mod(t-1, J) + 1$ and $i = 1, \ldots, m$. This formulation lets us iterate over the J weak learners and still conform to the AdaBoost framework [34]. Indeed, the AdaBoost formulation allows us to select weak classifiers from different families at different iterations. We take advantage of that and force the algorithms to alternate between weak learners in Step (4) of Algorithm 1. Thus, the resulting strong classifier has an equal representation for all features.

Initially, $D_t(i)$ are set equally to $\frac{1}{m}$, where m is the number of training examples for the symbol pair. However, with each iteration, the training weights of incorrectly classified examples are increased, so the weak learners can focus on them. The strength of a weak hypothesis is measured by its error

$$\epsilon_t = Pr_{i \sim D_t}[h_t(\vec{x}_i(j)) \neq y_i] = \sum_{i:h_t(\vec{x}_i(j)) \neq y_i} D_t(i). \tag{8}$$

Given a weak hypothesis, the algorithm measures its importance using the following parameter:

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right). \tag{9}$$

With α_t, the distribution D_t is updated using the following rule:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(\vec{x}_i(j)))}{Z_t}, \tag{10}$$

where Z_t is a normalization factor ensuring that D_{t+1} is a probability distribution. This rule increases the weight of samples misclassified by h_t so that subsequent weak learners will focus on more difficult samples. Once the algorithm has gone through $T * J$ rounds, a final hypothesis

$$H(x) = \operatorname{sgn}\left(\sum_{t=1}^{T*J} \alpha_t h_t(x)\right) \tag{11}$$

is used to classify symbols, where α_t is the weight of the weak learner from round t, and h_t is the weak hypothesis from round t. If $H(x)$ is positive, the new symbol is labeled with the first symbol in the pair, and if $H(x)$ is negative, it is labeled with the second symbol in the pair (Algorithm 1 summarizes our approach). These strong hypotheses are computed for each pairwise recognizer with the labels and

4. Once the Microsoft recognizer is added into our formulation, the number of weak learners increases based on how many symbols the Microsoft recognizer can support in a user's symbol alphabet.

5. This phenomenon most often occurs in cases where feature values are Boolean (see Section 3.3).

strong hypothesis scores tabulated. To combine the results from each strong hypothesis, we use the approach suggested by Friedman [14]; the correct classification for the new symbol is simply the one that wins the most pairwise comparisons. If there is a tie, then the raw scores from the strong hypotheses are used, and the one of greatest absolute value breaks the tie.

Algorithm 1. Algorithm summary for generating a pairwise classifier using AdaBoost learning
Input: Training set $(\vec{x}_1, y_1), \ldots, (\vec{x}_m, y_m)$
Output: A pairwise recognizer
GENERATECLASSIFIER$((\vec{x}_1, y_1), \ldots, (\vec{x}_m, y_m))$
(1) Set $D_1(i) = \frac{1}{m}$ for $1 \leq i \leq m$
(2) **for** $t = 1$ **to** $T * J$ /* # of rounds */
(3) $j = mod(t-1, J) + 1$ /* index into C_j */
(4) Create weak classifier h_t from C_j using $(\vec{x}_i(j), y_i)$, and $D_t(i)$ for $1 \leq i \leq m$
(5) Calculate error ϵ_t using $\sum_{i:h_t(\vec{x}_i(j)) \neq y_i} D_t(i)$
(6) Calculate weak classifier weight α_t using $\frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
(7) Update $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(\vec{x}_i(j)))}{Z_t}$ where Z_t ensures D_{t+1} is a probability distribution
(8) **return** Strong Classifier $H(x) = \operatorname{sgn}\left(\sum_{t=1}^{T*J} \alpha_t h_t(x)\right)$

3.3 Using the Microsoft Recognizer as a Feature

Incorporating the Microsoft handwriting recognizer into our pairwise AdaBoost classifier takes advantage of its robustness and strengthens our existing feature set (see Section 3.1). Thus, we added its output as an additional feature. Since the output of the Microsoft recognizer is a symbol, we needed to convert it into a numeric quantity so that it would work as part of our recognition algorithm. We chose to encode the Microsoft recognizer's output as a vector $\vec{v}_{msft}$ of Boolean values where the number of elements in $\vec{v}_{msft}$ is equal to the number of possible symbols that the Microsoft recognizer can report.[6] $\vec{v}_{msft}$ will be all zeros except for the vector element corresponding to the recognized symbol. For example, if $\vec{v}_{msft}$ encodes the symbols $a - z$ consecutively and the Microsoft recognizer classifies an input symbol as an a, then the $\vec{v}_{msft}$ will have a 1 in its first element and a 0 for the remaining elements. A weak learner is associated with each element in the $\vec{v}_{msft}$. One concern with this approach is that for any pairwise classifier, the Microsoft recognizer will output a symbol that is not one of the two symbols in the pair. In these cases, two different possibilities can occur.

First, consider applying our algorithm to the ab-pair classifier. In the case that the Microsoft recognizer outputs only an a or b for each training sample, $\vec{v}_{msft}$ will have a 1 in its first or second element (a or b) and a 0 for all other elements in the vector. In these cases, (6) and (7) will evaluate to zero regardless of the values of D_t. The weak learners will therefore always output 1s (or -1s depending on the implementation) for each training sample, which is equivalent to random guessing. ϵ_t from (8) will be 0.5, resulting in (9) evaluating to zero. Thus, the strong hypothesis will not be affected by these weak learners. Second, it is possible that the Microsoft recognizer will output a symbol other than the symbols in a particular pairwise classifier. In these cases, the weak learner corresponding to the misrecognized symbol will have an ϵ_t that is not 0.5. The weak learner for this other symbol will have a correspondingly important effect on the strong hypothesis depending on how frequently the Microsoft recognizer reports that symbol. For example, if in the training data for a and b, the Microsoft recognizer reports some as as 9s, then the weak learner corresponding to 9 will have a potentially negative impact on the output of the ab-pair classifier. However, based on informal observations, α_t is often close to zero for all weak learners from the Microsoft recognizer feature other than the two corresponding to the symbols in the pairwise classifier. In cases where α_t is not close to zero, such as pairwise classifiers for symbols that are in a user's training set but not supported by the Microsoft recognizer (for example, $\int$ and $\sum$), our AdaBoost algorithm allows the other weak learners from the features described in Section 3.1 to compensate for any negative contributions made to the strong hypothesis from the Microsoft recognizer.

3.4 Pruning Symbol Pairs with the Microsoft Recognizer

When using the Microsoft handwriting recognizer as a stand-alone recognition engine, we observed that it stored the correctly classified symbol in its n-best list the majority of the time. To test our observation, we ran a simple experiment using four writers. Each writer wrote the symbols a-z, 0-9, (,), $-$, {, $<$, $>$, and $+$ 12 times each. The Microsoft handwriting recognizer had the correct symbol in its n-best $(n = 10)$ list 99.58 percent of the time. Based on this result, we decided to use the Microsoft handwriting recognizer as a prerecognition step.[7]

We call the Microsoft handwriting recognizer on a new handwritten symbol before proceeding to the pairwise recognizers. All the symbols from Microsoft's recognizer are collected from its n-best list. (Note that this approach works for any independent symbol recognizer that has an n-best list.) Next, any symbols that are not in the user's training data are removed, and symbols that the Microsoft recognizer cannot handle, such as $\int$, Σ, α, and so forth, are added to the symbol list. Finally, only the pairwise recognizers having these symbols are used in the main recognition step.

One of the important benefits of using this prerecognition step is that it increases the accuracy of the main recognition step by providing a reduced list of candidate symbols. Another important benefit with this approach is that it reduces the number of pairwise classifiers the algorithm needs to run. One of the issues with using the compare-all-pairs approach is that $\frac{m_s(m_s-1)}{2}$ unique comparisons are required, where m_s is the total number of symbols; if there are 40 symbols in the user's alphabet, the algorithm needs to run 780 pairwise recognizers. With the prerecognition approach, we reduce the computation time by requiring at most $\frac{(r+k+1)(r+k)}{2}$ comparisons. Note that n is the total number of alternates the Microsoft recognizer provides in its n-best list, r is the number of alternates from the n-best list that are in a user's training set, k is the number of additional symbols added that the Microsoft recognizer cannot handle, and $(r + k + 1) < m_s$ with $r \leq n$.

6. In our implementation, we, in fact, only make $\vec{v}_{msft}$ large enough to hold Boolean values for symbols that are in the user's training set. Therefore, mathematical and other symbols that are not currently available in the Microsoft recognizer are not included in $\vec{v}_{msft}$.

7. Not all writer-independent recognizers have n-best lists and we cannot use them with our current pruning approach.

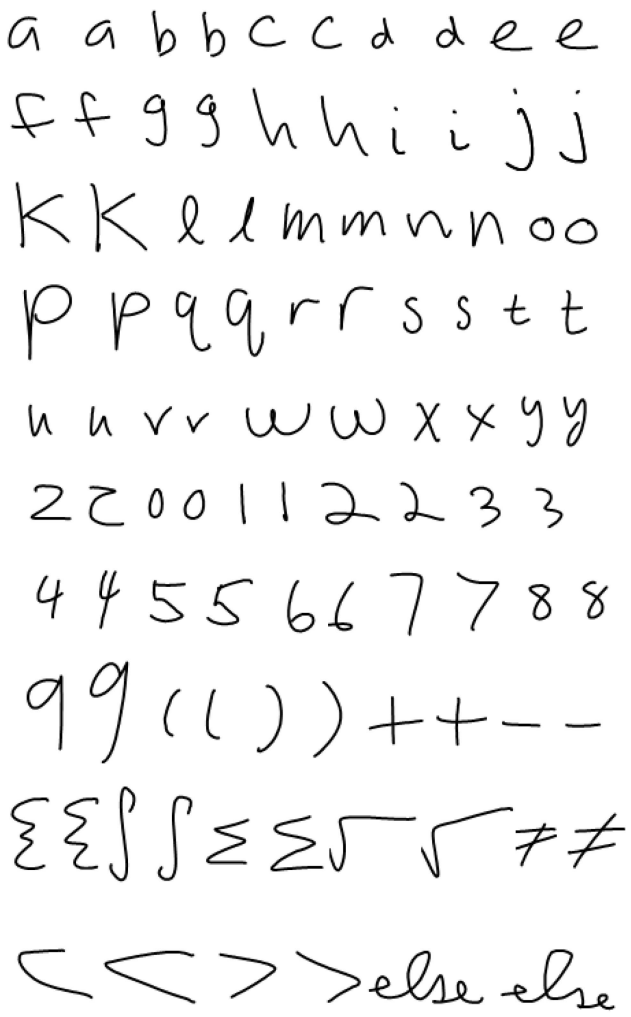

Fig. 4. Example symbols taken from the 11 test subjects used in the testing and training of our recognizer.

4 Experiment Results and Discussion

To examine the effect of augmenting our writer-dependent pairwise AdaBoost recognition engine with the Microsoft handwriting recognizer, we ran several experiments to explore our hybrid recognizer's performance. Handwriting samples from 11 subjects (seven males and four females), taken with an Hewlett-Packard Laboratories (HP) Compaq tc1100 Tablet PC, were used to conduct our experimental evaluation on 48 different symbols including a-z, 0-9, Σ, (,), $-$, $\sqrt{}$, $\int$, {, $<$, $>$, $+$, $\neq$, and $else$ (see Fig. 4). Note that we chose to include some mathematical symbols as part of our evaluation because this recognizer is designed to be part of a mathematical expression recognition system [22]. Using a simple training application, each subject provided 10 samples of each symbol to train the recognizer, and for each pairwise recognizer, $15 * J$ rounds were used for AdaBoost learning. In total, it took users about 40-45 minutes to enter their training samples and a little less than a minute for the learning algorithm to run. After training, each subject wrote each symbol 12 times for testing.

For each subject's test data, experiments were run on the pairwise AdaBoost recognizer

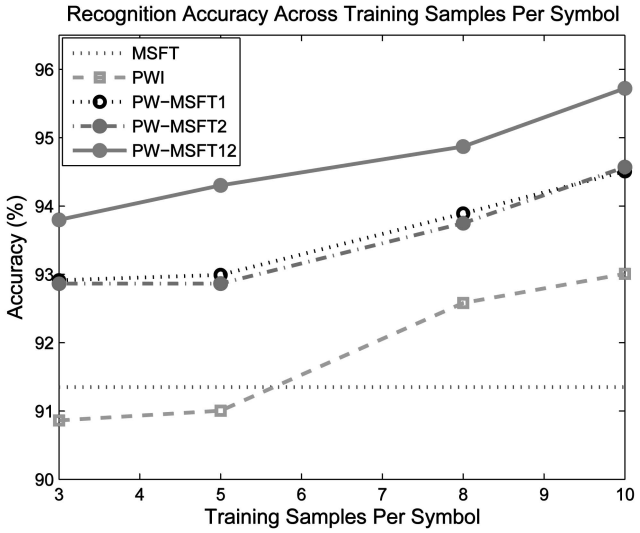

Fig. 5. Mean accuracy results shown across different numbers of training samples per symbol and the different recognizer configurations.

1. in isolation (PWI),
2. with the Microsoft recognizer used as weak learners (PW-MSFT1),
3. with Microsoft recognizer pruning (PW-MSFT2), and
4. with both weak learners and pruning using the Microsoft recognizer (PW-MSFT12),

where PWI and PW-MSFT2 use 47 weak learners $(J = 47)$ during training, and PW-MSFT1 and PW-MSFT12 use 86 $(J = 86)$. In addition, these experiments were run across different numbers of training samples per symbol (3, 5, 8, and 10) to see how reducing the training set affects recognition. We also tested the Microsoft recognizer in isolation (MSFT) to compare it against our hybrid recognizer and to see if our accuracy results were overly biased by it. Note that the Microsoft recognizer we used does not support certain mathematical symbols, so we could only test it on a-z, 0-9, (,), $-$, {, $<$, $>$, and $+$. In addition, because we were unable to tell the Microsoft recognizer to recognize only these symbols, we counted certain capital letters such as C, K, M, O, P, S, U, V, W, X, and Z as being correctly recognized since these letters are often confused with their lowercase counterparts.

Accuracy Results. A total of 6,336 symbols were tested in each of the experiments, except for the MSFT test where 5,676 symbols were tested due to the Microsoft recognizer's inability to handle some of the mathematical symbols. The mean recognition accuracies across the different recognizer configurations are summarized in Fig. 5 and their standard deviations in Table 1. MSFT has a standard deviation of 4.17 percent. Fig. 6 shows the recognition accuracies for PWI and PW-MSFT12 for each subject using 10 training samples per symbol. In particular, the graph shows that there was a 4 to 5 percentage point improvement in accuracy for subjects 5, 6, and 10. We suspect this improvement is a result of these subjects having handwriting styles that fit especially well with the combination of the Microsoft recognizer and our AdaBoost classifier. Future work is needed to verify this conjecture.

These results show that using PW-MSFT1 and PW-MSFT2 both make slight accuracy improvements over using PWI.

TABLE 1
Standard Deviations for Each Recognizer Configuration across Different Numbers of Training Samples per Symbol

Standard Deviation				
	Training Samples Per Symbol			
	3	5	8	10
PWI	3.41%	3.43%	3.34%	3.51%
PW-MSFT1	3.42%	3.07%	3.03%	3.02%
PW-MSFT2	3.0%	3.22%	3.31%	3.28%
PW-MSFT12	3.23%	2.81%	2.92%	2.77%

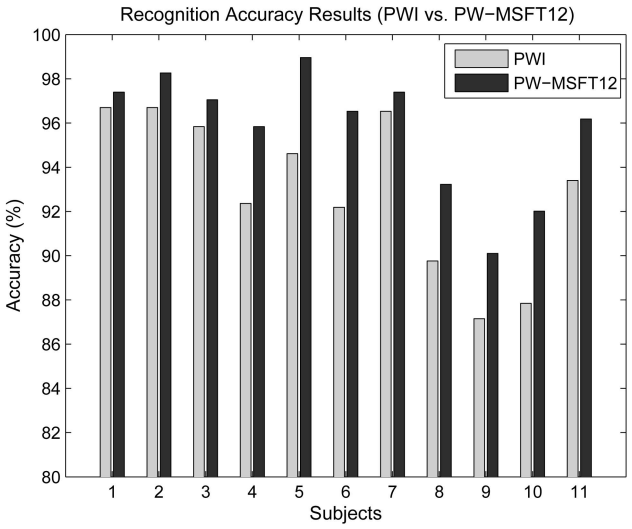

Fig. 6. Recognition accuracy results per subject using 10 training samples per symbol for the pairwise AdaBoost recognizer in isolation (PWI) and with both weak learners and pruning using the Microsoft recognizer (PW-MSFT12).

PW-MSFT12 improves the accuracy over PWI even further. PW-MSFT12 performs significantly better than PWI when using only 3 $(t_{10} = 8.23, p < 0.000006)$ and 10 $(t_{10} = 6.42, p < 0.00005)$ training samples per symbol.[8] The results when using five and eight training samples per symbol are similar. PW-MSFT12 using three samples per symbol also performs significantly better than PWI using 10 training samples per symbol $(t_{10} = 1.83, p < 0.048)$ indicating that using PW-MSFT12 can be used to reduce the number of training samples per symbol that users need to provide. This reduction has the important effect of reducing the start-up costs for using a writer-dependent recognizer while still maintaining accuracy. In our experimental setup, it takes users only 10-15 minutes to enter three samples per symbol for 48 symbols; a significant decrease in start up time over the 40-45 minutes required for writing 10 samples per symbol.

The standard deviations shown in Table 1 are all lower than the standard deviation for the MSFT test indicating that there was less fluctuation in the accuracy results for the writer-dependent configurations. These results make intuitive sense because the writer-dependent configurations are tailored toward an individual's handwriting style. Additionally, the majority of the standard deviations for the different configurations are between 3 and 3.5 percent, with the exception of the deviations for PW-MSFT12 under 5, 8, and 10 training samples per symbol. This result indicates that recognition stability is increased when using the Microsoft recognizer as a feature and as a pruning step.

These results also show that PW-MSFT12's recognition accuracy is not just a function of incorporating the Microsoft recognizer into PWI because of MSFT's lower recognition accuracy (91.35 percent) compared to the other recognition configurations. Thus, adding the Microsoft recognizer to the pairwise AdaBoost recognition engine improves accuracy by an average of 2.7 percentage points over PWI with 10 training samples required per symbol. With three training samples per symbol, PW-MSFT12 improves accuracy by an average of 2.9 percentage points over PWI.

TABLE 2
The Microsoft Handwriting Recognizer's Percent Contribution to the Total Number of α_t with Maximum Weight

MSFT Contribution		
	Samples Per Symbol	
	3	10
Subject 1	4.16%	5.7%
Subject 2	5.51%	8.97%
Subject 3	5.34%	9.1%
Subject 4	4.12%	6.71%
Subject 5	4.66%	6.41%
Subject 6	4.86%	7.01%
Subject 7	4.18%	6.79%
Subject 8	3.38%	4.49%
Subject 9	3.99%	5.54%
Subject 10	4.45%	6.24%
Subject 11	4.43%	7.51%

The values are calculated over all pairwise recognizers.

Weight Analysis. In addition to recognition accuracy, we also examined the weights α_t from the strong hypotheses from all pairwise classifiers for all 11 subjects. Specifically, we are interested in those α_t with maximum weight (calculated using ϵ_t's with minimum error[9]) because they provide the most significant contributions to the overall classification for any pair of symbols. Counting the α_t with maximum weight lets us answer two important questions: how much does the Microsoft handwriting recognizer contribute to the pairwise classifiers and how many maximum weight α_t are there for each pairwise classifier for $T = 15$ iterations over the J weak learners. We collected this data for the cases where the recognizer was trained on 3 and 10 samples per symbol and applied to PW-MSFT1 and PW-MSFT12, where $J = 86$. Table 2 shows what percentage of the weak learners with maximum α_t are from the Microsoft handwriting recognizer for all pairwise classifiers per subject. For example, with subject three, with three training samples per symbol, 5.34 percent of the weak learners with maximum α_t are from the Microsoft handwriting recognizer. Table 3 shows the average number of weak learners with maximum α_t for each iteration over J weak learners of AdaBoost training for all pairwise classifiers per subject. For example, with subject seven, 12.2 weak learners on the average have a maximum α_t for each iteration in every pairwise classifier with 10 training samples per symbol.

Tables 2 and 3 show two important general trends. First, the Microsoft recognizer appears to make more of a contribution to the pairwise classifiers with 10 training samples per symbol than with three training samples per

8. We use matched pairs for examining the mean differences for all of our statistical tests. Note the notation t_n refers to the t-distribution with n degrees of freedom and p is the significance value.

9. Minimum error in this experiment is less than $1e-8$.

TABLE 3
The Average Number of Weak Learners with Maximum α_t for Each Iteration over the J Weak Learners for All Pairwise Recognizers

Maximum Weights Per Iteration		
	Samples Per Symbol	
	3	10
Subject 1	24.5	13.2
Subject 2	21.9	11.5
Subject 3	21.6	11.2
Subject 4	23.4	14.1
Subject 5	23.3	14.1
Subject 6	21.5	11.2
Subject 7	22.4	12.2
Subject 8	21.3	11.9
Subject 9	21.4	11.4
Subject 10	20.8	11.5
Subject 11	23.1	13.4

There are 15 iterations over 86 weak learners of AdaBoost training.

symbol. Second, the average number of weak learners with maximum α_t over each iteration of J significantly decreases from 3 to 10 training samples per symbol. These results make sense since having more weak learners with maximum α_t per iteration will reduce the impact of the Microsoft handwriting recognizer. However, we expected there would be more of a contribution from the Microsoft recognizer for the three samples per symbol case. Although the Microsoft recognizer makes a contribution that leads to an accuracy improvement from PWI to PW-MSFT1, these results suggest we can improve recognition accuracy even further with three training samples per symbol. Reducing the average number of weak learners with maximum α_t would increase the Microsoft recognizer's contribution. One approach to dealing with this problem is to reduce the number of features by removing those that are highly correlated (for example, using principal component analysis). However, removing these features globally might not be sufficient since correlations between features could be different depending on the particular symbol pair. Thus, removing correlations on a pairwise basis is a promising option and we leave this for future work.

Runtime. To determine the impact of the prerecogniton step on runtime, we took a sample of the experimental runs from PWI and PW-MSFT12 using 10 training samples per symbol and recorded how long it took to recognize a given symbol. The mean runtime over 100 samples for PWI was 216.2 ms and for PW-MSFT12 was 27.69 ms. Thus, PW-MSFT12 achieves a *higher* recognition accuracy at a significantly *lower* runtime than PWI.

Ties. Finally, because our pairwise scheme makes a classification based on the class that wins the most comparisons, ties are inevitable. Thus, we explored the effect that ties had on PWI and PW-MSFT12 across the different training sample configurations. For the PWI configuration, 1.31 percent of the recognitions were ties, and of those, 52.63 percent were correct. For PW-MSFT12, only 0.76 percent of the recognitions were ties, with 63.64 percent correct. From this data, ties play only a minor role in our recognizers, but utilizing the Microsoft handwriting recognizer seems to reduce the number of ties and help improve tie-breaking accuracy.

Miscellaneous. Based on our experimental data, we believe our approach has the ability to increase accuracy of writer-dependent recognizers, reduce the number of training samples needed per symbol, and decrease online recognition time. In the future, we wish to explore more sophisticated approaches for dealing with pairwise classification such as loss-based decoding [1] and explore how an n-best list can be used as part of our feature set. We also plan to explore how to optimize our feature set for any pairwise classifier and how our approach affects other writer-dependent recognition algorithms to determine its use in more general terms.

5 Conclusion

We have presented two strategies for utilizing a writer-independent symbol recognizer in a writer-dependent recognition engine by incorporating it into a pairwise AdaBoost framework and as a preprocessing step to reduce the number of pairwise classifiers needed in the main recognition step. Although we use the Microsoft handwriting recognizer as our writer-independent recognizer, any writer-independent recognizer that provides an n-best list can be used. Our empirical results indicate the feasibility of our approach and show that we can improve the overall accuracy of the recognizer, reduce its computation time, and reduce the amount of training samples users must supply to the training algorithm. We believe this work is a good starting point toward making customizable symbol recognizers more practical.

Appendix

To facilitate the reproducibility of this work and to provide handwritten character data to the recognition community, we have converted our training and test data to UNIPEN format [16]. It can be freely downloaded at graphics.cs.brown.edu/research/pcc/symbolRecognitionDataset.zip or from as supplemental materials, which can be found at http://computer.org/tpami/archives.htm.

Acknowledgments

The authors would to thank Gregory Shakhnarovich for his valuable comments and suggestions. This work is supported in part by a gift from Microsoft and grants from the US National Science Foundation and the Joint Advanced Distributed Co-Laboratory.

References

[1] E. Allwein, R. Schapire, and Y. Singer, "Reducing Multiclass to Binary: A Unifying Approach to Margin Classifiers," *J. Machine Learning Research*, vol. 1, pp. 113-141, 2000.

[2] C. Bahlmann, B. Haasdonk, and H. Burkhardt, "On-Line Handwriting Recognition with Support Vector Machines—A Kernel Approach," *Proc. Eighth Int'l Workshop Frontiers in Handwriting Recognition*, pp. 49-54, 2002.

[3] A. Belaid and J. Haton, "A Syntactic Approach for Handwritten Formula Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 6, no. 1, pp. 105-111, Jan. 1984.

[4] A. Biem, "Minimum Classification Error Training for Online Handwriting Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 28, no. 7, pp. 1041-1051, July 2006.

[5] A. Brakensiek, A. Kosmala, and G. Rigoll, "Comparing Adaptation Techniques for Online Handwriting Recognition," *Proc. Int'l Conf. Document Analysis and Recognition*, pp. 486-490, 2001.

[6] K. Chan and D. Yeung, "Mathematical Expression Recognition: A Survey," *Int'l J. Document Analysis and Recognition*, vol. 3, no. 1, pp. 3-15, Jan. 2000.

[7] S.D. Connell and A.K. Jain, "Writer Adaptation for Online Handwriting Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 24, no. 3, pp. 329-346, Mar. 2002.
[8] S.D. Connell and A.K. Jain, "Template-Based On-Line Character Recognition," *Pattern Recognition,* vol. 34, no. 1, pp. 1-14, Jan. 2000.
[9] A.M. Day, J.R. Parks, and P.J. Pobgee, "On-Line Written Input to Computers," *Machine Perception of Pictures and Patterns,* pp. 233-240, 1972.
[10] V. Deepu, S. Madhvanath, and A.G. Ramakrishnan, "Principal Component Analysis for Online Handwritten Character Recognition," *Proc. 17th Int'l Conf. Pattern Recognition,* pp. 327-330, 2004.
[11] Y. Dimitriadis and J. Coronado, "Towards an Art-Based Mathematical Editor that Uses On-Line Handwritten Symbol Recognition," *Pattern Recognition,* vol. 28, no. 6, pp. 807-822, June 1995.
[12] A. Donahey, "Character Recognition System and Method," US patent 3,996,557, 1976.
[13] Y. Freund and R. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," *J. Computer and System Sciences,* vol. 55, no. 1, pp. 119-139, Aug. 1997.
[14] J. Friedman, "Another Approach to Polychotomous Classification," technical report, Stanford Univ., 1996.
[15] G.F. Groner, "Real-Time Recognition of Hand-Printed Symbols," *Pattern Recognition,* L.N. Kanal ed., pp. 103-108, 1968.
[16] Guyon, I.L. Schomaker, R. Plamondon, M. Liberman, and S. Janet, "UNIPEN Project of On-Line Data Exchange and Recognizer Benchmarks," *Proc. 12th Int'l Conf. Pattern Recognition,* pp. 29-33, Oct. 1994.
[17] T. Hastie and R. Tibshirani, "Classification by Pairwise Coupling," *The Annals of Statistics,* vol. 26, no. 2, pp. 451-471, Apr. 1998.
[18] R. Jarrett and P. Su, *Building Tablet PC Applications.* Microsoft Press, 2003.
[19] D. Kerrick and A. Bovik, "Microprocessor-Based Recognition of Hand-Printed Characters from a Tablet Input," *Pattern Recognition,* vol. 21, no. 5, pp. 525-537, May 1988.
[20] M. Koschinski, H.-J. Winkler, and M. Lang, "Segmentation and Recognition of Symbols within Handwritten Mathematical Expressions," *Proc. Int'l Conf. Acoustics, Speech, Signal Processing,* pp. 2439-2442, 1995.
[21] A. Kosmala and G. Rigoll, "On-Line Handwritten Formula Recognition Using Statistical Methods," *Proc. 14th Int'l Conf. Pattern Recognition,* pp. 1306-1308, 1998.
[22] J. LaViola, "Mathematical Sketching: A New Approach to Creating and Exploring Dynamic Illustrations," PhD dissertation, Dept. of Computer Science, Brown Univ., May 2005.
[23] X. Li and D. Yeung, "On-Line Handwritten Alphanumeric Character Recognition Using Dominant Points in Strokes," *Pattern Recognition,* vol. 30, no. 1, pp. 31-44, Jan. 1997.
[24] R. Marzinkewitsch, "Operating Computer Algebra Systems by Hand-Printed Input," *Proc. Int'l Symp. Symbolic and Algebraic Computation,* pp. 411-413, 1991.
[25] C. Mathis and T.M. Breuel, "Classification Using a Hierarchical Bayesian Approach," *Proc. 16th Int'l Conf. Pattern Recognition,* pp. IV: 103-106, 2002.
[26] N.E. Matsakis, "Recognition of Handwritten Mathematical Expressions," master's thesis, Dept. of Electrical Eng. and Computer Science, Massachusetts Inst. of Technology, 1999.
[27] E. Miller and P. Viola, "Ambiguity and Constraint in Mathematical Expression Recognition," *Proc. 15th Nat'l Conf. Artificial Intelligence,* pp. 784-791, 1998.
[28] Y. Nakayama, "A Prototype Pen-Input Mathematical Formula Editor," *Proc. World Conf. Educational Multimedia and Hypermedia,* pp. 400-407, 1993.
[29] R. Plamondon and S.N. Srihari, "On-Line and Off-Line Handwriting Recognition: A Comprehensive Survey," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 22, no. 1, pp. 63-84, Jan. 2000.
[30] V.M. Powers, "Pen Direction Sequences in Character Recognition," *Pattern Recognition,* vol. 5, pp. 291-302, Mar. 1973.
[31] L. Prevost and M. Milgram, "Automatic Allograph Selection and Multiple Expert Classification for Totally Unconstrained Handwritten Character Recognition," *Proc. 14th Int'l Conf. Pattern Recognition (ICPR '98),* pp. 381-383, 1998.
[32] D. Rubine, "Specifying Gestures by Example," *Proc. ACM 18th Ann. Conf. Computer Graphics and Interactive Techniques,* pp. 329-337, 1991.
[33] P. Sarkar and G. Nagy, "Style Consistent Classification of Isogenous Patterns," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 27, no. 1, pp. 88-98, Jan. 2005.
[34] R. Schapire, "A Brief Introduction to Boosting," *Proc. 16th Int'l Joint Conf. Artificial Intelligence,* pp. 1401-1406, 1999.
[35] H. Schwenk and Y. Bengio, "AdaBoosting Neural Networks: Application to On-Line Character Recognition," *Lecture Notes in Computer Science,* vol. 1327, pp. 967-972, 1997.
[36] M. Shilman, P. Viola, and K. Chellapilla, "Recognition and Grouping of Handwritten Text in Diagrams and Equations," *Proc. Ninth Int'l Workshop Frontiers in Handwriting Recognition,* pp. 569-574, 2002.
[37] S. Smithies, K. Novins, and J. Arvo, "A Handwriting-Based Equation Editor," *Proc. Graphics Interface Conf.,* pp. 84-91, 1999.
[38] J. Subrahmonia, K. Nathan, and M. Perrone, "Writer Dependent Recognition of Online Unconstrained Handwriting," *Proc. Int'l Conf. Acoustics, Speech, and Signal Processing,* vol. 6, pp. 3478-3481, 1996.
[39] C. Tappert, C.Y. Seun, and T. Wakahara, "The State of the Art in On-Line Handwriting Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 12, no. 8, pp. 787-808, Aug. 1990.
[40] E. Weisstein, *CRC Concise Encyclopedia of Mathematics.* Chapman and Hall/CRC, 1998.
[41] H.-J. Winkler, "Symbol Recognition in Handwritten Mathematical Formulas," *Proc. Int'l Workshop Modern Modes of Man-Machine Comm.,* pp. 7/1-7/10, June 1994.

Joseph J. LaViola Jr. received the ScM degree in computer science in 2000, the ScM degree in applied mathematics in 2001, and the PhD degree in computer science in 2005 from Brown University. He is an assistant professor in the School of Electrical Engineering and Computer Science, University of Central Florida, and an adjunct assistant research professor at the Computer Science Department, Brown University. His primary research interests include pen-based interactive computing, 3D interaction techniques, predictive motion tracking, multimodal interaction in virtual environments, and user interface evaluation. His work has appeared in journals such as *Presence* and the *IEEE Computer Graphics and Applications*, and he has presented research at conferences including the ACM International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH), the ACM Symposium on Interactive 3D Graphics, the IEEE Virtual Reality Conference, and the Eurographics Symposium on Virtual Environments. He has also coauthored *3D User Interfaces: Theory and Practice*, the first comprehensive book on 3D user interfaces. He is a member of the IEEE and the IEEE Computer Society.

Robert C. Zeleznik received the MSc in computer science from Brown University, where he developed the gestural SKETCH 3D modeling system. He is the director of research for Brown University's Computer Graphics Group and the Microsoft Center for Research on Pen-Centric Computing at Brown University. His overarching research interest is the design of 2D and 3D post-WIMP user interfaces. Currently, his focus is on symbolic and diagrammatic pen-centric computing.