PERGAMON

Pattern Recognition 34 (2001) 1–14

PATTERN RECOGNITION
THE JOURNAL OF THE PATTERN RECOGNITION SOCIETY

www.elsevier.com/locate/patcog

Template-based online character recognition

Scott D. Connell*, Anil K. Jain

Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824, USA

Received 5 November 1998; received in revised form 16 August 1999; accepted 16 August 1999

Abstract

Handwriting is a common, natural form of communication for humans, and therefore it is useful to utilize this modality as a means of input to machines. One well-known method of classifying individual characters or words is template matching. We demonstrate a template-based system for online character recognition where the number of representative templates is determined automatically. These templates can be viewed as representing different styles of writing a particular character. The templates are then used as a reference for efficient classification using decision trees. Overall, our classifier achieves an 86.9% accuracy on a set of 17,928 alphanumeric characters (36 classes; 10 digits and 26 lowercase letters) with a throughput of over 8 characters per second on a 296 MHz Sun UltraSparc. © 2000 Pattern Recognition Society. Published by Elsevier Science Ltd. All rights reserved.

Keywords: Clustering; String matching; Online handwriting; Prototypes; Decision trees

1. Introduction

The amount of information that can be stored and processed by desktop computers is increasing at a tremendous rate. Given this rate of increase, the ease at which information can be exchanged with a user becomes a serious bottleneck. In order to be effective, user interfaces should be (1) efficient and (2) natural to the user, thereby requiring little or no learning curve for the user. While there has been much progress on machine presentation of data to humans, such as data visualization tools, the primary mode of data input from a human to a computer is the keyboard. Speech recognition and handwriting recognition utilize other, more natural, forms of human communication, which have recently been integrated in many consumer products (e.g., Apple's Newton Messagepad, the CrossPad by A.T. Cross and IBM, Interactive Voice Response Units (IVRUs) used by many telephone companies, etc). However, for these input modalities to be economical and user-friendly, their recognition accuracy must be sufficiently high such that the user needs to make only a minimal number of corrections to the recognized text or speech.

Handwriting recognition can be broken into two fields which differ in the form in which the data is presented to the system. In *off-line* handwriting recognition, the user writes on paper which is later digitized by a scanner. The data is presented to the system as an image, requiring a segmentation of the writing from the image background before recognition can be done. In contrast, the field of *online* handwriting recognition requires that the user write on a digitizing tablet using a special stylus, so that the user's written strokes are captured as they are being formed by sampling the pen's (x, y) coordinates at evenly spaced time intervals. The use of a pressure-sensitive switch on the pen tip indicates pen-up/pen-down status and disambiguates stroke segmentations.

To perform classification of handwritten characters, a recognition system must attempt to leverage between class variations, while accommodating potentially large within-class variations. If a recognition system is to work well for a large number of different writers (a writer-independent system), this within-class variation can be very large. The variance within a particular character

* Corresponding author. Tel.: + 1-517-355-9319; fax: + 1-517-432-1061.

E-mail addresses: connell@cse.msu.edu (S.D. Connell), jain@cse.msu.edu (A.K. Jain).

PII: S0031-3203(99)00197-1

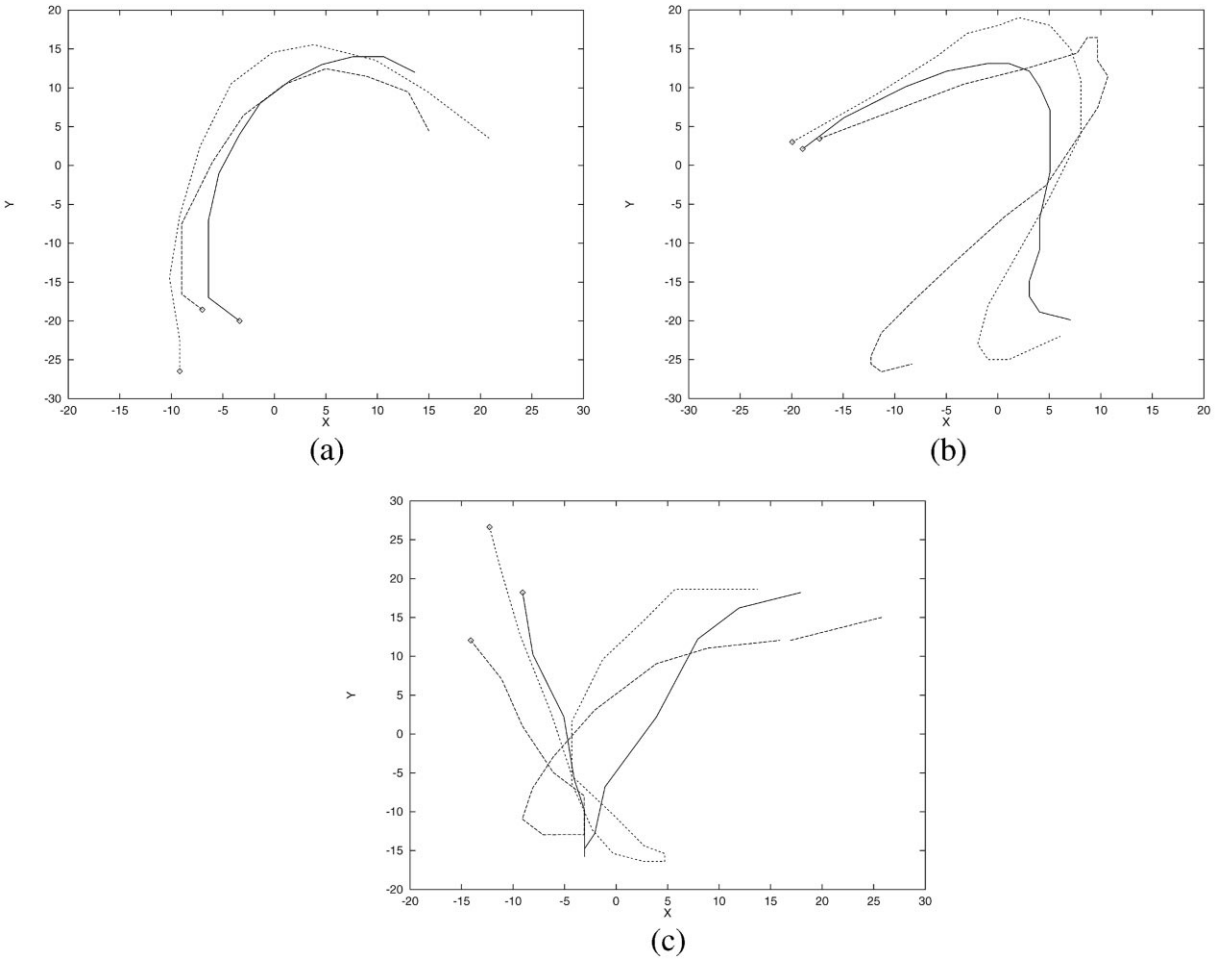


Fig. 1. Variations within three different writing styles for the lowercase character "r". Each box, (a), (b), and (c), show three examples of the same writing style, with a different style represented in each box. The beginning of each stroke is shown by a dot.

class can be thought of as made up of two components: the variations between writing styles, and the variations within a writing style. This is illustrated in Fig. 1. Our goal is to automatically learn these writing styles from a given data set, so that adequate character models can be built. In this paper, we attempt to first find an appropriate representation for each character class by modeling the individual writing styles, and then make use of these models to train a classifier to discriminate between character classes. Section 2 gives a background to the problem of online handwriting recognition. Section 3 describes our template-based method of character recognition, including two methods of data reduction: cluster center representation (Section 3.3.1), and training set editing (Section 3.3.2). We present results on two different classifiers which are discussed in Section 4: nearest neighbor, and decision trees. Section 5 presents our experimental results, and our conclusions are discussed in Section 6.

2. Background

With online handwritten data, the trace of a writer's pen is stored as a sequence of points sampled at equally spaced time intervals. The information captured for each sample is the (x, y) coordinates of the pen on the digitizing tablet. It has been shown [1] that utilizing this information about the pen dynamics can lead to better recognition accuracies than applying OCR techniques alone.

Bellegarda et al. [2] identify five categories of online handwriting recognition techniques: (i) primitive decomposition, (ii) motor models, (iii) elastic matching, (iv) stochastic models, and (v) neural networks. The advantages and disadvantages of each of these techniques are summarized in Table 1.

Primitive decomposition identifies sub-strokes that form common building blocks for characters. Examples of such building blocks are loops, dots, crossovers, arcs, ascenders, and descenders. This method generally

Table 1
Summary of techniques for the recognition of online handwriting

Technique	Advantages	Disadvantages
Primitive decomposition [3,4]	Powerful high-level features	Not very robust to large variations in writing style
Motor models [5–7]	Takes advantage of pen dynamics	May lack robustness when writing style variations are large
Elastic matching [8–10]	Works very well for writer-dependent data writer-dependent data Does not require a relatively large amount of training data	Does not generalize well for writer-independent tasks Classification time grows linearly with the number of training examples
Stochastic models [2,13]	Models temporal relations well	Requires a large amount of training data
Neural networks [14–16]	Classification time is fast	Does not model temporal relations very well

requires a pre-segmentation of the strokes into sub-stroke pieces. Word recognition is performed by matching identified sub-stroke sequences to previously observed sequences for words using such methods as dictionary lookup [3] or hidden Markov models [4].

Motor models [5–7] are a technique commonly used in what is known as *Analysis by Synthesis* in which models of stroke segments are created along with rules for connecting them to form characters. Motor models represent these stroke segments as parameterized models of the motion of the pen tip, simulating the physical properties of human hand motion.

Elastic matching [8–10] works on the sequence of sample points directly by searching for an alignment of data points between an input character, and some template character. Duin et al. [11] have referred to such data as "featureless". The distance between an input character and a template is taken as the sum of distances between aligned points. Classification can then be done using a nearest-neighbor classifier. A "featureless" representation of offline characters has been demonstrated by Jain and Zongker [12], in which they used deformable models to find the similarity between characters.

Stochastic models are often used in a similar fashion as elastic matching in that they represent the data in terms of a temporal sequence. The most common such method is to represent each class using a hidden Markov model. These models are often created using features extracted from the individual sample points [13], or from the points that are contained within a *sliding window* which slides along the sample point sequence thereby producing a sequence of features [2].

Time delay neural networks are often used to recognize characters or character segments as a sliding window passes over their temporally sampled sequence. Features extracted from the sample points in the sliding window are passed to the input layer of a feed-forward neural network. The activation level of each output node, one per class identity, approximates the likelihood that the sequence of points in the sliding window belongs to that class. This then produces a sequence of likelihood values, which can be used to find the best sequence of character identities using methods such as non-linear *dynamic time warping* algorithms [14] and hidden Markov models [15]. Other neural network classifiers [16] have combined both stroke based features, and OCR-type features, in which a character is converted into a pixel array and features are taken as the pixel values in different regions of the image.

Table 2 presents some recent results from the literature for the isolated character recognition problem, where segmentation is assumed to have been correctly done. It should be stressed that these different studies have used different datasets, and this makes a comparison of these results difficult. Recently, results have been reported using data from the Unipen dataset [18]. The Unipen project was formed to gather data from many different organizations to eventually form a publicly available dataset which would enable researchers to report results that are comparable to each other. At the time of this writing however, this database is not yet publicly available. The dataset used in our experiments is particularly difficult in that it contains both discretely written and manually segmented cursive characters in a variety of handwriting styles. Some examples of these characters are shown in Fig. 2.

3. Template-based approach

Our method of online character recognition is based on template matching using a string matching distance measure. We describe two methods of data reduction, one based on a clustering of the data, and the second based on editing the full training set to retain only those templates that lie near the class boundaries. An overview of the system is shown in Fig. 3.

Table 2
Recent results from the literature on presegmented character recognition

Author	Method	Accuracy	Notes
Li and Yeung [8]	Nearest neighbor using elastic matching	Upper and lowercase classifier: 87.1% on 780 lowercase, 92.9% on 780 uppercase Digit classifier: 96.3% on 300 digits	0.35 s/char. 180 reference templates selected from 840 examples
Scattolin and Krzyzak [9]	Nearest neighbor using weighted elastic matching	88.67% on 1650 digits	33 writers used 293 reference templates
Yaeger et al. [16]	Combined online and offline neural network classifier	95-class classifier (52 chars, 10 digits, 33 symbols): 86.1%	45 writers used
Chan and Yeung [17]	Manually designed structural models	97.4% on 9300 upper and lowercase chars	150 writers used
Li et al. [19]	Hidden Markov models	92% (after 3.9% reject) on 3126 digits	Unipen data used
Prevost and Milgram [20]	Combined online and offline nearest-neighbor classifiers	digits: 98.70% uppercase: 97.81% lowercase: 96.84%	Total of 11,162 chars from Unipen data used

Fig. 2. Examples of characters in our dataset.

3.1. Preprocessing

Strokes captured by a digitizing tablet tend to contain a small amount of noise, most likely introduced by the digitizing device, causing the sampled curves to be somewhat jagged. In order to reduce this noise, some form of smoothing is often applied such as a smoothing spline filter [21], or a Yulewalk filter for low-pass filtering of the data [22]. Most preprocessing techniques produce a resampling of the data so that points are equi-distant in space rather than time. This provides a time normalization, without which slowly written strokes would contain a much larger number of sample points than quickly written strokes of the same shape. In our approach, we have applied an equi-distant resampling of the data, followed by a Gaussian filter applied independently to each of the x and y coordinates of the point sequence:

$$x_t^{filtered} = \sum_{i=-3\sigma}^{3\sigma} w_i x_{t+i}^{orig}, \tag{1}$$

where

$$w_i = \frac{e^{-i^2/2\sigma^2}}{\sum_{j=-3\sigma}^{3\sigma} e^{-j^2/2\sigma^2}}. \tag{2}$$

This is defined similarly for $y_t^{filtered}$. Throughout both of these operations, there are certain critical points along the curve whose locations should be preserved, such as endpoints and points of high curvature. These points are detected and included with the resampled points. In addition, size normalization is applied such that all digits have the same height, but retain their original aspect ratio.

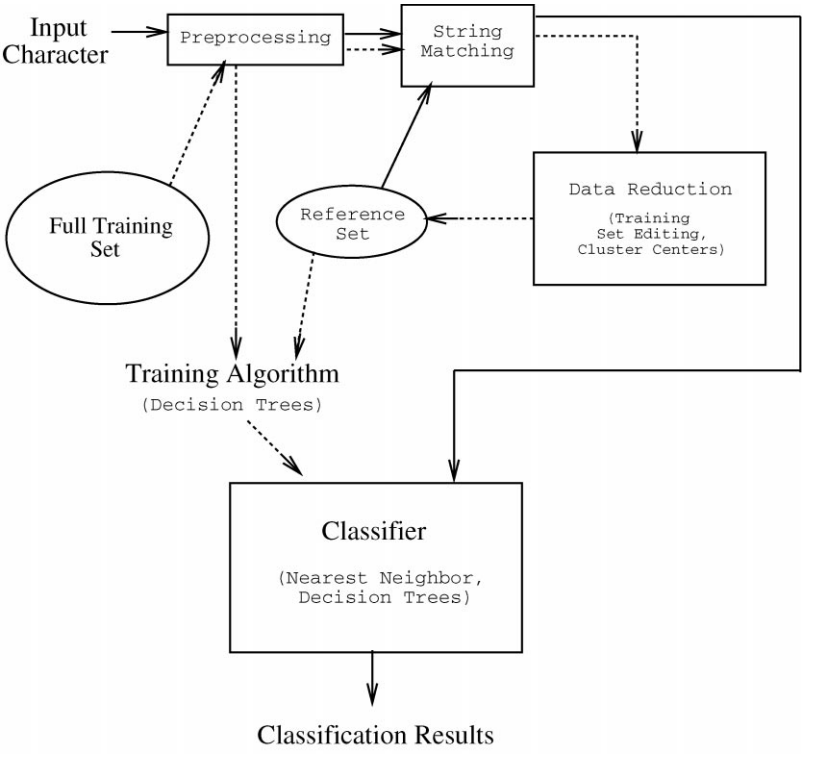


Fig. 3. Online character recognition system. The flow of data during training is shown by the dashed line arrows, while the data flow during recognition is shown by the solid line arrows.

3.2. String matching

A stroke is represented as a sequence of events (feature vectors, which will be described later), corresponding to the sequence of sample points in the stroke. This sequence forms a variable-length string with an average of about 62 events after preprocessing. The distance between two different strings, A and B, involves computing the distance between the corresponding pair of events e_i^A and e_j^B. This requires an alignment of the events between the two strings, and calculating the distances between the individual pairs of aligned events. This string matching technique is used to provide a distance measure between character pairs. Similar methods under the name *elastic template matching* have been applied to online handwritten data [10].

In our approach, adopted from Jain et al. [23], each event is represented with three measurements: the x and y offsets with respect to some reference coordinate, and the angle of curvature of the written stroke at the sample point. We define the reference coordinate as the first sample point of the digit's first stroke. Table 3 shows this string representation for three different characters. Given an alignment of the events between two strokes, the distance, $d_E(i,j)$, between a pair of events, $e_i^A = (x_i^A, y_i^A, \theta_i^A)$ from stroke A and $e_j^B = (x_j^B, y_j^B, \theta_j^B)$ from stroke B, is defined as a linear combination of the respective differences between the two events of the three measurements taken for each event:

$$d_x(i,j) = |(x_i^A - x_1^A) - (x_j^B - x_1^B)|, \tag{3}$$

$$d_y(i,j) = |(y_i^A - y_1^A) - (y_j^B - y_1^B)|, \tag{4}$$

$$d_\theta(i,j) = Min(|\theta_i^A - \theta_j^B|,\ 360 - |\theta_i^A - \theta_j^B|), \tag{5}$$

$$d_E(i,j) = \alpha d_x(i,j) + \beta d_y(i,j) + \gamma d_\theta(i,j), \tag{6}$$

where all θ are in the range (0, 360), and α, β, and γ are the weights of the linear combination. These weights are empirically determined, but we have observed that placing a large weight on $d_\theta(i,j)$ results in a better classification accuracy.

The distance between two characters is then the sum of the distances between each pair of corresponding points from the strings, given some alignment of the points. For our experiments, characters containing multiple strokes (*delayed strokes*) are handled by creating a connecting segment from the last point of a stroke to the first point of the next stroke, and therefore all characters are represented as a single string.

Table 3
The string representation ($e_t = (x_t, y_t, \theta_t)$) after preprocessing of the three digits from Fig. 4 — (a) the digit "2", (b) the digit "3" of Fig. 4 — (a), (c) the leftmost digit "3" of Fig. 4(b).

(0, 0, 223), (− 2, − 1, 223), (− 1, 2, 223), (0, 5, 182), (1, 9, 194), (3, 13, 193), (5, 16, 190), (7, 19, 201), (10, 22, 239), (14, 24, 289), (16, 21, 261), (16, 17, 196), (16, 13, 180), (16, 9, 188), (16, 5, 189), (15, 1, 182), (15, − 2, 187), (14, − 5, 178), (13, − 9, 172), (13, − 13, 187), (12, − 17, 193), (11, − 21, 189), (9, − 25, 183), (8, − 28, 186), (6, − 32, 196), (4, − 35, 190), (1, − 38, 185), (− 1, − 41, 207), (− 4, − 43, 211), (− 8, − 44, 218), (− 12, − 45, 254), (− 15, − 42, 255), (− 16, − 38, 240), (− 15, − 34, 237), (− 12, − 31, 232), (− 9, − 30, 220), (− 5, − 30, 211), (− 1, − 31, 196), (2, − 33, 172), (6, − 34, 172), (9, − 35, 179), (13, − 36, 172), (17, − 37, 159), (21, − 37, 155), (25, − 36, 155), (27, − 35, 155)

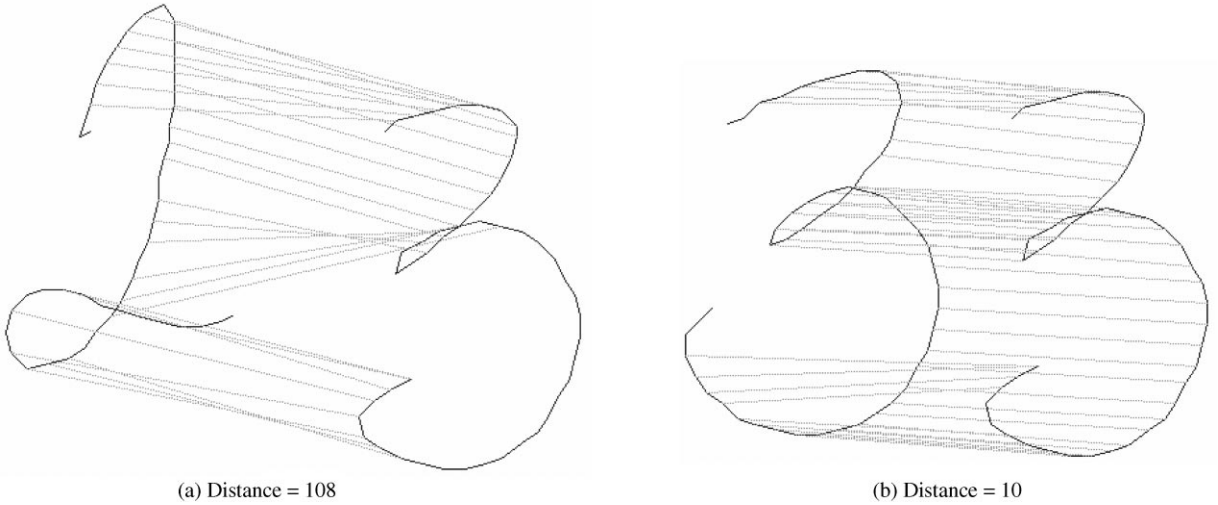
(a)

(0, 0, 192), (2, 2, 192), (6, 3, 192), (10, 4, 187), (14, 5, 201), (18, 5, 216), (22, 4, 232), (25, 1, 251), (25, − 1, 232), (24, − 5, 200), (22, − 9, 199), (20, − 12, 195), (17, − 15, 191), (14, − 18, 186), (11, − 20, 180), (8, − 23, 186), (5, − 25, 264), (2, − 27, 343), (3, − 23, 286), (7, − 21, 207), (10, − 19, 195), (14, − 18, 203), (18, − 17, 208), (22, − 18, 203), (26, − 19, 205), (29, − 21, 211), (32, − 24, 201), (34, − 28, 192), (36, − 31, 202), (37, − 35, 212), (37, − 39, 203), (36, − 43, 196), (35, − 47, 195), (33, − 50, 186), (31, − 54, 195), (29, − 57, 205), (26, − 59, 196), (22, − 62, 200), (19, − 63, 202), (15, − 64, 202), (11, − 64, 201), (7, − 63, 196), (3, − 62, 199), (0, − 60, 213), (− 3, − 58, 253), (− 4, − 54, 259), (− 1, − 51, 221), (1, − 49, 199), (5, − 47, 199), (5, − 47, 199)

(b)

(0, 0, 61), (− 3, − 1, 61), (0, 0, 61), (3, 3, 193), (7, 4, 186), (11, 6, 185), (14, 7, 189), (18, 8, 188), (22, 9, 209), (26, 9, 243), (29, 7, 255), (30, 3, 230), (29, 0, 204), (28, − 4, 204), (26, − 7, 196), (23, − 10, 180), (21, − 13, 191), (18, − 16, 195), (15, − 18, 184), (11, − 21, 189), (8, − 23, 260), (5, − 24, 330), (6, − 21, 276), (9, − 18, 207), (12, − 16, 199), (16, − 14, 209), (20, − 13, 215), (24, − 14, 213), (27, − 15, 210), (30, − 18, 197), (33, − 21, 201), (35, − 24, 205), (36, − 28, 196), (37, − 32, 203), (37, − 36, 203), (36, − 40, 196), (35, − 44, 195), (33, − 47, 192), (31, − 51, 201), (28, − 54, 201), (25, − 56, 196), (22, − 58, 199), (18, − 59, 196), (14, − 60, 201), (10, − 60, 201), (6, − 59, 188), (2, − 58, 199), (− 1, − 57, 204), (− 4, − 54, 192), (− 7, − 52, 201), (− 9, − 49, 210), (− 11, − 45, 234), (− 11, − 41, 240), (− 8, − 38, 240), (− 6, − 36, 240)

(c)



(a) Distance = 108 (b) Distance = 10

Fig. 4. The alignment between two digits, and the resulting matching score for (a) a '2' compared with a '3', (b) comparing two '3's. Dotted lines are drawn to indicate corresponding aligned points between the two digits.

We place a restriction that prohibits alignments in which two or more events in one string map to a single event in the second string. Alternate alignments are created by two methods: (1) skipping an event from the first string if it is determined that, for the given alignment, the event is spurious, (2) skipping an event in the second string if it is determined that, for the given alignment, the corresponding event in the first string is missing. In each

case, we add a penalty to the total distance between the strings, respectively, called spurious penalty and missing penalty. These penalties also act as threshold values on the distance between two events in determining if a spurious or missing event exists.

The distance between two strings is calculated by considering all possible alignments of events in the two strings, and finding the alignment for which the total distance is minimum using dynamic programming. The calculation of the distance between two strings, A and B, is shown here in terms of the calculation of the distances between a set of events, e_i^A and e_j^B:

$$D(i,j) = Min \begin{cases} D(i-1,j-1) + d_E(i,j), & \\ D(i-1,j) + \text{missing penalty}, & 1 \leqslant i \leqslant N_A, \\ D(i,j-1) + \text{spurious penalty} & 1 \leqslant j \leqslant N_B, \\ D(i-1,j-1) + \text{missing penalty} & \\ \quad + \text{spurious penalty}, & \end{cases} \tag{7}$$

$$Dist(A,B) = D(N_A, N_B). \tag{8}$$

For our experiments, we have set missing penalty = spurious penalty. The final distance between two strings uses an additional global measurement: a stroke count difference penalty term

$$Dist_{SP}(A,B) = \frac{Dist(A,B)^2}{Norm\_Factor(N_A,N_B)} + (SP)|S_A - S_B|, \tag{9}$$

where S_A (S_B) is the number of strokes that make up digit A (B), SP is the stroke penalty, and $Norm\_Factor(N_A, N_B)$ is the maximum possible distance between any two strings of lengths N_A and N_B scaled by a constant factor.

3.3. Data reduction

Selection of representative prototypes from the training set is presented in this section. Two methods, cluster centers and training set editing, are described.

3.3.1. Cluster centers

One of the methods of identifying representative prototypes from the training data is to cluster the data and retain the pattern (prototype) closest to each cluster center. This can be viewed as a technique of separating the training data based on writing styles and deriving representative examples for these styles. Since our data is "featureless", each character is represented by its distance to every other character, with the same class identity, in the training set. Using the distances calculated by our string matching method, a pattern matrix is constructed for each class of characters. Clusters are derived using a squared-error clustering algorithm, called CLUSTER [24].

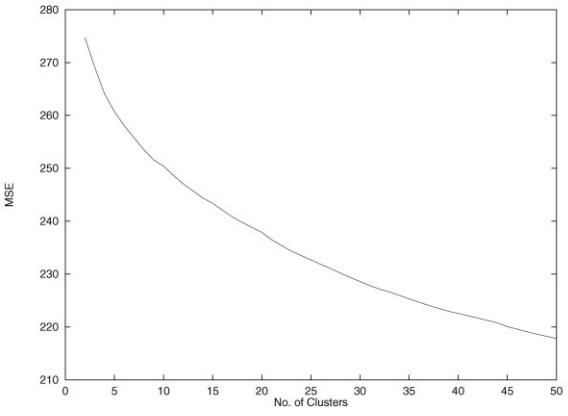


Fig. 5. Mean-squared error for different values of K, the number of clusters, for the character "m".

CLUSTER attempts to produce the best clustering for each value of K, where K is the number of clusters into which the data is to be partitioned. Fig. 5 shows an example of the number of clusters plotted against the mean-squared error for the character class "m". As expected, the mean-squared error decreases monotonically as a function of K. An inspection of Fig. 5 shows that identification of a "knee" in the curve is not easily accomplished. This example is typical for most characters, making the choice of K very difficult by this method. Automatically determining the number of clusters that are present in a data set is a very difficult problem [24]. We use the method proposed by Davies and Bouldin [25] to select an "optimal" value of K. Fig. 6 shows the prototypes closest to the centers of three clusters, each identified for characters "f" and "s", and four clusters identified for the character "z".

3.3.2. Training set editing

The goal of nearest-neighbor editing algorithms is to select only those templates from the training set that fall along the class boundaries in the feature space. These selected prototypes then make up the reduced training set. Common methods of identifying this reduced set involve first deriving the Voronoi Tessellation of the training set, and then removing any training example whose cell does not border the cell of a training pattern of a differing class [26].

A Voronoi Tessellation requires that the training patterns be represented as feature vectors in some d-dimensional Euclidean space. Since our patterns are "featureless", meaning that we have only their inter-pattern distances, we have used the following algorithm to identify

the border patterns. This method relies only on knowing the inter-class nearest neighbor to a character example.

Algorithm for constructing the Reduced Training Set

$T_{Orig} \equiv$ Original training set
$T_R \equiv$ Reduced training set
$C \equiv$ Set of class labels
$C(x) \equiv$ The class of pattern x
$NN_c(x) \equiv$ The nearest neighbor template to pattern x, that is from class c

$T_R = \emptyset$
$\forall x_i \in T_{Orig}$
  $\forall c \in C$, where $c \neq C(x_i)$
    If $NN_c(x_i) \notin T_R$
      $NN_c(x_i) \rightarrow T_R$

The output of this algorithm, set T_R, contains only those patterns which have been identified as the nearest neighbor of another pattern which is across a class boundary; in other words, those training patterns which are near the class boundaries. However, this method retains many more training patterns than the clustering method described in the previous section.

4. Classification

We discuss two different methods of classification: nearest neighbor and decision trees. Both of these methods can be used in combination with the data reduction techniques from the previous section.

Nearest-neighbor classification is a common technique [26] used in template-based approaches. We make use of the string matching measure described in Section 3.2 as the distance measure between patterns.

Decision trees [27] take a "divide and conquer" approach to solve a complex classification problem and attempt to identify those features which provide the most discriminating information. In order to represent a character as a fixed length vector of features, the distance from a given character to each of the M representative templates (which will henceforth be referred to as the reference set characters) was measured giving the following feature vector: $[D_{c,1}, D_{c,2}, \ldots, D_{c,M}]$, where $D_{c,i}$ is the distance between a character, c, and the ith reference character. These features shall be referred to as the *similarity* features and they can be used to partition the training data at each node of a decision tree. For example, at node, n, when feature $D_{c,i(n)}$ is used to partition the data based on threshold $T_{D_{c,i(n)}}$, the following rule will be applicable:

If $D_{c,i(n)} < T_{D_{c,i(n)}}$, traverse the left branch of the node n

otherwise, traverse the right branch of the node n.

In such a case, characters which traverse the left branch can be said to be similar to the $i(n)$th reference character, and therefore may belong to the cluster corresponding to this reference character, while the characters that traverse the right branch can only be said to be dissimilar to the $i(n)$th reference character, creating an imbalance in the division of data between the two branches.

This imbalance motivates a second type of feature which can be used to partition the template data into two groups at each node. By measuring the distance of a character, c, to two reference characters rather than only one at each node, we can ask the following question: "Is character c more similar to reference i, or to reference j?". To pose such a question as a vector of continuously valued features, we define

$$[D_{c,i,j}], \quad i = 1, \ldots, M, j = 1, \ldots, M,$$

where

$$D_{c,i,j} = D_{c,i} - D_{c,j}.$$

These features shall be referred to as the *difference* features. When constructing a decision tree, at any node, n, we choose a pair of reference characters, $i(n)$ and $j(n)$, with corresponding feature $D_{c,i(n),j(n)}$, and a threshold $T_{D_{c,i(n),j(n)}}$ for partitioning the data based on the following rule

If $D_{c,i(n),j(n)} < T_{D_{c,i(n),j(n)}}$, traverse the left branch of the node n

otherwise, traverse the right branch of node n.

Fig. 7 illustrates these two feature representations. The difference features have the disadvantage of requiring two distance calculations per feature. However, as we shall see in Section 5, these features have the potential to produce superior classification results, and redundancies in the distances used by each feature keep the average number of distances that must be calculated to classify a test character relatively small.

For decision tree feature sets, we have removed the effects of the number of strokes and aspect ratio from the distance measure by eliminating the stroke count difference from the measure, and by normalizing the size of each character to a standard height and width. The stroke count difference and original aspect ratio are then presented to the decision tree as two additional features along with the "local" features. In this way we allow the decision tree to determine the relative importance of each feature.

5. Experimental results

This section compares classification results based on the full training set, edited training set, and cluster centers. All timing results were obtained using a 296 MHz UltraSparc 1.

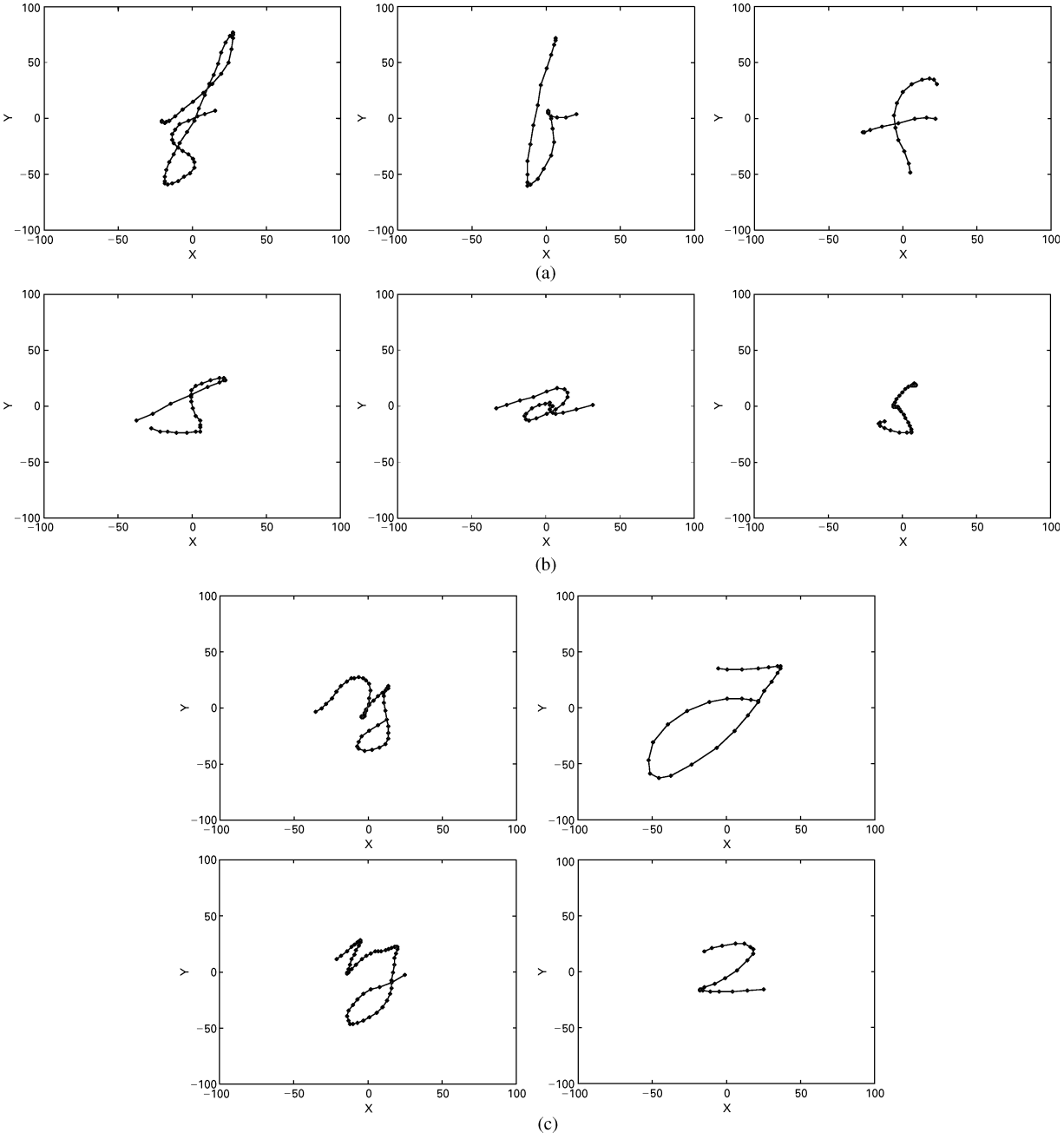


Fig. 6. Examples of the patterns closest to each cluster center found for the character classes (a) "*f*", (b) "*s*", and (c) "*z*".

5.1. Datasets

Experiments were run using a combination of three different sets of data: (i) a set of 600 handwritten digits captured from 21 different writers, which will be referred to as DA; (ii) a set of digits, referred to as DB, which is an independent set of 360 digits taken from an unknown number of writers and has been shown to have a slightly different writing style distribution than DA [28]; (iii) a set of 35,875 lowercase letters, originally written in a cursive style, but manually segmented so that they are now available as individual isolated characters. This set, referred to as *C*, was produced by the same group of 21 writers as DA.

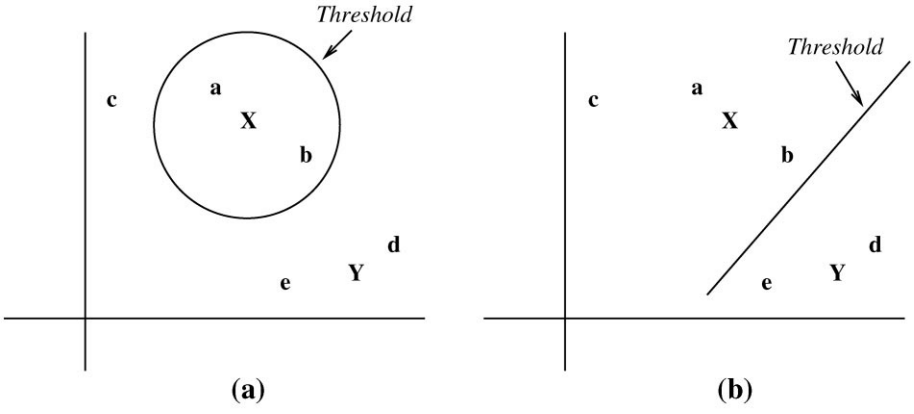


Fig. 7. Similarity and difference features. (a) Similarity features: while training patterns a and b are grouped based on their similarity, to template X, examples c and d are very different from each other but will be grouped together nonetheless; (b) Difference features: training patterns on each side of the decision threshold (difference between template X and template Y) are more similar to each other than to patterns on the other side.

Table 4
Number of clusters chosen for 10 digit classes by manual selection, and by automatic selection

Digit class	0	1	2	3	4	5	6	7	8	9
Manually selected K	4	5	7	6	3	3	4	4	5	10
Auto. selected K	4	3	3	6	3	4	7	3	4	7

5.2. Digit recognition

Digit classifiers were trained using the 600 digits in set DA, and tested using the 360 digits of set DB. Clustering was done for each of the 10 classes of the training set. Table 4 shows the number of clusters chosen for each of the digits using the following two methods: manually selecting K using the K vs. MSE graphs (e.g., Fig. 5), or automatic selection based on Davies and Bouldin's method [25]. For automatic selection, we consider only cluster counts greater than 2.

5.2.1. Nearest-neighbor classifier

Representative templates were chosen, one for each cluster, based on their close proximity to the center of the cluster. Table 5 shows the results of testing a nearest-neighbor classifier using four different training sets: the full set of 600 training examples, the set of 51 cluster centers where the number of clusters is manually chosen, the set of 44 clusters chosen by the Davies and Bouldin method, and the edited training set. As can be seen, both the cluster representation methods result in a significant reduction is classification time at the expense of lower classification accuracy. The edited set does not lead to substantial reduction in classification time, however surprisingly it results in a slightly better recognition accuracy.

Table 5
Digit recognition rates (10 classes)

Training set	No. templates	Throughput	Recog. rate (%)
Full set	600	~ 2 char/s	91.10
Manually selected K	51	~ 26 char/s	88.90
Auto. selected K	44	~ 31 char/s	87.50
Edited	402	~ 4 char/s	91.39

5.2.2. Decision trees

The construction of decision trees was accomplished using the package C5.0 [29]. The reference character set used in the definition of features (similarity or difference) to construct a decision tree can be any of the training sets used for the nearest-neighbor classifiers: the set of cluster centers, the edited training set, or the full training set.

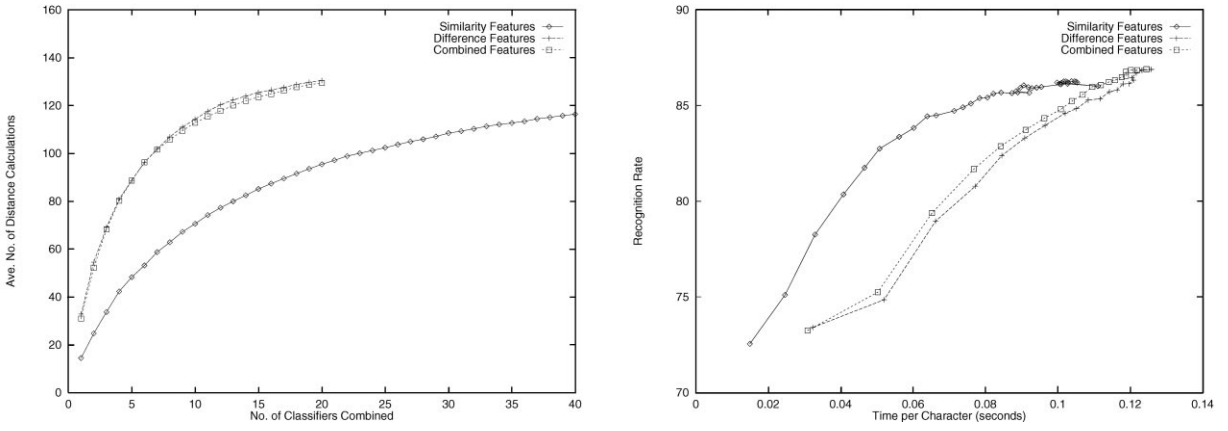


Fig. 8. Boosting for 36 alphanumeric classes. Comparing decision trees built using similarity features and difference features, both based on the cluster center reference set, and a combination of the two feature sets.

Table 6
Summary of best results for digit recognition using boosted decision trees

Reference set	Features	No. templates	Throughput	Recog. rate (%)
Cluster centers	Similarity features (18 comb. classifiers)	51 (31.4 ave.)	~ 34 char/s	88.6
Cluster centers	Difference features (9 comb. classifiers)	51 (31.6 ave.)	~ 27 char/s	88.1
Edited	Similarity features (19 comb. classifiers)	402 (65.7 ave.)	~ 17 char/s	87.2
Full set	Similarity features (11 comb. classifiers)	600 (48.4 ave.)	~ 22 char/s	86.4

Table 7
Number of clusters chosen for 36 alphanumeric classes

Class	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	g	h
K	4	3	3	6	3	4	7	3	4	7	3	3	3	3	3	3	3	3

Class	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
K	5	6	4	3	3	3	3	3	3	9	3	3	3	3	5	5	8	4

One of the advantages of the C5.0 package is that it can successively train multiple tree classifiers from the same data, with each new classifier focusing on improving the misclassifications of the previously trained classifiers, and then combine them using a technique called *boosting* [30]. As the number of classifiers used is increased, so is the average time required to classify each character, while the error rate approaches some lower bound (see Fig. 8 for an example). The best results are summarized in Table 6 for the similarity and difference features based on the cluster center reference set, and similarity features based on the edited reference set and full set of training examples. This table also reports the total number of templates available (the reference character set), and the average number of templates used in the classification of a single character.

It seems clear from Table 6 that the classifiers based on the cluster center reference sets perform better than the classifiers based on the edited and full sets. Overall, the similarity features based on the cluster center reference set gives the best results: 88.6% recognition accuracy with a throughput of approximately 34 characters/s.

This classifier compares well against our best nearest-neighbor classifier using cluster centers which achieves 88.9% recognition accuracy with a throughput of approximately 26 characters/s (Table 5).

5.3. Alphanumeric character recognition

Experiments were run using a set of 17,947 alphanumeric characters for training and 17,928 characters for testing, created by combining datasets DA and C, and splitting this combined set into testing and training sets by random selection. Clustering was performed on each of the 36 classes of this training set. The distribution of templates retained using cluster centers is shown in Table 7.

Table 8 shows the results of the nearest-neighbor classifier using the full training set, cluster centers where the number of clusters are automatically chosen, and the edited set. No significant increase in accuracy was found by using a K-nearest-neighbor classifier over a 1-NN classifier in either the digit or alphanumeric classification cases. Upon examination, this appears to be an indication that most character misclassifications occur because some of the characters are not well represented in the training set. As can be seen, the cluster centers method does not work well at all in this alphanumeric classification case. This exposes a problem in using cluster centers as training examples in a nearest-neighbor classifier: all the information about the size and shape of the cluster is lost which may result in decision boundaries not being properly represented. The edited set achieves a 33.7% reduction in classification time while maintaining most of the recognition accuracy. The overall computational requirement using this set, 12.18 s/ test character, cannot be considered practical, however.

Fig. 8 shows the results of classifying the data using boosted decision trees based on the 144 cluster centers and using similarity features, difference features, and a combination of the two. Due to the high dimensionality of the difference feature set, a reduced set of these features was obtained by training a single tree on the full set of difference features using a small amount of randomly sampled training data, and observing which features were selected for use by the training algorithm. This reduced the number of difference features from 20,736 (144×144) to 644. As Fig. 8 shows, the decision tree classifier does a better job of discriminating classes through the use of it is trained decision thresholds, compared to the simple nearest-neighbor classifier. Although the difference features tend to provide a higher classification accuracy for a given level of boosting, the similarity features give a better time vs. accuracy trade-off. The best classification accuracy reported is 86.9% with a throughput of $\sim$8 characters/s (see Table 9). Comparing this to the nearest-neighbor classifier using the cluster centers as a reference set, there is a 25.3% decrease in classification time, and compared to using the entire dataset as a reference set, there is a 99.3% decrease in classification time. In addition, the decision tree classifier retains 97.7% of the recognition accuracy achieved by the full dataset nearest-neighbor classifier.

Table 8
Alphanumeric recognition rates (36 Classes) using nearest-neighbor classifier

Training set	No. templates	Throughput	Recog. rate (%)
Full set	17,947	$\sim$ 0.05 char/s	88.92
Cluster centers	144	$\sim$ 6 char/s	45.41
Edited	11,867	$\sim$ 0.08 char/s	87.88

6. Conclusions

We have demonstrated a method of online character recognition which focuses on a representation of characters that models the different writing styles found in the training set. These models are then used by a decision tree classifier which yields good class discrimination. A method of automatically identifying writing styles and modeling them using templates has been proposed. Through this character representation and using a string matching distance measure, we are able to obtain an 86.9% classification accuracy for a 36-class set of alphanumeric characters with a throughput of over 8

Table 9
Standard nearest-neighbor vs. decision tree classifiers for 36-class alphanumeric character classification

Classifier	No. templates	Throughput	Recog. rate (%)
NN-full set	17,947	$\sim$ 0.05 char/s	88.92
NN-cluster centers	144	$\sim$ 6 char/s	45.41
Decision tree-similarity (23 boosted)	144 (100.1 ave.)	$\sim$ 11 char/s	86.1
Decision tree-difference (20 boosted)	144 (130.5 ave.)	$\sim$ 8 char/s	86.9
Decision tree-combined (18 boosted)	144 (127.7 ave.)	$\sim$ 8 char/s	86.9

characters/s on a 296 MHz Sun UltraSparc. In addition, improvements in accuracy should be attainable through the use of a rejection threshold.

In future work, we will study the benefits of writing style-based representations, when characters are represented using hidden Markov models. The optimal space for clustering should then be one in which the distance from a character to a cluster is the probability that the character was generated by a model for that cluster. A pairwise comparison of characters is difficult in this space, which means that an initial clustering must contain only clusters with some minimal membership such that model parameters can be estimated for each. Further, large intracluster variations may cause the clustering to be very susceptible to local minima, and therefore a good starting point may be crucial to the success of such a clustering method. We believe the string matching technique presented here will provide a good initial clustering.

Acknowledgements

The authors would like to greatfully acknowledge Krishna Nathan, Jayashree Subrahmonia, and the Consumer Devices group at IBM T. J. Watson Research Center for providing data, feedback, and financial support for this research.

References

[1] E. Mandler, R. Oed, W. Doster, Experiments in on-line script recognition, Proceedings of fourth Scandinavian Conference on Image Analysis, June 1985, pp. 75–86.

[2] E.J. Bellegarda, J.R. Bellegarda, D. Nahamoo, K.S. Nathan, A probabilistic framework for on-line handwriting recognition, Proceedings of IWFHR III, Buffalo, New York, May 25–27, 1993, pp. 225–234.

[3] S.A. Guberman, I. Lossev, A.V. Pashintsev, Method and apparatus for recognizing cursive writing from sequential input information, US Patent WO 94/07214, March 1994.

[4] S. Bercu, G. Lorette, On-line handwritten word recognition: an approach based on hidden Markov models, Proceedings Third International Workshop on Frontiers in Handwriting Recognition, Buffalo, USA, May 1993, pp. 385–390.

[5] J.M. Hollerback, An oscillation theory of handwriting, Biol. Cybernet. 39 (1981) 139–156.

[6] R. Plamondon, F.J. Maarse, An evaluation of motor models of handwriting, IEEE Trans. Systems Man Cybernet. 19 (5) (1989) 1060–1072.

[7] L.R.B. Schomaker, H.-L. Teulings, A handwriting recognition system based on the properties and architectures of the human motor system, Proceedings of the IWFHR, CENPARMI Concordia, Montreal, 1990, pp. 195–211.

[8] X. Li, D.-Y. Yeung, On-line handwritten alphanumeric character recognition using dominant points in strokes, Pattern Recognition 30 (1) (1997) 31–44.

[9] P. Scattolin, A. Krzyzak, Weighted elastic matching method for recognition of handwritten numerals, Proceedings of Vision Interface '94, pp. 178–185.

[10] C.C. Tappert, Adaptive on-line handwriting recognition, Proceedings of Seventh International Conference on Pattern Recognition, July–August 1984, pp. 1004–1007.

[11] R.P.W. Duin, D. de Ridder, D.M.J. Tax, Experiments with object based discriminant functions; a featureless approach to pattern recognition, Pattern Recognition Lett. 18 (11–13) (1997) 1159–1166.

[12] A.K. Jain, D. Zongker, Representation and recognition of handwritten digits using deformable templates, IEEE Trans. Pattern Anal. Mach. Intell. 19 (12) (1997) 1386–1391.

[13] T. Starner, J. Makhoul, R. Schwartz, G. Chou, On-line cursive handwriting recognition using speech recognition methods, Proceedings of ICASSP'94, Vol. 5, 1994, pp. 125–128.

[14] S. Manke, U. Bodenhausen, A connectionist recognizer for on-line cursive handwriting recognition, Proceedings of ICASSP'94, Vol. 2, 1994, pp. 633–636.

[15] M. Schenkel, I. Guyon, D. Henderson, On-line cursive script recognition using time delay neural networks and hidden Markov models, Proceedings of ICASSP'94, Vol. 2, 1994, pp. 637–640.

[16] L.S. Yaeger, B.J. Webb, R.F. Lyon, Combining neural networks and context-driven search for online, printed handwriting recognition in the Newton, AI Mag. (1998) 73–89.

[17] K.-F. Chan, D.-Y. Yeung, Elastic structural matching for on-line handwritten alphanumeric character recognition, Proceedings of 14th International Conference on Pattern Recognition, Brisbane, Australia, August 1998, pp. 1508–1511.

[18] I. Guyon, L. Schomaker, R. Plamondon, M. Liberman, S. Janet, UNIPEN project of on-line data exchange and recognizer benchmarks, Proceedings of 12th International Conference on Pattern Recognition, Jerusalem, Israel, October 1994, pp. 29–33.

[19] X. Li, R. Plamondon, M. Parizeau, Model-based on-line handwritten digit recognition, Proceedings of 14th International Conference on Pattern Recognition, Brisbane, Australia, August 1998, pp. 1134–1136.

[20] L. Prevost, M. Milgram, Automatic allograph selection and multiple classification for totally unconstrained handwritten character recognition, Proceedings of 14th International Conference on Pattern Recognition, Brisbane, Australia, August 1998, pp. 381–383.

[21] J. Hu, A.S. Rosenthal, M.K. Brown, Combining high-level features with sequential local features for on-line handwriting recognition, Proceedings of Italian Image Process. Conference, Florence, Italy, September 1997, pp. 647–654.

[22] H. Beigi, Pre-processing the dynamics of on-line handwriting data, feature extraction and recognition, Proceedings of the International Workshop on Frontiers of Handwriting Recognition, Colchester, England, September 2–5, 1996, pp. 255–258.

[23] A.K. Jain, L. Hong, S. Pankanti, R. Bolle, An identity-authentication system using fingerprints, Proc. IEEE 85 (9) (1997) 1365–1388.

[24] A.K. Jain, R.C. Dubes, Algorithms for Clustering Data, Prentice-Hall, Englewood Cliffs, NJ, 1988.
[25] D.L. Davies, D.W. Bouldin, A cluster separation measure, IEEE Trans. Pattern Anal. Mach. Intell. 1 (2) (1979) 224–227.
[26] R.O. Duda, P.E. Hart, Pattern Classification and Scene Analysis, Wiley-Interscience, New York, 1973.
[27] L. Breiman, J. Friedman, R. Olshen, C. Stone, Classification and Regression Trees, Wadsworth, Belmont, CA, 1984.
[28] S. Connell, A.K. Jain, Learning prototypes for on-line handwritten digits, Proceedings of 14th International Conference on Pattern Recognition, Brisbane, Australia, August 1998, pp. 182–184.
[29] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kauffman, Los Altos, CA, 1993.
[30] J.R. Quinlan, Bagging, boosting, and C4.5, Proceedings Thirteenth National Conference on Artificial Intelligence, Portland, OR, August 1996, pp. 725–730.