ONLINE HANDWRITING RECOGNITION USING MULTIPLE PATTERN CLASS MODELS

By

Scott D. Connell

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science and Engineering

2000

Abstract

ONLINE HANDWRITING RECOGNITION USING MULTIPLE PATTERN CLASS MODELS

By

Scott D. Connell

The field of personal computing has begun to make a transition from the desktop to handheld devices, thereby requiring input paradigms that are more suited for single hand entry than a keyboard and recent developments in online handwriting recognition allow for such input modalities. Data entry using a pen forms a natural, convenient interface. The large number of writing styles and the variability between them makes the problem of writer-independent unconstrained handwriting recognition a very challenging pattern recognition problem. The state-of-the-art in online handwriting recognition is such that it has found practical success in very constrained problems. In this thesis, a method of identifying different writing styles, referred to as lexemes, is described. Approaches for constructing both non-parametric and parametric classifiers are described that take advantage of the identified lexemes to form a more compact representation of the data, while maintaining good recognition accuracies. Experimental results are presented on different sets of unconstrained online

handwritten characters and words. In addition, a method of combining information from lexeme models built on different feature sets is described, and results are presented on both English characters and Devanagari characters. Finally, a method of writer-adaptation is described which makes use of the lexemes identified from a large group of writers to define lexemes within a small amount of data from a single writer.

Acknowledgments

The author would like to sincerely thank the following people for providing assistance, support, encouragement, and inspiration during the writing of this dissertation. First, I'd like to thank my advisor Dr. Anil Jain. He has provided guidance, encouragement, opportunities, and knowledge at a level that few advisors are capable of. His dedication to his students and his field is a true inspiration. Second, I'd like to thank the other members of my committee: George Stockman, Eric Torng, Habib Salehi, and Jayashree Subrahmonia. I'd also like to thank Dr. Subrahmonia, along with the other members of the IBM Pen Technologies group at Watson Research Center, Michael Perrone, Gene Ratzlaff, and John Pirelli, for their support and assistance. They have provided data, financial support, internships, and advice that have been extremely helpful in the completion of this dissertation. In addition, I'd like to thank Dr. R.M.K. Sinha for assisting me with his expertise in Devanagari character recognition.

No student can survive graduate school without the help of their fellow students to discuss ideas, share opinions, and to make time spent in the lab an all around enjoyable experience. In particular, I'd like to thank Aditya Vailaya, Salil Prabhakar, Arun Ross, Nicolae Duta, Lin Hong, Yonghong Li, Paul Albee, Friederike Griess, and Anoop Namboodiri, and the other members of the Pattern Recognition and Image Processing lab. In addition, I'd like to thank the Computer Science Department staff for their valuable assistance, in particular Linda Moore and Cathy Davison.

I'd also like to give special thanks to my parents, sister, and the rest of my family. They have always been there to support me, and I am very grateful. Finally, I'd like to thank all the friends I have made during my time at Michigan State University.

Table of Contents

List of Tables

List of Figures

Chapter 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 between a computer and a user is becoming a serious bottleneck. In order to be effective, user interfaces should be i) efficient and ii) natural to the user, thereby requiring little or no learning curve for the user. While there has been much progress on how data is presented to a user, such as data visualization tools [2], primary mode of data input from a human to a computer is still the keyboard. Speech recognition and handwriting recognition enable other, more natural, forms of man-machine 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 (where this is largely domain dependent) such that the user needs to make only a minimal number of corrections to the recognized text or speech.

While much of today's data is entered directly into computers using the keyboard, many tasks still exist in which people tend to prefer handwriting over keyboard entry. Note taking (e.g., in a classroom) is a task that can still be done more efficiently by hand for most users. In addition, while people can produce annotated hand sketches very quickly, data entry into a computer using a combination of the mouse and keyboard is relatively time consuming. Personal Digital Assistants (PDAs) are pocket sized consumer devices that can store calendars and address books, provide access to email, and contain other productivity tools. These devices are too small to have full sized keyboards, or sometimes may be too small for any keyboard at all, requiring pen or voice interfaces to enter data. Finally, some natural languages contain a very large number of symbols (e.g., Kanji contains 4,000 commonly used characters) making keyboard entry even a more difficult task. For these languages, handwriting recognition has the potential to provide a much more efficient data entry method.

The problem of handwriting recognition has now been a topic of research for over three decades. Only in recent years has the technology progressed to the point where consumer products based on handwriting recognition have become affordable and commercially available. There are many types of problems (with varying complexity) within handwriting recognition, based on how the data is presented to the recognition system, at what level the data can be unambiguously broken into pieces (e.g., individual characters or words), and who will use the recognizer. A typical handwriting recognition system focuses on only a subset of these problems.

Table 1.1: Some applications of handwriting recognition.

Application	Requirements	State-of-the-Art Accuracy
Interface for Personal Digital Assistant (PDA)	Boxed character recognition; real-time speed.	$> 99\%$ on unnatural character set (e.g., graffiti)
Handwritten Notes (online) Recognition	Segmentation into individual words; large vocabulary of words categories; can be performed in batch mode, therefore, need not be real time.	approx. $85\% - 95\%$ word accuracy if trained on user (writer-dependent) given an adequate amount of data
Postal Address Recognition	High-volume; must be real-time; requires an extremely low error rate, therefore, reject rate is often high.	$> 99\%$ on ZIP code, city and state with approx. $25\% - 35\%$ reject
Online Signature Verification	Must have a low false rejection rate, while maintaining a low false accept rate.	$2\% - 5\%$ Equal error rate

1.1 Online vs. Offline Handwriting Recognition

At the highest level, handwriting recognition can be broken into two categories: *offline* and *online*. Offline handwriting recognition focuses on documents that have been written on paper at some previous point in time. Information is presented to the system in the form of a scanned image of the paper document. In contrast, online handwriting recognition (by its original definition) focuses on tasks where recognition needs to be performed at the time of writing. This requires the use of special equipment, such as a digitizing tablet, to capture the strokes of the pen as they are being written. 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. While this pen trace could be used

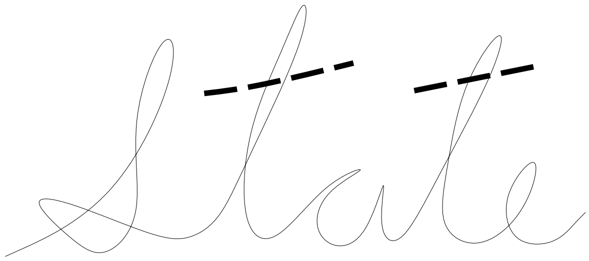


Figure 1.1: An example of delayed strokes (shown as dashed lines). In cursive handwriting, delayed strokes are particularly troublesome because it is unclear to which character they belong.

to construct a static image of the writing, thus allowing offline character recognition techniques to be applied, it has been shown [63] that the information about the pen dynamics can be used to obtain better recognition accuracies than static data alone. Therefore, it is beneficial to capture the data in an online form, even if the real-time processing requirements can be relaxed.

Another advantage of online handwritten data over offline data is the availability of stroke segmentation and order of writing. Ink in static images must first be separated from the image background, creating a potential source of error. Pen devices used in online recognition include the ability to detect the states of *pen-down* (when the pen is touching the tablet) and *pen-up* (when the pen is not touching the tablet). A single stroke is defined as the sequence of sample points occurring between consecutive pen-down and pen-up transitions. However, a complication occurs when a stroke is added to a character in a word after the rest of the word has already been written, such as the cross of a 't' or an 'x', or the dot of an 'i' or a 'j'. These types of strokes are called *delayed strokes* (see Figure 1.1).

1.2 Handwriting Segmentation

A necessary step of handwriting recognition is the process of segmenting the input in such a way that we can present isolated words or characters to the recognition system. With offline data, segmentation must be addressed on a pixel by pixel basis, first separating writing from background, then separating words and characters. The process of segmentation is relatively simpler for online handwriting recognition since there is no "signal" when the user is not writing. Further, we have the additional dimension of time in which characters or words can be separated. Even with this additional information, the problem of reliably identifying online stroke "clusters" that form individual characters or words can be difficult. Figure 1.2 shows the five different categories of character separation. In particular, *cursive handwriting* is a style of writing in which a number of consecutive characters in a word may be written using a single stroke (last two rows in Figure 1.2), thus making the problem of character segmentation very difficult. The more difficult the character segmentation problem, the greater the opportunity for misclassifications within a word, and the more a recognition system must rely on higher level information about the words (and language) that can be formed to restrict the segmentation hypotheses.

The most difficult level of segmentation is when cursive and handprinted characters are mixed within the same word. In this case we cannot count on the beginning and endings of strokes to define possible character segmentations, nor do they always provide reliable information for word segmentation. When the type of character and word separation is not known a priori, such writing is referred to as *unconstrained.*

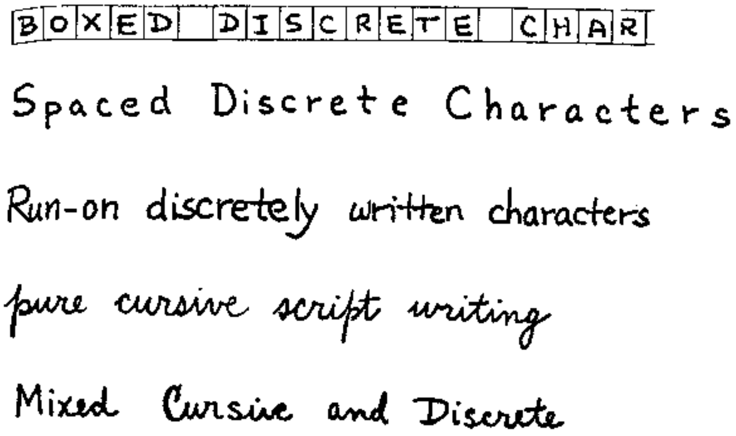


Figure 1.2: Levels of difficulties in handwriting segmentation [103].

When we impose the restriction that the writer must write each character in a separate box, character segmentation becomes trivial and the problem of recognition is reduced to the much simpler character recognition problem. This has motivated some countries to require postal zip code digits to be written in boxes to increase the effectiveness of automatic mail sorting. It should be noted that for online handwritten data, an equivalent form of this level of character separation is when the characters are separated by some predefined minimum amount of time, forming a temporal box as opposed to a spatial box.

An even more restrictive form of handwriting exists in which the writer is not only expected to make character segmentation explicit, but the motion of the pen to form each character is highly confined such that class discrimination is very simple. In fact, the formation of some characters may have little similarity to their corresponding Roman alphabet characters. Examples of such alphabets are Graffiti [75] and Jot [51]. Currently, many commercial systems exist (Palm Pilot, Apple Newton

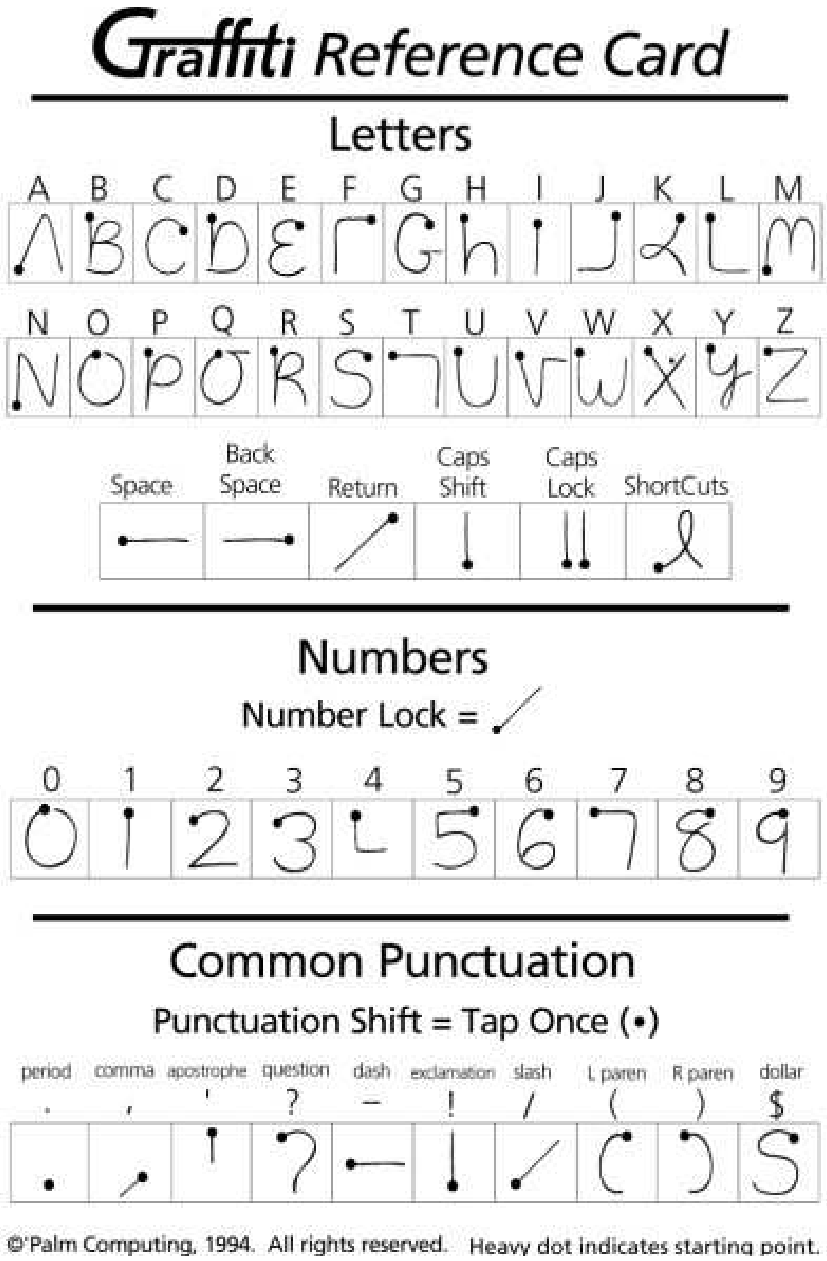


Figure 1.3: The Graffiti character set. Reproduced from [32].

Messagepad, Windows CE machines, etc.) which make use of these alphabets. The recognition accuracies reported using such alphabets tend to be very good (> 99%), however they are not practical for applications in which writing speed is an issue (e.g., note-taking). Further, these alphabets tend to have a somewhat large learning curve. Other commercial systems make use of more natural alphabets, but they restrict writing to handprinted characters and single isolated words (smARTwriter by Advanced Recognition Technologies, Inc. [1]). Still other products exist (CrossPad, Figure 1.4, by IBM and AT Cross, Inc., and Calligrapher by Paragraph, Intl. [76]) which attempt to recognize unconstrained handwriting. However, these systems do so at the cost of a lower recognition accuracy, and often include the option of more restricted modes of writing (such as cursive-only, handprint-only, and digit-only modes) for increased recognition accuracy. Figure 1.5 shows some examples of unconstrained handwritten sentences written on the CrossPad, and their recognition results using the CrossPad's writer-independent mode of recognition.

1.3 Writer-dependent vs. Writer-independent Recognition

A handwriting recognition system can further be broken down into the categories of *writer-dependent*, or *writer-independent*. A writer-independent system is trained to recognize handwriting in a wide variety of writing styles, while a writer-dependent system is trained to recognize handwriting of a single individual. A writer-dependent

Figure 1.4: The CrossPad.

Written Sentence	Recognition Results
All work and no play makes Jack a dull boy.	All work and no play play makes Jack a dull **try**.
One small step for a man, one great leap for mankind.	One **Sin all** step for a **in an**, one **9-9 it** leap for man kind.

Figure 1.5: Recognition results using the CrossPad in writer-independent mode. The left column contains examples of handwritten sentences, while the right column contains the recognition results for those sentences. Recognition errors are marked in bold.

system works on data with a smaller variability, and therefore achieves a higher recognition rate. A writer-independent system sacrifices some of this recognition accuracy that could be obtained for a single user, for the ability to handle a much greater variety of writing styles. On the other hand, when training a writer-independent system, it is easy to obtain a large amount of training data, since many writers are included in the data collection process, while expecting a single writer to provide a similar amount of data to train a writer-dependent system is not feasible.

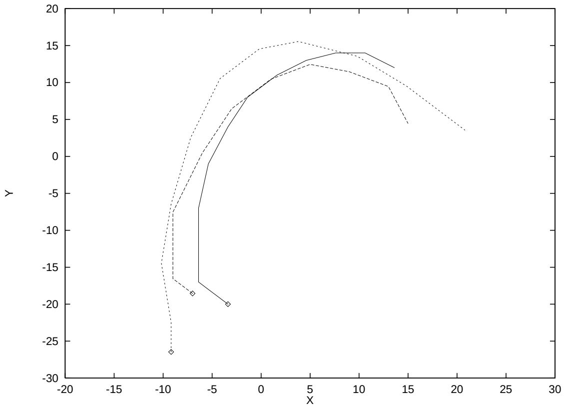
At first, it would seem that the required data collection for writer dependent training should not be difficult if we start with a working writer-independent system and adapt it to the writer-dependent problem (known as *writer-adaptation*). Since the user will be constantly providing new data (writing) to the system, as he or she uses it, why can't this data be used for incremental learning? The difficulty arises in that the "ground truth" (true categories) of this data are unreliable since they have been provided by the writer-independent system that we wish to improve upon in the first place. Therefore, the data must undergo a verification process in which the user looks over the data and category labels, and corrects them. This is especially challenging for cursive handwriting in which the segmentation points between characters chosen by the writer-independent system might not be reliable as well. The CrossPad's recognition engine allows for writer-adaptation by requesting that the writer provide a set of predefined characters, after which the user must go through a verification process to ensure that the predefined labels are correctly matched up with the handwritten characters before training begins. The amount of data that can be collected in this fashion is limited by the patience of the writer.

The smARTwriter system [1] adapts by collecting those character examples that the user has explicitly corrected during normal use, and therefore can be assumed to be correctly labeled. While this method does not require the writer to spend his time explicitly in providing training data, it further limits the amount of data that can be collected in a short period of time.

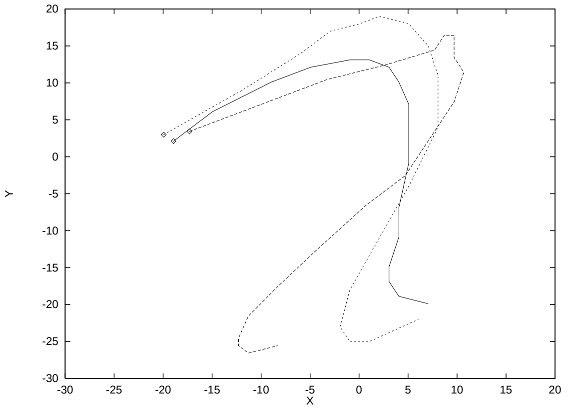
1.4 Pattern Class Models

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 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 Figure 1.6.

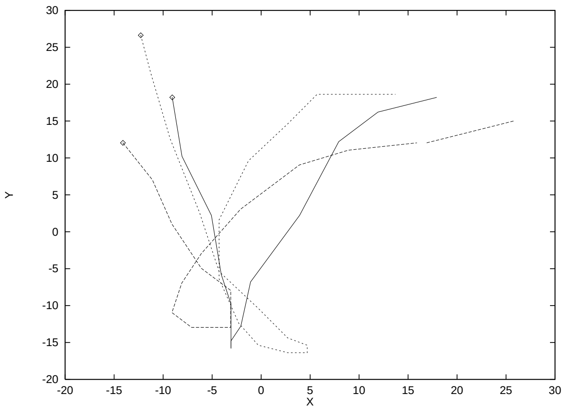
We refer to these different ways of writing a character as lexemes [96], patterned after the corresponding speech recognition term, phoneme. Other work has appeared in the literature in which these character subclasses are referred to as allographs [82, 106, 115]. If these lexemes can be identified then it is often possible to increase the recognition accuracy by modeling each lexeme separately, creating multiple models per character class. An automated method of reliably determining the lexemes that are appropriate for a certain data set is considered a difficult problem. Currently, the role of lexemes in writer-adaptation has remained largely unexplored.



(a)



(b)



(c)

Figure 1.6: 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 (writer) represented in each box. The beginning of each stroke is shown by a dot.

Table 1.2: Recent results on presegmented character recognition.

Author	Method	Accuracy	Comments
Li & Yeung [60]	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 sec/char. 180 reference templates selected from 840 examples.
Scattolin & Krzyzak [91]	Nearest Neighbor using Weighted Elastic Matching	88.67% on 1650 digits	33 writers used. 293 reference templates.
Yaeger et al. [114]	Combined Online and Offline Neural Network Classifier	95-class classifier (52 chars, 10 digits, 33 symbols): 86.1%	45 writers used.
Chan & Yeung [18]	Manually Designed Structural Models	97.4% on 9300 upper and lowercase chars.	150 writers used.
Li et al. [59]	Hidden Markov Model	92% (after 3.9% reject) on 3126 digits.	Unipen data used.
Prevost & Milgram [82]	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.

1.5 State-of-the-Art in Handwriting Recognition

Table 1.2 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, which makes a comparison of these results difficult. Recently, results have been reported using data from the Unipen dataset [37]. 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 was not publicly available.

Table 1.3 presents some recent results for the problem of handwritten word recognition. Again, the lack of common datasets makes it difficult to compare these results. In many of these approaches, the character models are chained together to form words which makes the recognition of words from a very large vocabulary (over 20,000 words) possible without requiring explicit examples of all these words for training. However, it should be noted that the results reported are on test sets containing words from only a fraction of the total word lexicon that can be recognized by these handwriting recognition systems. Generally, the best accuracies reported in this table are for writer-dependent systems. One writer-independent system that achieves a very good accuracy is that of Manke et al [64], which obtains 91.4% accuracy for a 20,000 word vocabulary for 40 writers. It is not clear, however, if the training data is obtained from an independent set of writers. Hu et al. [44] achieve a writer-independent accuracy of 94.5% on 3,823 unconstrained word examples, but this is achieved by limiting the lexicon size to only 32 words. In another study, Hu et al. [43] achieved an error rate of 9.0% for a lexicon of 1905 lowercase words. Examples of how difficult the unconstrained word recognition task can become can be seen in the 64.4% [95] and 81.1% [72] accuracies achieved by the other recent writer-independent systems.

1.6 Research Problem Statement

While much success has been obtained on simplified handwriting recognition problems, such as isolated word recognition, the goal of fully unconstrained, large vocabu-

Table 1.3: Recent results on handwritten word recognition.

Author	Method	Accuracy	Comments
Nathan, et al. [72]	HMM	81.1% on unconstrained word examples.	25 writers. Writer-independent. 21K word vocab.
Manke, Finke, and Waibel [64]	Multi-state TDNN	91.4% on 2,500 examples	Writer-independent. Lowercase words. 40 writers. 20K word vocab.
Rigoll, et al. [56, 89]	HMM/NN hybrid	95% on 200 examples	Writer-dependent. 3 writers. 30K word vocab.
Hu, Rosenthal, and Brown [43]	HMM	91% on 2,000 examples	16 writers. Writer-independent. Lowercase words. 1,905 word vocab.
Seni et al. [95]	TDNN	62.4% for writer-independent, 91.6% for writer-dependent.	9 writers. 466 cursive word examples. 21K word vocab.
Hu, Brown, and Turin [44]	Stroke-level HMM	94.5% on 3,823 unconstrained word examples	Writer-independent. 18 writers. 32 word lexicon

lary handwriting recognition still remains a challenge due to the amount of variations found in characters as they appear within words. The goal of this thesis is to define a methodology to model online handwritten character data through the identification and representation of disparate writing styles (lexemes), and thereby better model handwritten words. Due to the temporal nature of online data, this work has possible application to the domain of speech recognition as well. The benefits of this data segmentation to both non-parametric, and parametric classifiers is explored. In the latter, we present a system for word-level recognition, which makes use of character models and a grammar to define word constructs, and investigate at what level of

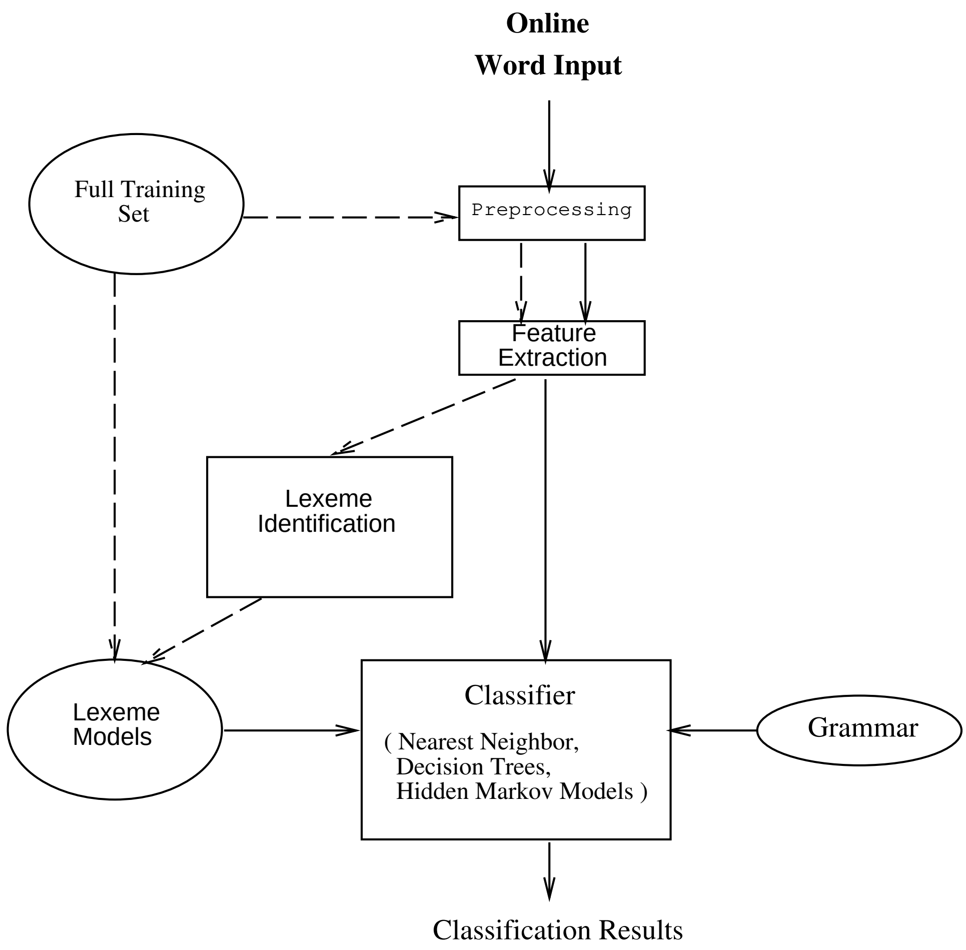


Figure 1.7: Online word recognition system. The flow of data during training is shown by the dashed line arrows, while the data flow during recognition is shown by the solid line arrows.

the training process lexemes are useful. A block diagram of this system is presented in Figure 1.7. In addition, we explore the identification and usefulness of lexemes in writer-independent and writer-dependent recognition, as well as writer adaptation from the former to the latter. Modeling is done at the character level, and word recognition is accomplished by chaining together character models. Therefore, results will be first presented at the character level, and later at the word level.

The scope of this thesis consists of the following problems:

1. Automatic identification of character writing styles from a pool of multiple-writer data, in which the number of styles present in the data is not known a priori. We also explore the dependence of the identification method on the type of classifier that will make use of these lexemes.

2. The use of lexemes in optimizing non-parametric classifiers in which we wish to reduce classification time while maintaining our recognition accuracy.

3. The use of lexemes in optimizing the recognition accuracy of parametric classifiers.

4. Adaptation of a writer-independent system to a writer-dependent system based on a lexeme labeling and subselection method. This has the potential to improve the system's accuracy on an individual writer while requiring only a minimal sampling of the writer's handwriting as training data.

5. The identification of lexemes within a set of data as driven by the parametric models that will represent these lexemes.

6. The application of our handwriting recognition methodologies to the domain of Devanagari script recognition.

7. The fusion of different information sources to improve classification accuracies.

1.7 Thesis Outline

The organization of the thesis is as follows. Chapter 2 presents a literature review of online character and word recognition, including work related to subclass models and identifying writing styles. Our method of identifying lexemes which makes use of a string matching measure and K-Means clustering algorithm, is presented in Chapter 3. Methods of non-parametric character modeling, in which prototypical examples are identified for each lexeme and used to reduce classification times are presented in Chapter 4. Chapter 5 demonstrates how lexeme models can be used to increase recognition accuracies of a parametric classifier, and how the representation of these models can be used to drive the identification of the lexemes. In Chapter 6, the topic of information fusion is discussed as a means of combining classifiers to achieve better handwriting recognition accuracies. Our unconstrained word recognition system is presented in Chapter 7. In Chapter 8 we show the ability of our handwriting recognition system to generalize to other languages, in particular Devanagari script. Issues involved in creating a writer-dependent handwriting recognition system, in which lexemes are involved, and how these lexemes can be used to leverage knowledge from a writer-independent system to perform writer-adaptation are discussed in Chapter 9. Finally, some conclusions and future work are presented in Chapter 10.

Chapter 2

Literature Review

2.1 Introduction

This chapter describes some of the issues and techniques involved in online handwriting recognition. For a more complete survey of techniques applied to the recognition of this type of handwritten data, a number of survey papers exist [73, 79, 81, 105, 112]. Online handwriting recognition can be broken into the following steps: data collection, preprocessing, segmentation, feature extraction, modeling/recognition, and postprocessing. Section 2.2 describes data collection devices and characteristics. Common preprocessing techniques are presented in Section 2.3, and segmentation methods, at both the word and character level, are discussed in Section 2.4. Modeling techniques used for handwriting recognition, and their corresponding feature sets, are described in Section 2.5, including a discussion of defining multiple models per class. Finally, postprocessing techniques which make use of higher-level knowledge about words are presented in Section 2.6.

2.2 Data Collection

Online handwritten data must be collected using a special device. Typically, a digitizing tablet is used that samples the location of a stylus on the tablet at the rate of approximately 80 - 200 times per second. This generates a sequence of (x, y) coordinates which define the trace of the pen over time. The stylus will typically have a switch to detect pen-down (when the pen is touching the tablet) and pen-up (the pen is not touching the tablet) status. Some devices make use of a pressure sensor on the tip of the pen and collect pressure information in addition to the location information. Other devices that can be used to collect data include touch screens, such as found on many PDAs, and the CrossPad. The CrossPad is a device that receives an RF signal from a special pen. A receiving grid inside the pad allows the CrossPad to detect the location of the pen. The advantage of such a device is that the writing surface is not part of the sensor, and is actually the paper (e.g., a legal pad) making the writing experience more natural.

2.3 Preprocessing

The goal of preprocessing is to reduce or eliminate some of the variations in handwriting that may exist that are not useful for pattern class discrimination. For a good survey of preprocessing techniques, see [35]. 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 using a spline filter [43], or a Yulewalk

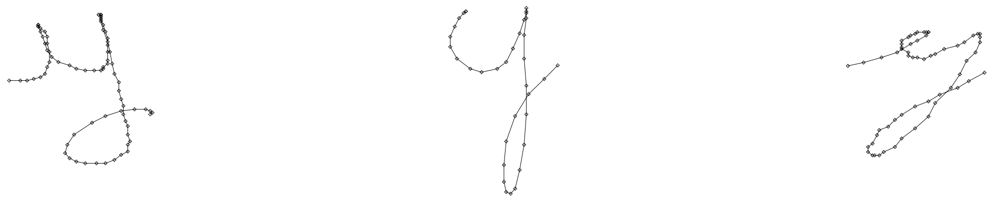


Figure 2.1: Differences in sample point spacings for handwritten characters. Each dot indicates the location of a sample point.

filter for low-pass filtering of the data [6]. Often the beginning or ending of strokes may have small hooks that are created by a quick motion that occurs when the pen is lowered or raised. *Dehooking* is the process of detecting and removing these hooks.

Most preprocessing techniques resample 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 (see Figure 2.1).

The fewer the restrictions placed on a writer, the larger is the intra-class variations. In general, the $x-y$ space that handwritten characters occupy can be divided into 3 regions (see Figure 2.2):

1. A middle region from the *baseline* of writing (the line which supports the writing) to the *midline* (line that sits on top of writing which all ascenders extend above).

2. An ascender region that extends from the midline to the top of the ascenders.

3. A descender region that extends from the baseline to the bottom of the

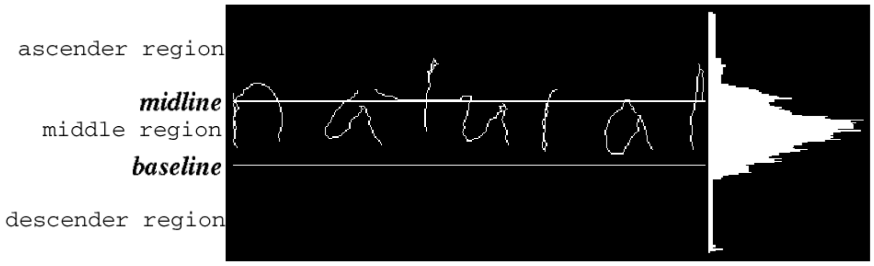


Figure 2.2: The three regions used for size normalization.

descenders.

If a writer was required to write within a set of horizontally running lines which mark these regions, this would reduce the amount of variation in character size. In addition, it would ensure that the baseline is not skewed upward or downward as the writing proceeds from left to right. However, without putting such restrictions on the writer, variations in size and skew needs to be reduced by first detecting these regions. A large peak in a horizontal projection profile of the writing can be used to indicate the middle region of writing [7]. The height of the writing can then be adjusted such that the middle region has a prespecified standard height. The degree of baseline skew can be detected by creating projection profiles at a number of orientations off the horizontal direction, and search for the profile in which the average slope of the peak is the greatest [84]. Other methods makes use of the Hough transform [38].

Another form of variation in handwriting is the amount of slant in the characters (see Figure 2.3). While the variance of the slant may be very small for a single writer, it can be rather large for a writer-independent system. Slant can be detected by estimating the average slope of up and down strokes in the writing [14].

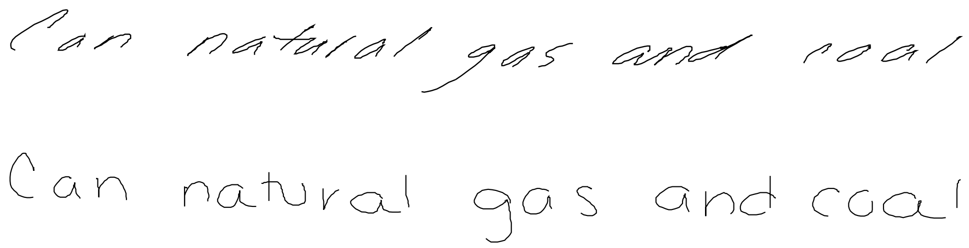


Figure 2.3: The amount of slant in handwriting for two different writers.

2.4 Segmentation

Depending on the form in which data is presented to a recognition system, some segmentation of the input may be required. Data which is presented in the form of a single word will require character segmentation if writing is modeled at the character level, while data which is presented as multiple words first requires a method of segmenting the data into individual words. The input may be a line of text, in which the segmentation points can be defined along the x-axis, or it may be an entire page, requiring some type of page decomposition technique as is done in document analysis [69, 74, 113]. Most of the handwriting recognition systems focus on either individually segmented words or individual characters.

While it may be possible to train full-word models for a small vocabulary of words, this is not feasible for large vocabularies (20,000 words or more) due to the amount of data that must be collected to train each model. Since each word is constructed from a subset of the character alphabet, it is much more efficient to classify words using character models. The degree of difficulty in segmenting a word into characters is dependent on the amount of separation that can be expected between the characters (see Figure 1.2). For well-separated characters, simple spatial gap thresholds can be used to segment characters. For overlapping or connected characters, a more sophisticated method is required. One method begins by an oversegmentation in which candidate segmentation points are selected [11]. A graph of possible segmentation sequences is constructed by connecting consecutive segments to form sequences of potential whole characters. Each of these combined segments is then classified into a character class by the recognizer and the "best" character sequence is identified during post-processing. A further extension of this approach is the method of segmentation during recognition as is commonly used with hidden Markov models. In this method all sample points along the handwritten word are potential segmentation points. Character models are chained together in such a way that all possible words can be formed by some path through the graph. The path that generates the best recognition score also implicitly defines the sequence of characters in the word. This method of combining character models to form word models using a grammar will be described further in Section 2.6, and then demonstrated in Chapter 7.

2.5 Modeling Methods

A number of recognition systems have been built using (i) primitive decomposition [10, 34], (ii) motor models [80, 93], (iii) deformation models such as local and global affine transformations [111] and elastic matching [18, 21, 103], (iv) hidden Markov models [8, 10, 44, 71, 89], and (v) neural network architectures such as time-delay neural networks [36, 65, 92, 95]. The advantages and disadvantages of each of these

Table 2.1: Approaches to online handwriting recognition.

Technique	Advantages	Disadvantages
Primitive Decomposition [10, 34]	Powerful high-level features.	Not very robust to large variations in writing style.
Motor Models [41, 80, 93]	Takes advantage of pen dynamics.	May lack robustness when writing style variations are large.
Deformation Models [60, 77, 91, 103, 111]	Works very well for 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.
Hidden Markov Models [8, 101]	Models temporal relationships well.	Requires a large amount of training data.
Neural Networks [65, 92, 114]	Classification time is fast.	Does not model temporal relationships very well.

techniques are summarized in Table 2.1.

2.5.1 Primitive Decomposition

Primitive decomposition [10, 15, 34, 61] 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 requires a pre-segmentation of the strokes into sub-stroke pieces. Guberman et al. [34] perform word recognition by segmenting the sequence of strokes forming the word into sub-strokes referred to as *metastrokes* (see Figure 2.4). This sequence of metastrokes is compared to metastroke sequences in a dictionary of words, and a scoring mechanism is used to determine the word identity. Bercu and Lorette [10] have applied a hidden Markov model to the recognition of words represented as a sequence of primitives. Chan et al. [15] have identified primitives as line segments belonging to one of the following categories: line, counter-clockwise curve, clockwise curve, loop, or dot. The sequence of primitives is matched to a template using elastic matching.

2.5.2 Motor Models

Motor Models [41, 80, 93] 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.

2.5.3 Deformation Models

Deformation models make use of a set of templates and a method of deforming these templates to represent the data. The amount of deformation required to match a template to a test example indicates the degree of match. Deformation model techniques have also been used in offline digit recognition [50]. Elastic Matching [60, 91, 103] works on the sequence of sample points directly by searching for an alignment of data points between an unknown character, and the stored templates. The distance between an unknown character and a template is then taken as the sum of their distances between aligned points. The most similar template (that is the shortest distance from the unknown character) defines its identity. Tappert [103] uses the Euclidean distance between the (x, y) coordinates of two aligned points as their interpoint distances, and computes the best alignment using dynamic programming.

CODE	SYMBOL	DESCRIPTION
0	(NULL)	
1		STROKE W/ WIDE BREAK
2	◇	ANYTHING
3	—	HORIZONTAL
4	∧	MAXIMUM
5	∨	MINIMUM
6	<	ANGLE W/O LOOP
7	=	STROKE W/ SMALL BREAK
8	•	DOT
9	+	CROSSOVER
10		BACKWARD UP ARC, FREE END AT START
11		BACKWARD UP ARC, FREE END AT END
12		BACKWARD UP ARC, NO FREE END
13		GAMMA UPSIDE DOWN, CCW
14		CIRCLE, CCW
15		GAMMA, CCW
16		FORWARD DOWN ARC, NO FREE END
17		FORWARD DOWN ARC, FREE END AT START
18		FORWARD DOWN ARC, FREE END AT END
19		FORWARD UP ARC, FREE END AT END
20		FORWARD UP ARC, FREE END AT START
21		FORWARD UP ARC, NO FREE END
22		GAMMA UPSIDE DOWN, CW
23		CIRCLE, CW
24		GAMMA, CW
25		BACKWARD DOWN ARC, NO FREE END
26		BACKWARD DOWN ARC, FREE END AT END
27		BACKWARD DOWN ARC, FREE END AT START
28	←	LEFT (ANY ARC)
29	→	RIGHT (ANY ARC)
30	|	VERTICAL COMPONENT
31	/	SUBARC AT MAXIMUM LEFT
32	◡	DOWN ARC (ANY DIRECTION)
33	\	SUBARC AT MAXIMUM RIGHT

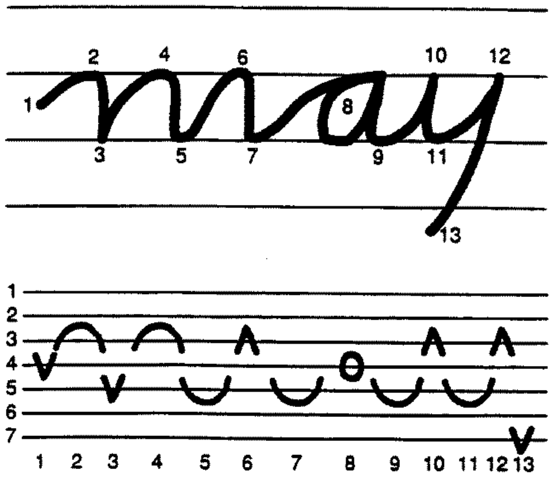


Figure 2.4: Metastroke categories and an example decomposition of a word into metastrokes [34].

Affine transformation is a technique which has been used for classification of Kanji characters [110, 111]. The difference between two characters is broken into two components: a stroke-based affine transformed component and a residual component which is the point-to-point distances between the affine transformed test character and a template. The final distance is then calculated as a combination of the sum of inter-character point distances between the original test character (before affine transformation) and a template, and the residual component between the two after affine transformation. Pavlidis et al. [77] have used a physics-based approach to online word recognition. This method first extracts critical points, which they define as points of high and low curvature, and points of inflection, and represents each character as the vector of these points. The distance between two characters is then measured by morphing one vector to another, and measuring the deformation energy defined by the required amount of stretching and compression of the line segments between critical points.

2.5.4 Hidden Markov Models

Hidden Markov models [57, 87] are often used in a similar fashion as elastic matching in that they represent the data in terms of its temporal sequence. Rather than representing a character or word as a group of templates, however, a hidden Markov model is trained to represent the population of data for each class. Hidden Markov models were first applied to speech recognition [3, 4, 45, 58] with great success. Makhoul et al. [101] have trained HMMs on sequences of two-dimensional features, writing

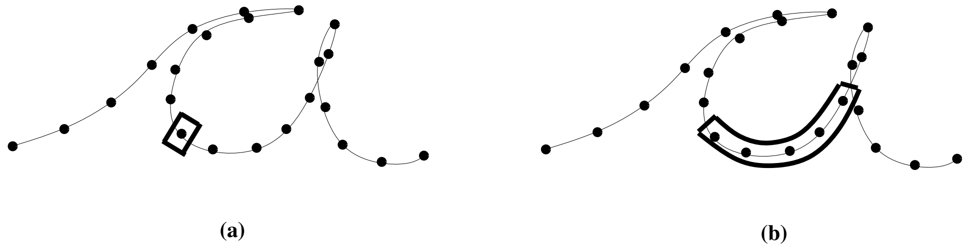


Figure 2.5: Feature extraction. (a) Point-by-point feature extraction (b) sliding window feature extraction. A sequence of features may be generated from a single sample point, or from a group of points that fall within a window which slides along the point sequence.

angle and change in writing angle, for each sample point. Many other approaches have made use of a *sliding window* - a window of sample points for which features are extracted, which slides along the sample point sequence thereby producing a sequence of features. Bellegarda et al. [8] train HMMs on a sequence of features computed for each of a number of *frames*. Frames are defined as a collection of 21 sample points centered at each x-axis or y-axis extrema along a written stroke.

2.5.5 Neural Networks

Time delay neural networks [36, 65, 92, 95] are often used to recognize characters or character segments as a sliding window passes over them. Features are extracted for the sample points in the sliding window and passed to the input layer of a feed-forward neural network. The array of output nodes of this network correspond to the different possible class identities that may be assigned to a sequence of points within the input window. The activation level of each output node corresponds to the likelihood that the sequence of points in the sliding window matches the corresponding identity for that output node. This then produces a sequence of likelihood values, which can be used to obtain the optimal sequence of characters. Manke and Bodenhausen [65] use a non-linear *dynamic time warping* algorithm to produce an alignment of words to the sequence of output values from the network. Other methods involving neural networks make use of hidden Markov models to form hybrid systems [9]. Schenkel et al. [92] have trained hidden Markov models for each character to model the sequence of output values from a neural network. Similarly, Rigoll et al. [89] have trained hidden Markov models in which the continuous densities of the state observations are represented using a neural network.

2.5.6 Local vs. Global Features

The focus of the features used in each of these methods can be characterized by the type of information provided (*local* vs. *global*) with respect to a word to be recognized. Elastic matching is a point-oriented method, and therefore the information used is considered very local, features used in HMMs and TDNNs are typically local at the level of a sliding window, and primitive decomposition focuses on higher-level sub-structures of a word. Global features are then characterizations of the word as a whole, such as its size or the presence or absence of descenders or ascenders, irrespective of their location in the word. Local features tend to be more robust than global features, since they can more easily handle shape variations, while global features tend to provide more information about the word's identity when the writing is well behaved. Hu et al. [43] combine low-level and high-level features by representing cusps, crossing points, and loops as point-oriented features which measure a point's distance from a high-level feature. Nalwa [70] proposes a technique that uses information at both local and global levels of a written word, and applies this to the problem of signature verification. Local information is a combination of 5 characteristic features that are computed for a sliding window. These features are center of mass, torque, and two moment of inertia measures. These are computed with respect to a coordinate frame that is a second sliding window which maintains a set distance along the curve from the first window. Nalwa defines global information as a combination of four functions: jitter (the difference between the raw stroke and smoothed stroke), aspect ratio, a measure of the amount of signature length warping applied, and the number of sample points in the signature. The local and global measures are combined using a biased harmonic mean. Using this method, Nalwa achieves an equal error rate (point at which the false accept and false reject rates are equal) of between $2\% - 5\%$ on datasets with a range of 43 - 102 different signers.

2.5.7 Writing Style-Based Models

Recently, there have been efforts to automatically identify different writing styles that exist in a set of writer-independent data, and to model each writing style separately in order to boost the performance of the recognizer. Clustering methods have been used to identify writing styles, and prototypical samples from each of these writing styles can be used to greatly reduce the time required to perform nearest neighbor classification [21, 83, 91, 115]. For parametric models, these clusters can be used to define multiple sub-class models that are more robust than a single class model due to reductions in the complexities of the resulting models. Bellegarda et al. [72] have identified multiple writing styles at the character level and modeled each of them using a hidden Markov model. Schomaker et al. [93, 109] have identified prototypical strokes, corresponding to writing styles, in a set of data taken from 30 writers using self-organizing maps. A hierarchical clustering of the data shows that certain writing styles appear to be based on nationality and gender. Schomaker et al. [94] make use of these prototypical strokes to perform writer-adaptation by retraining the transition net that connects these strokes to form words. These approaches currently impose one or more of the following restrictions:

1. The number of sub-classes must be pre-defined.
2. The clustering is done using a different representation space than used in classification.
3. The study is restricted to nearest neighbor classification.

The first restriction relies on a human expert to correctly choose the number of writing styles that exist for each character class. Not only can this prove to be a tedious task, it is difficult for a human to visualize the multidimensional temporal data in the same way as the highly complex models that will represent it. It should be noted that there may be significant differences in the temporal data between two characters, even when the static image representation of these characters appears very similar. In addition, if an automated method that is used for identification of the sub-classes makes use of a different representation space than will be used to model the data, as in the second restriction, there is no guarantee that the division of data is reasonable for the model space. Finally, the use of character-level nearest neighbor classifiers for word recognition requires a pre-segmentation of the words into characters. This makes the word classifier depend heavily on the ability of the segmentation algorithm to provide good segmentation alternatives. Other methods, such as hidden Markov models, have the ability to perform segmentation and classification at the same time, and are therefore less vulnerable to such problems.

In Chapter 3, we will present a method of sub-class identification in which the number of sub-classes is defined by an automatic method. In addition, we will show the use of these sub-classes for nearest neighbor and decision tree classifiers (Chapter 4), in which the representation space is the same as was used to identify the sub-classes, and hidden Markov models (Chapter 5). In Section 5.3, a method of identifying sub-classes in the representation space of hidden Markov models will also be presented.

2.6 Postprocessing

Many handwriting recognition techniques model data at the character or the stroke level. In a complete word-level recognizer, these models can be concatenated to form word-level models. A dictionary can be used to restrict the possible character combinations, thus providing some higher-level knowledge. This may be implemented as a grammar which specifies all possible character model combinations, forming a graph of character models. Often, prior probabilities of words are used to discourage misclassifications into infrequent word categories. The best path through this graph, as produced by a Viterbi alignment of a test word, can then be used to specify the characters that make up that word's identity. The restrictions of such a method are that the system has no chance of recognizing a word that is not in its vocabulary. Other, less restrictive methods, use character bigram or trigram probabilities to discourage unlikely character combinations. This is done at the cost of a reduced recognition accuracy.

2.7 Summary

In this chapter, an overview of the online handwriting recognition techniques from the current literature was given. The processes of data collection, preprocessing of the data, and segmentation at the word and character levels were discussed. The modeling techniques of primitive decomposition, motor models, deformation models, hidden Markov models, and neural networks have been described, along with a discussion of local and global features, and previous work involving the identification of writing-styles within a set of handwritten data. Finally, a discussion of common postprocessing techniques was presented.

Chapter 3

Writing Style Identification - Lexemes

3.1 Introduction

A partition of the training data for each character class into multiple sub-classes is motivated by the observation that there exist many different styles of writing a single character. The disparity between many of these styles motivates the construction of a separate model to represent each style. While it may be possible to enumerate many of the different styles in existence, and solicit examples of these styles from writers, there are two potential problems with this approach: i) new styles will most likely always exist which could not always be identified a priori, requiring the system designers to constantly search for new, so far unseen, writing styles (potentially a very tedious task); ii) the features used by a recognition system to classify a character are most likely different than the features used by a human, and therefore writing styles may be defined differently for the recognizer and the human eye. These factors motivate a reliable automatic method of discovering writing styles from a set of handwritten character examples without requiring human intervention.

This chapter describes our technique of automatic identification of the different handwriting styles, referred to as *lexemes*, within a set of handwritten character data. The chapter is organized as follows: Sections 3.2 and 3.3 describe a preprocessing and string matching distance measure, respectively, that are used to determine the similarity of two individual characters. Section 3.4 describes our technique of clustering these handwritten characters to identify possible lexeme groupings. The problem of determining the most appropriate number of lexemes for the given data is discussed, along with a description of the cluster similarity measure in Section 3.4.1. Section 3.5 discusses a technique by which lexeme models can be refined to better fit the representation space of the modeling technique used. Section 3.6 describes the set of writer-independent handwritten data that is used to identify lexemes in a writer-independent system.

3.2 Preprocessing

Figure 3.1 shows the steps involved in preprocessing. In our approach, we have applied a size normalization to each character such that all characters are of the same height, but retain their original aspect ratio. In addition, certain characters exist in which the height is very small (e.g., "." and "-") and this in itself is important to classification. These characters are detected by examining their height and aspect ratio. If either of

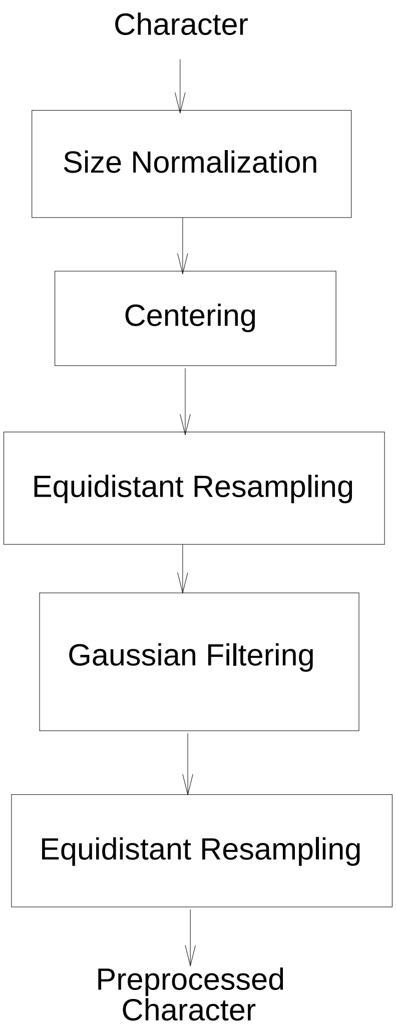


Figure 3.1: Preprocessing steps for handwritten characters.

these falls beyond a threshold value, then the character is not size normalized. Next, the character is centered to a standard reference point.

Following size normalization and centering, an equi-distant resampling of the data is applied, followed by a Gaussian filter applied independently to each of the x and y coordinates of the point sequence:

$$x_t^{filtered} = \sum_{i=-3\sigma}^{3\sigma} w_i \; x_{t+i}^{orig}, \tag{3.1}$$

where

$$w_i = \frac{e^{-i^2/2\sigma^2}}{\sum_{j=-3\sigma}^{3\sigma} e^{-j^2/2\sigma^2}}. \tag{3.2}$$

This is defined similarly for $y_t^{filtered}$. Equi-distant resampling must be done before the Gaussian filtering is applied so that the calculation of our filtered points is influenced by the points that are nearby in space rather than nearby in time. Throughout both of these operations, there are certain critical points such as endpoints and points of high curvature along the curve whose location we should preserve. These points are detected and included with the resampled points. Gaussian filtering is then applied independently to every sequence of points between two consecutive critical points, ensuring that these critical points remain in their original locations. Now that the digit is smoothed and the character size has been normalized, resampling is done one more time so that the final sequence of points is equi-distant with the proper spacing for feature extraction, once again preserving those points that have been marked as critical points. Figure 3.2 shows an example of the character 'a' after each preprocessing step. Figure 3.3 shows an example of three different digits before and after preprocessing. Critical points are marked on the preprocessed digits with dots.

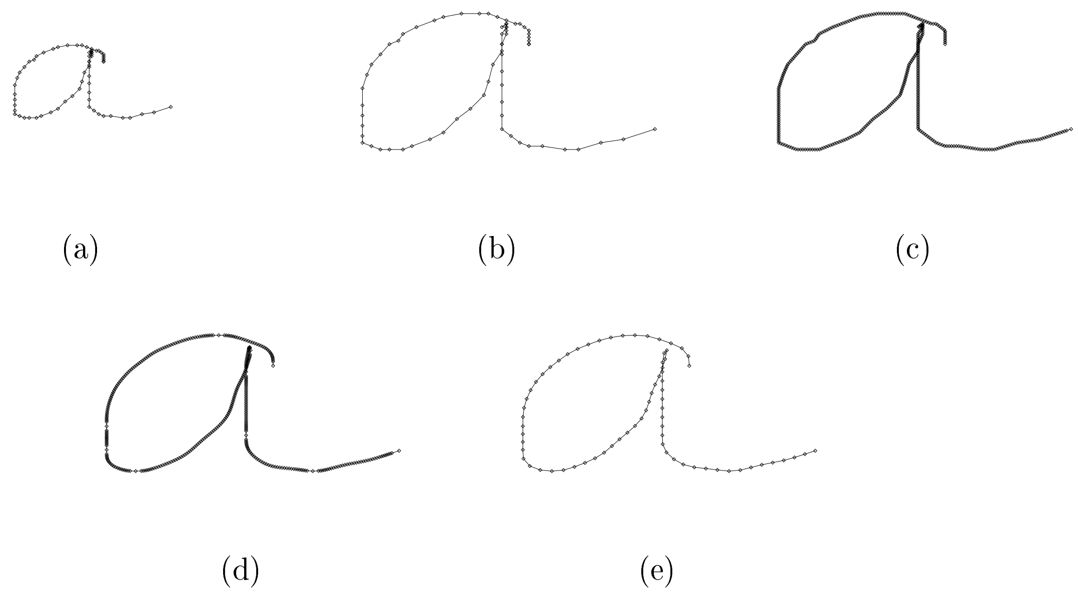


Figure 3.2: Preprocessing steps for character 'a' with sample points marked as dots: (a) original, (b) after size normalization and centering, (c) after first resampling, (d) after Gaussian filtering, and (e) after the final resampling.

3.3 String Matching Measure

A string matching technique is used to provide a distance measure between character pairs. A stroke is represented as a sequence of events (feature vectors), corresponding to the sequence of sample points in the stroke. This sequence forms a variable-length string. The distance between two different strings, $A = (e_1^A, \ldots, e_{N_A}^A)$ and $B = (e_1^B, \ldots, e_{N_B}^B)$, 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 string distances based on that alignment and the distances between the individual pairs of aligned event. The string matching technique presented here is a simplification of a technique that has been previously applied to fingerprint recog-

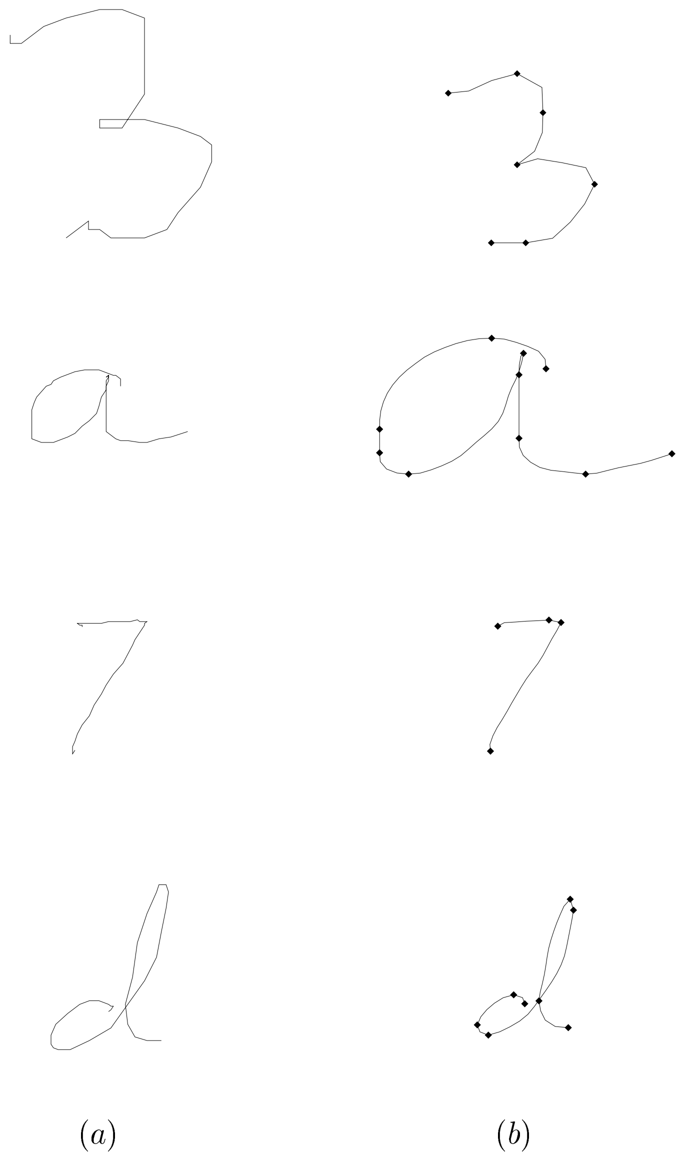
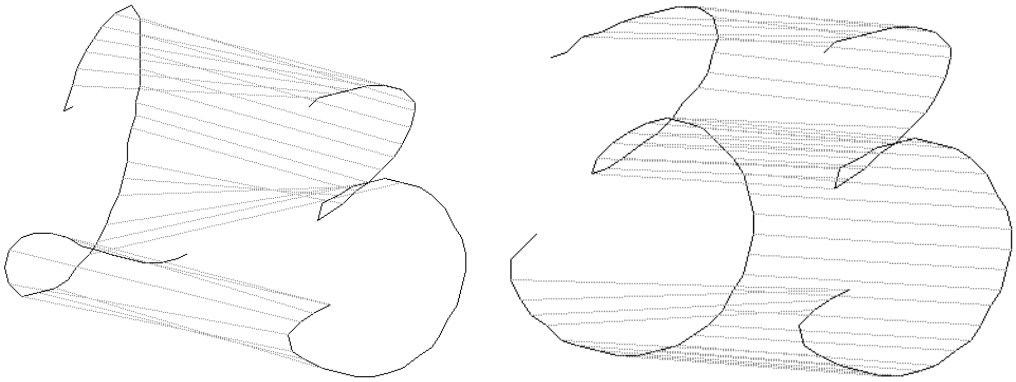


Figure 3.3: Examples of characters (a) before and (b) after preprocessing with critical points marked as dots.

nition [47]. Similar methods under the name *elastic template matching* have been applied to online handwritten data [103].



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

Figure 3.4: The alignment between two digits, and the resulting matching scores 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.

In our approach, each event (corresponding to a sample point after preprocessing) is represented with 3 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. Given an alignment of the events between two strokes, the distance, $d_e(i,j)$, between a pair of events, e_i^A from stroke A and e_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:

$$d_x(i,j) = |(x_i^A - x_1^A) - (x_j^B - x_1^B)| \tag{3.3}$$

$$d_y(i,j) = |(y_i^A - y_1^A) - (y_j^B - y_1^B)| \tag{3.4}$$

$$d_\theta(i,j) = Min(|\theta_i^A - \theta_j^B|,\;\; 360 - |\theta_i^A - \theta_j^B|) \tag{3.5}$$

$$d_e(i,j) = \alpha\, d_x(i,j) + \beta\, d_y(i,j) + \gamma\, d_\theta(i,j), \tag{3.6}$$

where all θ are in the range (0, 360), and α, β, and γ are the weights of the linear combination. Appropriate values for these weights are determined empirically. In general, 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.

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:

$$D(i,j) = Min \begin{cases} D(i-1, j-1) + d_e(i,j) & \\ D(i-1, j) + \text{Missing Penalty} & 1 \leq i \leq N_A \\ D(i, j-1) + \text{Spurious Penalty} & 1 \leq j \leq N_B \\ D(i-1, j-1) + \text{Missing Penalty} + \text{Spurious Penalty} & \end{cases} \tag{3.7}$$

$$Dist(A, B) = D(N_A, N_B). \tag{3.8}$$

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

$$Dist_{SP}(A, B) = \frac{Dist(A, B)^2}{Norm\_Factor(N_A, N_B)} + (SP)|S_A - S_B|, \tag{3.9}$$

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 that can be calculated between any two strings of lengths N_A and N_B scaled by a constant factor.

3.4 Squared-Error Clustering

Clusters of character samples are formed using a squared-error clustering algorithm, called CLUSTER [48]. In order to make use of this technique, we require that each character be represented by a common feature vector. However, our data does not have such a representation since it is in the form of a string. An approximation to a Euclidean feature space can be obtained by representing each character as the vector of distances to every character in the feature space. Using the distances calculated by our string matching method, a proximity matrix is constructed for each character class. Each matrix measures the intra-class distances for a particular character class. Clustering is then done in which the feature vector for a character is its M distances to each one of the M characters belonging to the same class.

CLUSTER attempts to produce the best clustering for each value of K over some range, where K is the number of clusters into which the data is to be partitioned. However, the final decision as to what value of K is most appropriate is still left up to the user. Figure 3.5 shows an example of the number of clusters plotted against the total intra-cluster mean squared error for the character class 'm'. As expected, the mean squared error decreases monotonically as a function of K. The "optimal" value of K can be chosen by identifying a "knee" in the curve. An inspection of Figure 3.5 shows that identification of a knee is not easily accomplished. This plot is typical for most characters, making the choice of K very difficult by this method.

3.4.1 Selecting An Appropriate Number of Clusters

Davies and Bouldin [25] present a measure of cluster similarity for a set of clusters. Their measure can be used to find an "optimal" value of K for which the cluster similarity is minimal, or equivalently, the cluster separation is maximal.

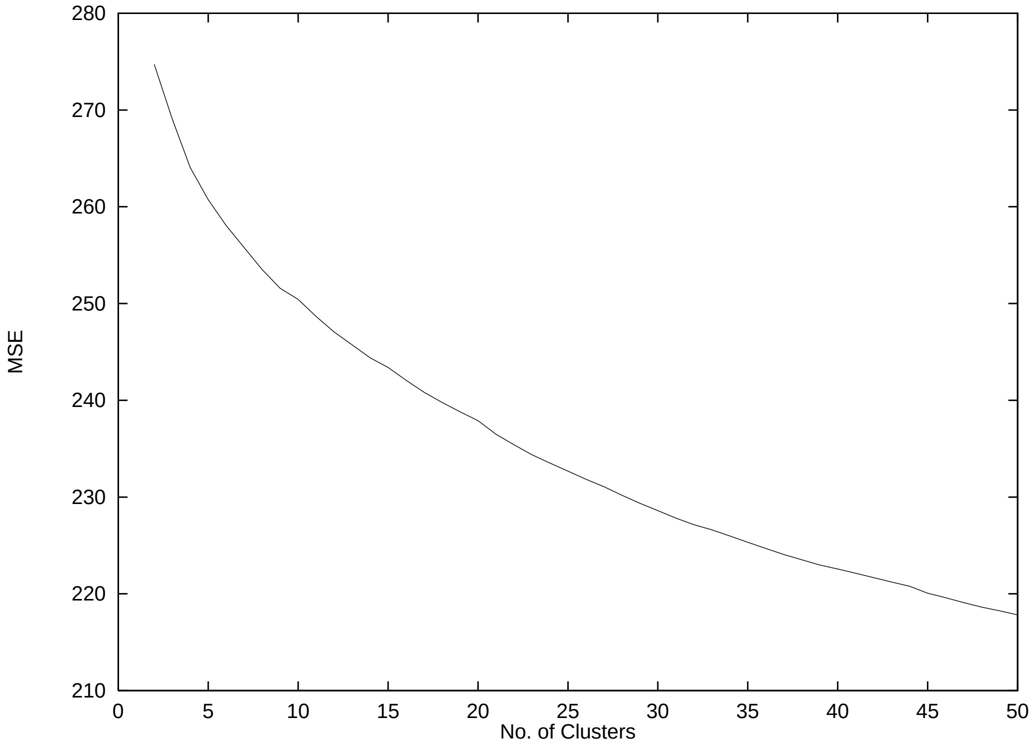


Figure 3.5: Mean squared error (MSE) for different values of K, the number of clusters, for the character 'm'.

The cluster similarity measure is defined as follows: For cluster i with cluster center A_i, population size T_i, and cluster members $(X_1, \ldots, X_{T_i})$, the dispersion of the cluster is defined as

$$S_i = \{\frac{1}{T_i}\sum_{j=1}^{T_i} ||X_j - A_i||^q\}^{1/q}, \quad q > 0 \tag{3.10}$$

where A_i is the vector $(a_{i1}, \ldots, a_{iK})^t$, and X_j is the vector $(x_{j1}, \ldots, x_{jK})^t$. The inter-cluster distance is defined as

$$M_{ij} = \{\sum_{k=1}^{K} |a_{ki} - a_{kj}|^p\}^{1/p}, \quad p > 0. \tag{3.11}$$

From these two measures, the similarity between two clusters i and j is defined as

$$R(S_i, S_j, M_{ij}) = \frac{S_i + S_j}{M_{ij}}, \tag{3.12}$$

and the similarity of a set of K clusters is defined as

$$\bar{R} = \frac{1}{K} \sum_{i=1}^{K} \max_{j \neq i} R(S_i, S_j, M_{ij}), \quad 1 \leq j \leq N. \tag{3.13}$$

Similar statistics have appeared elsewhere in the literature [19, 26]. Generally, p and q are set to 2 in which case S_i becomes the standard deviation of cluster i, and M_{ij} becomes the Euclidean distance between the center of cluster i and the center of cluster j.

3.5 HMM-based Clustering

The method of lexeme identification presented thus far in this chapter relies on clustering in a representation space which is based on the string matching measure. This seems appropriate if string matching also forms the basis of our classifier. However, in Chapter 5 we shall discuss the use of hidden Markov models (HMM) for classification. Ideally, lexemes should be defined using the same feature space as will be used to model these lexemes, and the same distance measure as will be used for recognition. A clustering of the data using the HMM likelihood as a distance measure between an HMM trained for a cluster, and a single character, can be defined such that a clustering can be done in HMM probability space. Given some initial set of clusters, one can iteratively train one model for each cluster and then redefine the membership of each cluster by assigning each character example to the cluster whose model, within the same character class, gives the maximum likelihood score for that example. Details of this method will be described in Section 5.3.

3.6 Results

The online handwriting database used in our experiments is particularly challenging 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 Figure 3.6.

Experiments were run using a set of 17,947 alphanumeric characters for training, and 17,928 for testing, from a combined pool of data representing 21 writers and a total of 36 classes: 10 digits and 26 handsegmented lowercase cursive letters[1]. An example of the cluster similarity measure for different values of K is shown in Figure 3.7. The final number of clusters chosen for each of the 36 character classes is shown in Table 3.1.

Table 3.1: Number of clusters, K, 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

[1]Data provided by the Pen Computing group at IBM Watson Research Center.

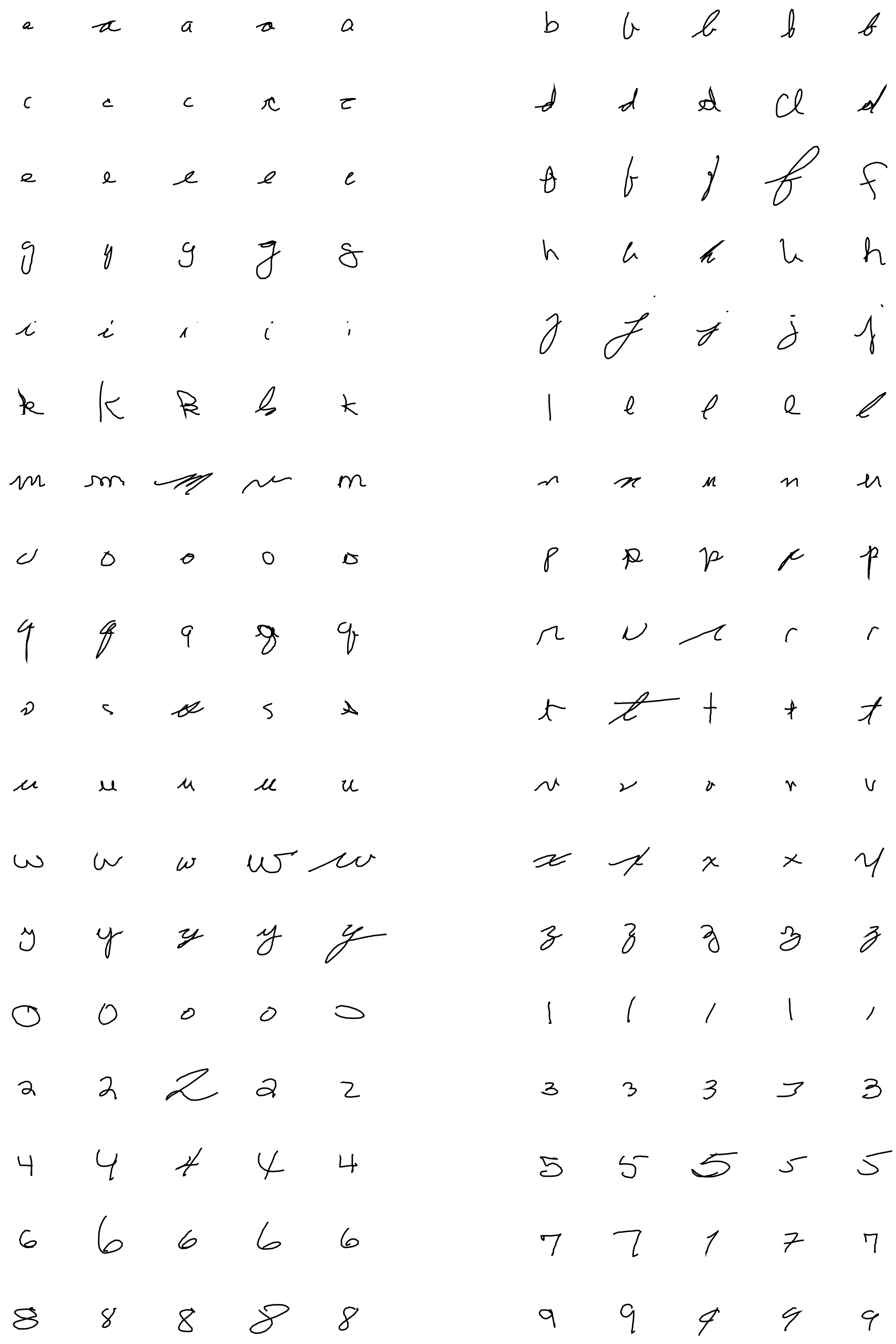


Figure 3.6: Examples of characters in our database.

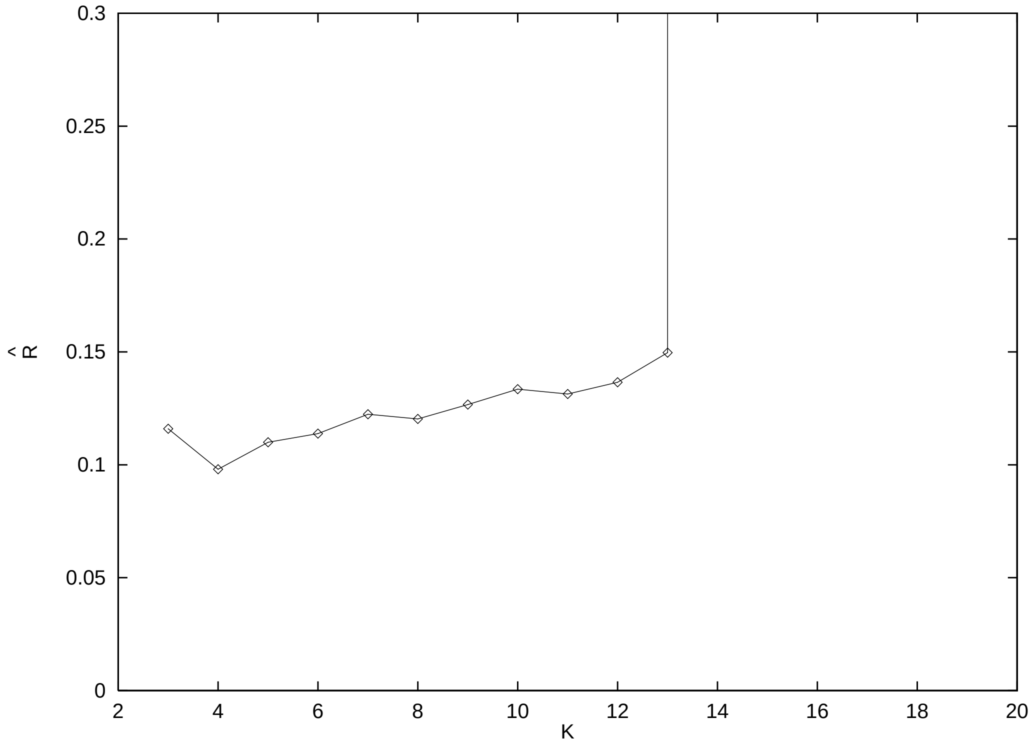


Figure 3.7: Davies and Bouldin index, $\hat{R}$, for the digit "0" over different values of K, the number of clusters.

Figure 3.8 shows the character prototypes closest to the centers of each of the clusters identified within each of the 36 character classes. We see that most of the prototypes appear to have different shapes than the other prototypes of the same character class. Figure 3.9 shows some randomly selected samples from the clusters for two different lexemes of the digit '2'. This figure shows the intra-lexeme variations that occur in the data.

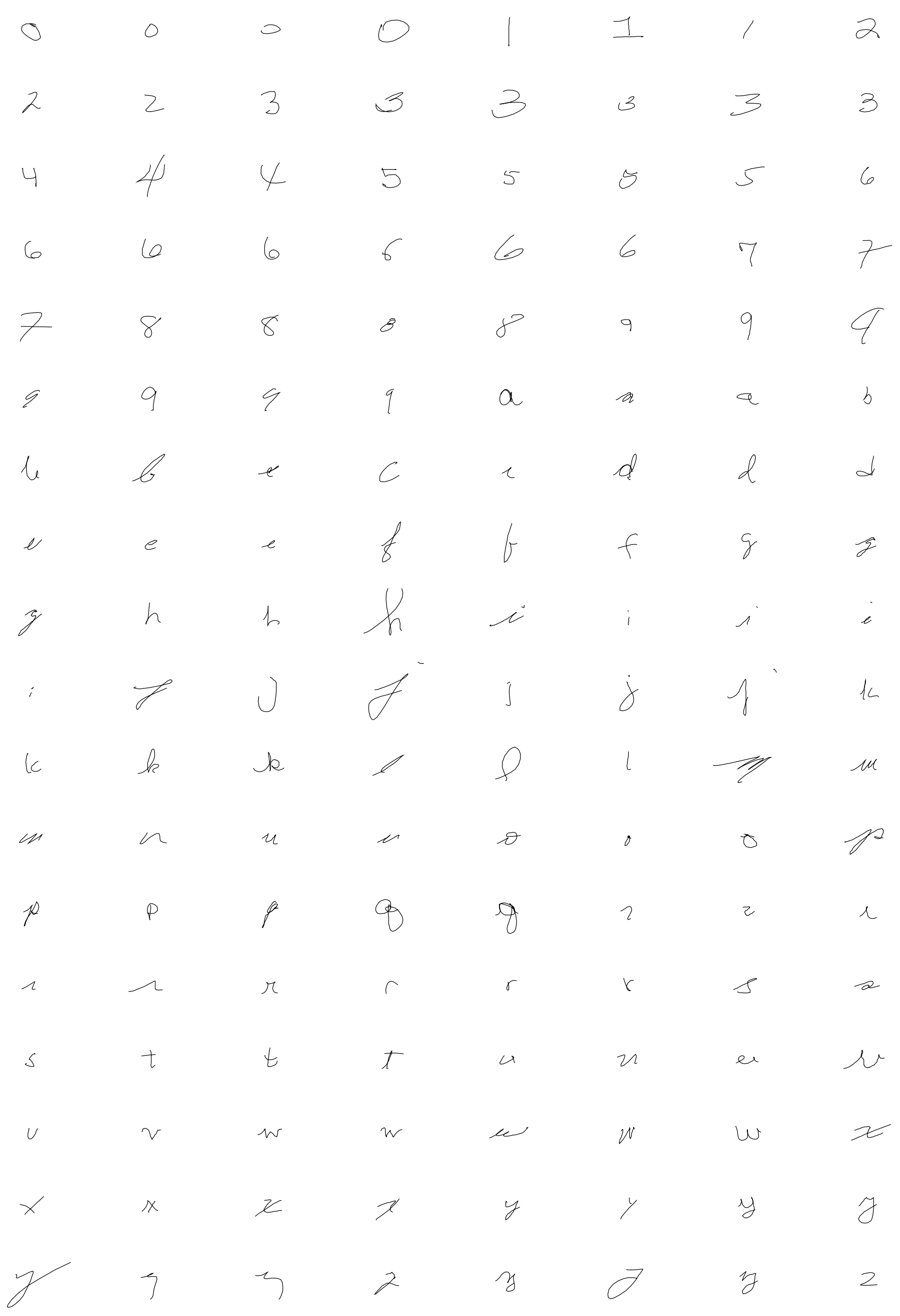


Figure 3.8: Examples of the patterns closest to each cluster center found for each character class.

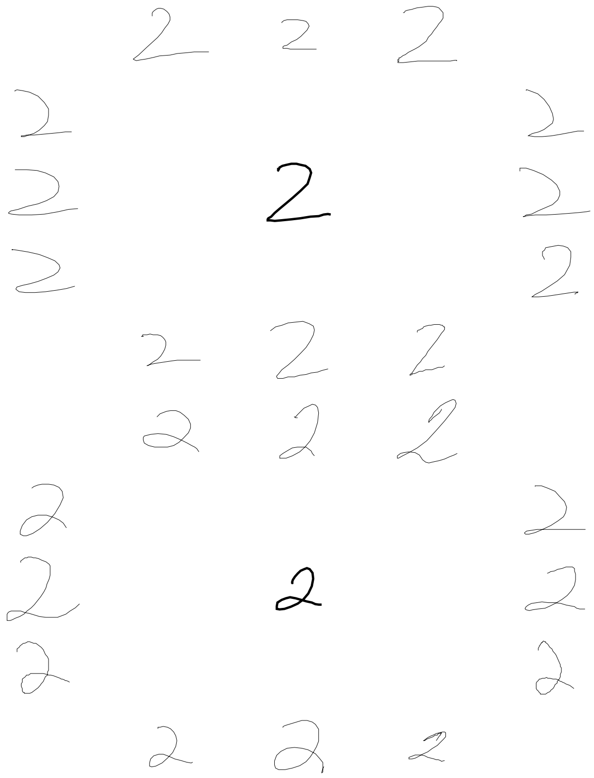


Figure 3.9: Two different lexemes of the digit '2' as identified by our method. The digit in the center of each cluster shown here, is the training sample that lies closest to the cluster center.

3.7 Summary

A method of identifying different styles, or lexemes, that exist in a set of online handwritten character data has been presented. Lexeme identification is accomplished through the use of a string matching measure to calculate the distance between any two characters in the data. Within each character class, the data is partitioned into clusters, one for each lexeme, using the CLUSTER algorithm [48]. The final decision regarding the number of clusters is made using the Davies and Bouldin cluster separation measure [25]. This method will be used in the following chapters to create lexeme models used in character and word classification tasks.

Chapter 4

Non-parametric Models

4.1 Introduction

Non-parametric methods for designing classifiers have the potential to use, to the fullest extent, the available data (templates) to perform the classification. In the asymptotic sense, a nearest neighbor method achieves the accuracy of the optimal Bayes rule [48]. However, this requires the test pattern to be compared to all the training samples. This requires a (possibly linear) increase in classification time as the number of templates is increased. There is thus, a tradeoff between classification time and accuracy.

A number of data reduction methods such as nearest neighbor editing have been reported in the literature that strive to reduce the number of templates that must be retained in the training set, however the size of this reduced set is still a function of the original training set size. A further complication exists when the data cannot be represented as patterns in the Hilbert space, as is the case in our string representation of online characters. We demonstrate a system for online handwriting recognition that makes use of a reduced set of prototypical templates where the number of retained templates is determined automatically, and is approximately a function of the number of writing styles rather than the amount of training data. Experiments are conducted in which these templates are then used as a reference for nearest neighbor classification, and for classification using decision trees. 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. These methods were previously introduced in [22]. An overview of the system is shown in Figure 4.1.

4.2 Cluster Centers

One of the methods of identifying representative prototypes from the training data is to cluster the data (defining lexemes) and retain the pattern (prototype) closest to each cluster center. The clustering technique used here is based on the string matching measure as described in Chapter 3 and is used to create a set of reference templates, where each template is the closest character to the center of each lexeme's cluster.

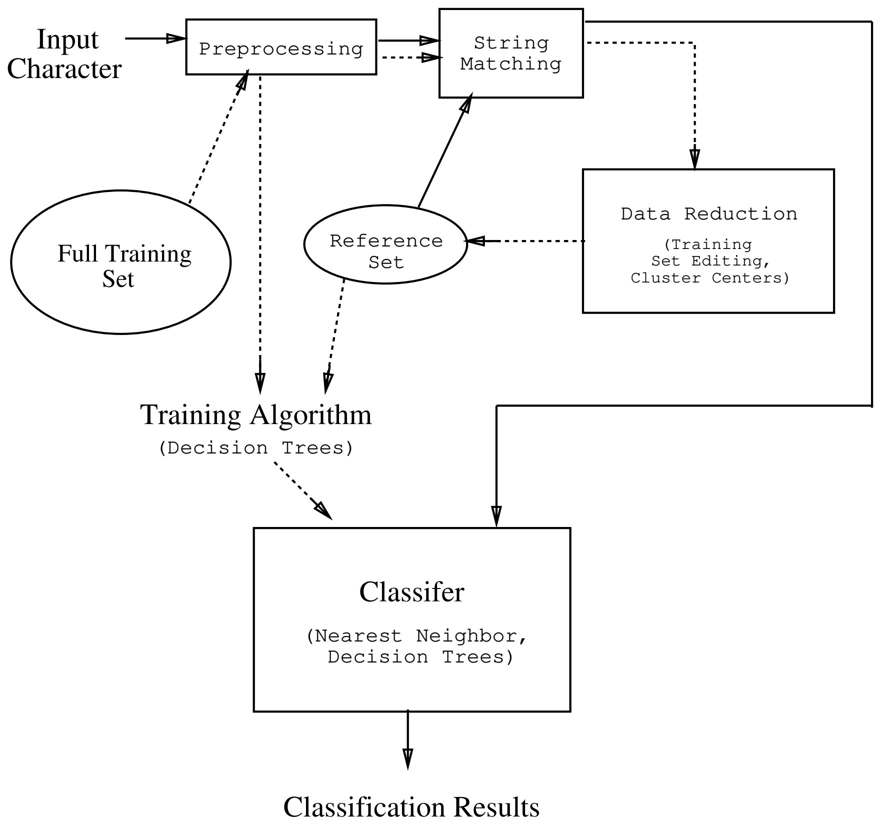


Figure 4.1: Online character recognition system. The flow of data during training is shown by the dashed line arrows, while the data flow during recognition is shown by the solid line arrows.

4.3 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. These selected templates then make up the reduced training set. Common methods of editing involve first deriving the Voronoi tessellation of the training set, and then removing any training example whose cell does not border the cell of an example of a different 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 inter-pattern distances rather than their locations in d-dimensional feature space, this method of deriving the boundary patterns cannot be applied. Alternatively, we have used the following algorithm to identify border patterns. This method relies only on knowing the inter-class nearest neighbors of each character sample.

<u>Algorithm for constructing the Reduced Training Set</u>

$T_{Orig} \equiv$ 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 $NN_c(x_i) \notin T_R$

      $NN_c(x_i) \rightarrow T_R$

At the completion of this algorithm, T_R contains only those patterns which have been identified by other patterns as being their nearest neighbor across a class boundary.

4.4 Classification

We discuss two different methods of classification: nearest neighbor because of the high accuracies that can be obtained as the training set size becomes large, and decision trees because of their efficient speed of classification. Both of these methods can be used in combination with the data reduction techniques from the previous section.

4.4.1 Nearest Neighbor Classifier

Nearest neighbor classification is a common technique [26] used in template-based approaches. We make use of the string matching measure described in Section 3.3 as the distance between patterns, and assign a test character to the class of the closest template character as defined by the string matching distance.

4.4.2 Decision Tree Classifier

Decision trees [13] 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 (see Figure 4.2). 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:

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

$$\text{otherwise traverse the right branch of the node } n.$$

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:

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

where

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

```
Decision tree:

d3.6 <= 232:
:...d0.79 <= 173:
:   :...d2.48 <= 135:
:   :   :...d1.36 <= 394: Class 0
:   :   :   d1.36 > 394: Class 7
:   :   d2.48 > 135:
:   :   :...d2.11 <= 153: Class 5
:   :       d2.11 > 153: Class 4
:   d0.79 > 173:
:   :...aspect > 3.91667: Class 1
:       aspect <= 3.91667:
:       :...strokes in {3,4,5}: Class 2
:           strokes = 2:
:           :...d7.52 <= 128: Class 7
:           :   d7.52 > 128:
:           :   :...d9.17 <= 289: Class 5
:           :       d9.17 > 289: Class 7
:           strokes = 1:
:           :...d7.16 <= 66: Class 7
:               d7.16 > 66:
:               :...d0.1 <= 325:
:                   :...d3.32 <= 53: Class 3
:                   :   d3.32 > 53: Class 2
:                   d0.1 > 325:
:                   :...d3.6 > 220: Class 7
:                       d3.6 <= 220:
:                       :...d5.17 <= 153: Class 3
:                           d5.17 > 153: Class 2
d3.6 > 232:
:...d1.36 <= 78: Class 1
    d1.36 > 78:
    :...d6.3 <= 158:
        :...d8.47 <= 288:
        :   :...strokes = 2: Class 4
        :   :   strokes in {3,4,5}: Class 0
        :   :   strokes = 1:
        :   :   :...d3.44 <= 223: Class 1
        :   :       d3.44 > 223: Class 0
        :   d8.47 > 288:
        :   :...d3.47 <= 320: Class 6
        :       d3.47 > 320:
        :       :...d0.1 <= 34: Class 0
        :           d0.1 > 34: Class 1
        d6.3 > 158:
        :
        :
```

```
Reference Character Set:

( d0.1, d0.79, d0.84, d0.51, d1.62, d1.23,
  d1.44, d1.9, d1.36, d2.11, d2.77, d2.68,
  d2.15, d2.48, d2.20, d2.61, d3.6, d3.33,
  d3.32, d3.47, d3.23, d3.44, d4.16, d4.33,
  d4.39, d5.17, d5.5, d5.8, d6.3, d6.37,
  d6.48, d6.23, d7.16, d7.52, d7.7, d7.31,
  d8.5, d8.32, d8.2, d8.47, d8.27, d9.17,
  d9.32, d9.44, d9.7, d9.26, d9.55, d9.49,
  d9.34, d9.43, d9.15 )

Global Features:

( aspect (aspect ratio of test char.),
  strokes (no. of strokes in test char.) )
```

Figure 4.2: An example of part of a decision tree using *similarity* features. Distances from a test character to a reference character are indicated in the tree by the name of the reference character. This is shown as dx.y, where x is the digit class, and y is the index of that reference digit from the training set.

These features shall be referred to as the *difference* features (see Figure 4.3). 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 training 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.

Figure 4.4 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 4.5, 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.

4.4.3 Global Measurements

It is often useful to use both global and local features of characters for classification. When using a nearest neighbor classifier we require some good method of combining these features to compute the inter-pattern distance. We use a weighted combination of global measurements (differences in the stroke counts) with local measurements (the string matching distance). The weights in this combination were determined empirically using a subset of the training data to evaluate the recognition accuracy

```
Decision tree:

d2.15-d6.23 <= -44:
:...aspect > 3.91667: Class 1
:   aspect <= 3.91667:
:   :...strokes in {3,4,5}: Class 2
:       strokes = 2:
:       :...d4.16-d7.52 > 101: Class 7
:       :   d4.16-d7.52 <= 101:
:       :   :...d2.15-d2.11 <= -13: Class 4
:       :       d2.15-d2.11 > -13: Class 5
:       strokes = 1:
:       :...d3.6-d7.16 > 37: Class 7
:           d3.6-d7.16 <= 37:
:           :...d5.17-d1.23 <= -16: Class 3
:               d5.17-d1.23 > -16:
:               :...d0.1-d6.3 <= 84: Class 2
:                   d0.1-d6.3 > 84: Class 3
d2.15-d6.23 > -44:
:...d1.44-d9.32 <= -60: Class 1
    d1.44-d9.32 > -60:
    :...d6.3-d7.31 <= -304:
        :...d0.1-d6.3 <= 11: Class 0
        :   d0.1-d6.3 > 11: Class 6
        d6.3-d7.31 > -304:
        :...d9.44-d8.27 <= -73:
            :...d8.2-d9.7 > -25: Class 9
            :   d8.2-d9.7 <= -25:
            :   :...d0.79-d4.16 <= -7: Class 5
            :       d0.79-d4.16 > -7: Class 8
            d9.44-d8.27 > -73:
            :
            :
```

```
Reference Character Set:

( d0.1, d0.79, d0.84, d0.51, d1.62, d1.23,
  d1.44, d1.9, d1.36, d2.11, d2.77, d2.68,
  d2.15, d2.48, d2.20, d2.61, d3.6, d3.33,
  d3.32, d3.47, d3.23, d3.44, d4.16, d4.33,
  d4.39, d5.17, d5.5, d5.8, d6.3, d6.37,
  d6.48, d6.23, d7.16, d7.52, d7.7, d7.31,
  d8.5, d8.32, d8.2, d8.47, d8.27, d9.17,
  d9.32, d9.44, d9.7, d9.26, d9.55, d9.49,
  d9.34, d9.43, d9.15 )


Global Features:

( aspect (aspect ratio of test char.),
  strokes (no. of strokes in test char.) )
```

Figure 4.3: An example of part of a decision tree using *difference* features. The the reference pair used to make a decision at a node is labeled dx_1.y_1-dx_2.y_2, where x_i is the digit class, and y_i is the index of that reference digit from the training set.

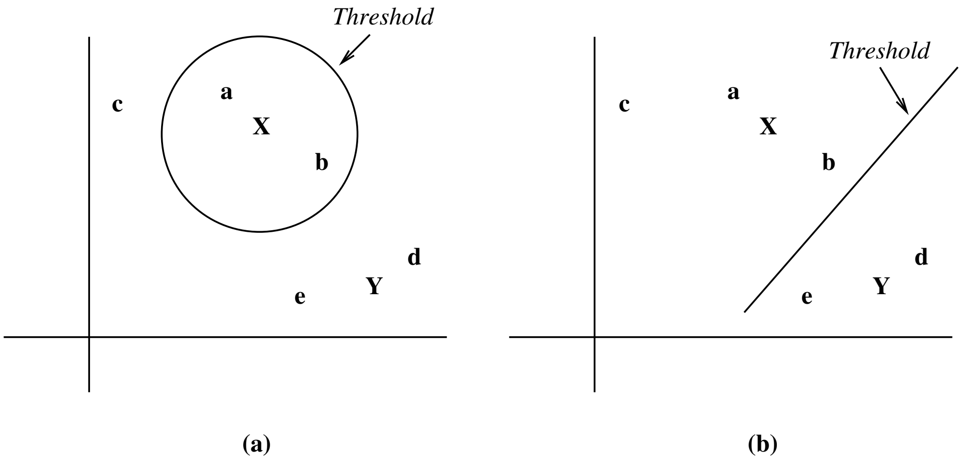


Figure 4.4: *Similarity* and *difference* features. (a) *Similarity* features: while training patterns a and b are grouped based on their similarity, to template X, examples c and d, although very different from each other, will be grouped together; (b) *Difference* features: training patterns on each 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.

for different values of the weights. Other global features, such as aspect ratio, are implicitly represented by the string matching measure since we maintain the original aspect ratio of the character during size normalization.

A preferred method of determining how to combine global and local measurements would be to use an automated learning method. 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.

4.5 Results

This section compares classification results for both nearest neighbor and decision tree classifiers based on the full training set, edited training set, and cluster centers. All timing results were obtained using a 296MHz UltraSparc workstation.

4.5.1 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 [21]; (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 DC, was produced by the same group of 21 writers as DA.

4.5.2 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 4.1 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., Figure 3.5), or automatic selection based on Davies and Bouldin's method [25]. For automatic selection, we consider only cluster counts greater than 2.

Table 4.1: Number of clusters, K, 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

Table 4.2: Digit recognition rates using the nearest neighbor classifier.

Training Set	No. Templates	Throughput	Recog. Rate
Full Set (DA)	600	˜2 char/sec	91.10%
Manually Selected K	51	˜26 char/sec	88.90%
Auto. Selected K	44	˜31 char/sec	87.50%
Edited	402	˜4 char/sec	91.39%

Nearest Neighbor Classifier

Representative templates were chosen, one for each cluster, based on their proximity to the center of the cluster. Table 4.2 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 a substantial reduction in classification time, however, it surprisingly results in a slightly better recognition accuracy.

Decision Trees

The decision tree was implemented using the package C5.0 [86]. 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. In addition, two global features, *number of strokes* and *aspect ratio*, were not used in computing the distance measures, but were added as two additional features in the decision tree, as discussed in Section 4.4.3.

Table 4.3 shows the results of the digit classification problem using decision trees where the reference templates are 51 cluster centers, the edited training set of size 402, and the full training set of 600 digits, respectively. *Similarity* and *difference* features are shown for decision trees where the set of cluster centers is the reference set. Due to it's large dimensionality and computational considerations, only the *similarity* features are shown when the reference set consists of the full training set, or the edited reference set. As can be seen in Table 4.3, the use of decision trees results in a great reduction in the average number of distances that must be calculated in order to classify a test digit. However, this comes at the cost of a significant reduction in recognition accuracy. Among the features based on cluster centers, however, it is apparent that the *difference* features produce superior results compared to the *similarity* features, at the cost of a 48.2% increase in the number of distances that must be calculated per test sample.

Table 4.4 shows how the results of Table 4.3 are changed when the effects of the

Table 4.3: Digit recognition rates using the decision tree.

	Cluster Centers		Edited	Full Set
	Similarity	Difference	Similarity	Similarity
Avg. Dist. Calcs	5.6	8.3	5.1	5.1
Throughput	~ 77 char/sec	~ 50 char/sec	~ 67 char/sec	~ 91 char/sec
Recog. Rate	73.3%	79.2%	76.4%	80.3%

Table 4.4: Digit recognition rates using the decision tree when global traits are treated as separate features.

	Cluster Centers		Edited	Full Set
	Similarity	Difference	Similarity	Similarity
Avg. Dist. Calcs	4.99	8.38	4.24	4.7
Throughput	~ 91 char/sec	~ 59 char/sec	~ 100 char/sec	~ 83 char/sec
Reco Rate	79.7%	81.1%	74.2%	71.7%

global features (*number of strokes* and *aspect ratio*) are removed from the distance measures, and are added as two additional features in the decision tree. A comparison of Table 4.4 with Table 4.3 shows that, in the case of the cluster center reference sets, representing these global traits as separate features produces an increase in recognition accuracy along with a decrease in the average number of distances that must be calculated to classify a test digit since distances need only be calculated that correspond to tree nodes traversed during classification. However, when either the edited training set or the full training set is used, the reduction in the average number of distances calculated is achieved at the cost of recognition accuracy.

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

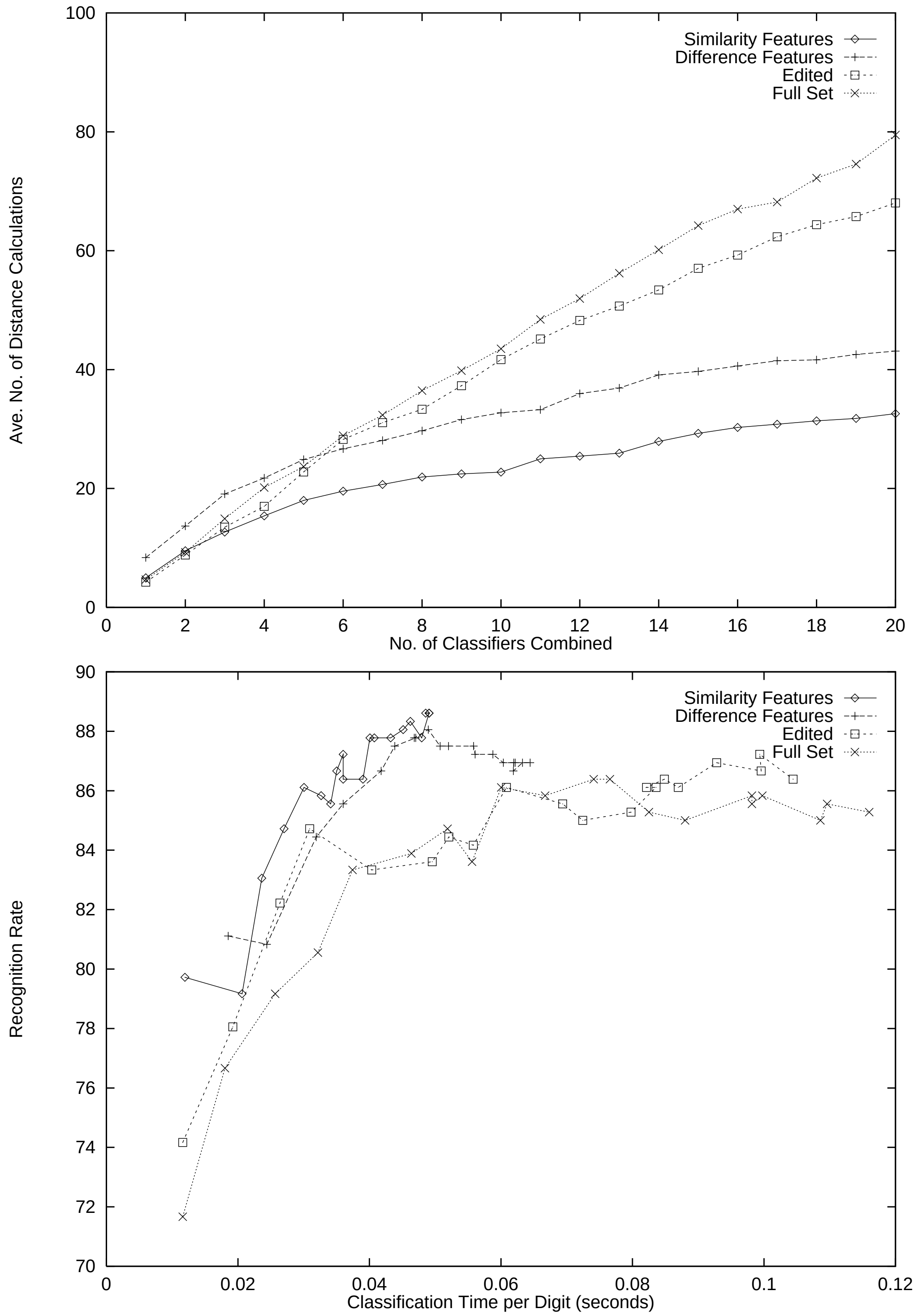


Figure 4.5: Digit recognition by combining multiple tree classifiers using boosting. Trees are constructed using the *similarity* and *difference* features based on cluster center reference sets, and on the edited reference set, and *similarity* features based on the full training set.

Table 4.5: Digit recognition using boosted decision trees. The column "No. Templates" indicates the number of reference templates available to the classifier, and the average number of templates used (number of distances that had to be calculated) per classification (shown in parenthesis).

Reference Set	Features	No. Templates	Throughput	Recog. Rate
Cluster Centers	Similarity Features (18 trees combined)	51 (31.4 ave.)	˜34 char/sec	88.6%
Cluster Centers	Difference Features (9 trees combined)	51 (31.6 ave.)	˜27 char/sec	88.1%
Edited	Similarity Features (19 trees combined)	402 (65.7 ave.)	˜17 char/sec	87.2%
Full Set	Similarity Features (11 trees combined)	600 (48.4 ave.)	˜22 char/sec	86.4%

technique called *boosting* [85]. Figure 4.5 shows the decrease in error rate for decision trees as the number of classifiers used in boosting is increased. This is shown 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. As the number of classifiers that are combined is increased, so is the average time required to classify each character, while the error rate approaches some lower bound (see Figure 4.6 for an example). The best results are summarized in Table 4.5 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 4.5 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 per second. 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 per second (Table 4.2).

4.5.3 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 DC, 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 3.1.

Nearest Neighbor Classifier

Table 4.6 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 ($K = 2$, $K = 3$) over a 1-NN classifier in either the digit or alphanumeric classification cases. 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 classifica-

Table 4.6: Alphanumeric recognition rates (36 classes) using nearest neighbor classifier.

Training Set	No. Templates	Throughput	Recog. Rate
Full Set	17,947	˜0.05 char/sec	88.92%
Cluster Centers	144	˜6 char/sec	45.41%
Edited	11,867	˜0.08 char/sec	87.88%

tion time while maintaining most of the recognition accuracy. However, the overall computational requirement using this set, 12.18 seconds per test character, cannot be considered practical.

Decision Trees

Figure 4.6 shows the results of classifying the 36-class alphanumeric character 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 in the decision tree. This reduced the number of *difference* features from 20,736 (144 by 144) to 644. As Figure 4.6 shows, the decision tree classifier does a better job of discriminating these 36 classes through the use of it's 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 over 8 characters per second (see

Table 4.7: Performance of nearest neighbor vs. decision tree classifiers for 36-class alphanumeric character classification. The column "No. Templates" indicates the number of reference templates available to the classifier, and the average number of templates used (number of distances that had to be calculated) per classification (shown in parenthesis).

Classifier	No. Templates	Throughput	Recog. Rate
NN - Full Set	17,947	˜0.05 char/sec	88.92%
NN - Cluster Centers	144	˜6 char/sec	45.41%
Decision Tree - Similarity (23 trees combined)	144 (100.1 ave.)	˜11 char/sec	86.1%
Decision Tree - Difference (20 trees combined)	144 (130.5 ave.)	˜8 char/sec	86.9%
Decision Tree - Combined (18 trees combined)	144 (127.7 ave.)	˜8 char/sec	86.9%

Table 4.7). 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.

4.6 Summary

We have demonstrated a method of online character recognition using non-parametric models 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. Through this character representation, we are able to obtain an 86.9% classification accuracy for a 36-class (alphanumeric characters) problem with a throughput of over 8 characters per second on a 296MHz

Sun UltraSparc workstation. There is a reduction in computation time of 99.4% as compared to the time requirements for a nearest neighbor classifier using the full set of available training examples. This is achieved by capturing the within-character class variability in terms of lexemes (clusters). Based on these experimental results, we draw the following conclusions:

- Cluster centers are reasonable prototypes for the individual characters for a nearest neighbor classifier in the case of digit classification, where the ratio of number of clusters to training data is not too small ($\frac{44}{600} = 0.073$ in the automated case). However, in the alphanumeric classification case where this ratio is very small ($\frac{144}{17948} = 0.008$), we require some additional information about the structure of these clusters.

- Representing the categories by their border patterns allows us to reduce the number of patterns that must be stored by approximately one third with almost no reduction in classification accuracy for the nearest neighbor classifier. However, since this data reduction is still related to the size of the original training set, classification time remains a function of the training set size. Cluster centers, on the other hand, are a function of the number of writing styles present in the data, and therefore not directly related to the size of the training set.

- The use of cluster centers produces a higher recognition accuracy for decision trees. This is an indication that the cluster centers do contain the relevant information for classification.

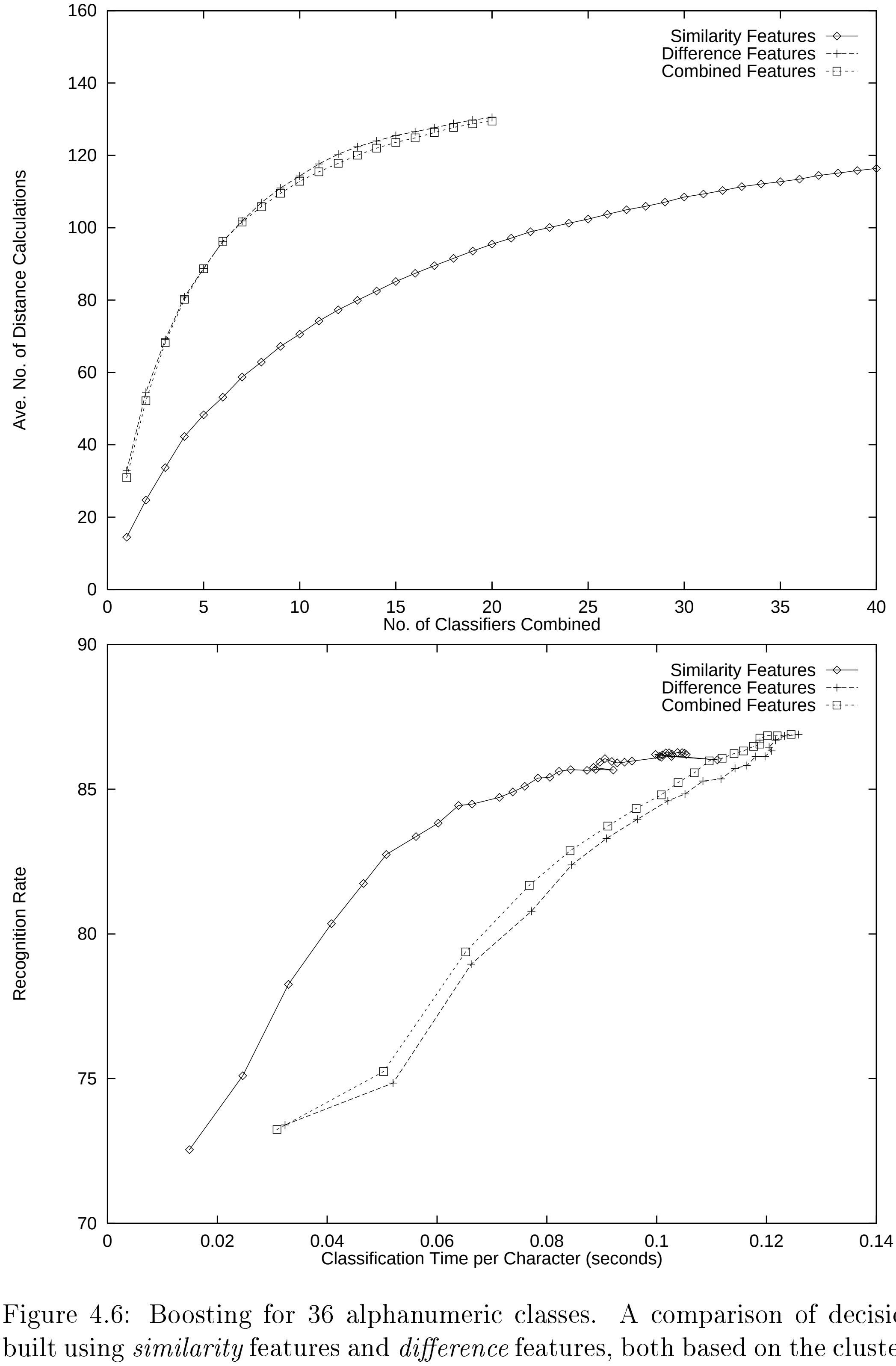


Figure 4.6: Boosting for 36 alphanumeric classes. A comparison of decision trees built using *similarity* features and *difference* features, both based on the cluster center reference set, and a combination of the two feature sets.

Chapter 5

Parametric Models

5.1 Introduction

Parametric models over make some assumptions about how the patterns from individual categories are distributed, estimate the parameters of that distribution, and thereby attempt to represent the data in a compact form. Unlike non-parametric models, classification time is then not a function of the amount of training data used to estimate that distribution, but of the number and complexity of the models that represent the distribution. If our assumptions about the underlying form of the distribution are valid, then as the amount of training data is increased, the performance of the parametric approach will approach the "optimal" value. Lexeme modeling allows us to represent complex class-conditional densities for individual categories as a combination of simpler distributions (lexeme distributions). In this chapter, we explore the advantages of constructing separate parametric models (hidden Markov models) for each lexeme. In addition, we describe another method of identifying lexemes in the representation space defined by the hidden Markov models.

5.2 Hidden Markov Model Classifier

We describe a handwritten character recognition system based on hidden Markov models.

5.2.1 Feature Extraction

Each character example is preprocessed in the manner described in Section 3.2. We extract four different features at each sample point on the character: the change in the x coordinate from the previous point to the current point (Δx), the y coordinate of the current point, and the sine and cosine of the angle of incline, θ, for the line segment between two consecutive points. The characters are modeled using hidden Markov models with continuous Gaussian observation densities. In order to keep the number of model parameters that must be estimated from growing too large, an attempt was made to de-correlate the four features using principal component analysis [42], thereby allowing the use of diagonal covariance matrices to represent state distributions. In addition, the feature dimensionality can potentially be reduced by eliminating those dimensions in the projected space that correspond to small percentage of the overall variance (very small eigenvalues relative to the total sum of eigenvalues for the system). Best recognition results were obtained when the feature dimensionality was reduced from 4 to 3 dimensions. It should be noted that principal component analysis, when applied to the full set of all feature vectors, does not

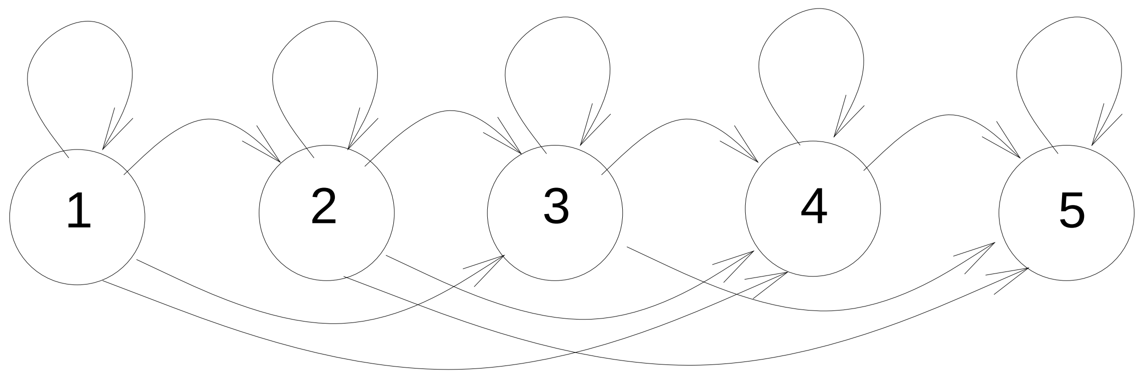


Figure 5.1: The topology of our hidden Markov models. Note that only single or double state skipping transitions are allowed.

guarantee that the features will be decorrelated for the subset of this data that is assigned to an individual state in the HMM.

5.2.2 Hidden Markov Models

In the simplest system, a single hidden Markov model is trained for each character class, using all the training data available for that class. These are left-to-right models in which each state contains a single continuous Gaussian distribution with a diagonal co-variance matrix to represent the feature observations for that state. In addition, each state allows for self-transitions and single-step transitions as well as transitions that involve skipping one or two consecutive states (see Figure 5.1). A test character is classified by identifying the model for which the probability of observing that character is the greatest. This approach can be enhanced by first categorizing the training data of each character class into different writing styles (lexemes), and training individual models for these lexemes. The individual strokes of multistroke characters are concatenated into a single string, and are therefore represented by a single model. This method has the advantage of being robust to noise during data collection that may break a single stroke into multiple strokes.

A character is then classified into class C_{Best} corresponding to the model which maximizes the aposteriori probability:

$$Best = \arg\max_{i \in L} P(\lambda_i|O), \tag{5.1}$$

where Bayes Formula defines:

$$P(\lambda_i|O) = \frac{P(O|\lambda_i)P(\lambda_i)}{P(O)} \tag{5.2}$$

in which O is a string of observations (feature vector) corresponding to the test character, λ_i is a HMM trained for lexeme i, and $P(O|\lambda_i)$ is the likelihood of O given that the correct model is λ_i. This likelihood is obtained using the Viterbi algorithm [30, 108]. Since $P(O)$ is constant over all λ_i, we can make the following simplification (for classification purposes):

$$P(\lambda_i|O) \approx P(O|\lambda_i)P(\lambda_i). \tag{5.3}$$

Furthermore, by assuming that the lexeme priors are equal, the decision rule can be reduced to

$$P(\lambda_i|O) \approx P(O|\lambda_i). \tag{5.4}$$

A character is then assigned to class C_{Best} with

$$Best = \arg\max_{i \in L} P(O|\lambda_i), \tag{5.5}$$

which is the class defined by the model, out of the set of lexeme models, L, which has the maximum likelihood of producing O.

5.3 HMM-based Clustering

The appropriateness of our string matching-based clustering method for identifying lexemes is not clear when our models, namely HMMs, are based on a different representation. A more appropriate clustering method would use the same distance measure as is used in classification. Note that our model is a parameterized stochastic model, and therefore the distance measure is probabilistic. While techniques of fitting clusters using stochastic models are well known when applied to non-temporal Gaussian mixtures [67], this remains largely unexplored for temporal data.

In our approach, a method based on K-Means clustering was explored, proposed by Perrone and Connell [78]. A character is given the class of the hidden Markov model for which the Viterbi algorithm produces the largest likelihood. This probability will then be used to define the distance between a single character and a cluster of characters. Since this distance measure is not defined until clusters are created, we initialize the process by using our string matching-based clusters. Although this clustering does not occur in the same space as the hidden Markov models, there are similarities between the two representations in that both the string matching measure and HMMs are based on stretching and compression of the temporal data.

A hidden Markov model is trained for each cluster using patterns in that cluster as training data, forming lexeme models. Refinement then proceeds by reassigning each training example to the cluster whose lexeme model has the greatest probability of having produced that training example. The second stage is applied iteratively until there is no further refinement. The algorithm proceeds as follows:

<u>HMM-based Clustering</u>

N_L is the total number of lexemes for class C.

N is the total number of training patterns for class C.

L_i = Initial Cluster Membership for Cluster i, for all i, $1 \leq i \leq N_L$

$L_i^{NEW} = L_i$, for all i, $1 \leq i \leq N_L$

Loop until ($L_i = L_i^{NEW}$, for all i, $1 \leq i \leq N_L$)

    For each i, $1 \leq i \leq N_L$

        $L_i = L_i^{NEW}$

        $L_i^{NEW} = \{\emptyset\}$

        $\lambda_{L_i} = \text{HMM\_Train}(L_i)$

    For each i, $1 \leq i \leq N$

        $k = \arg\max_j P(O^i|\lambda_{L_j})$, for all j, $1 \leq j \leq N_L$

        $L_k^{NEW} \leftarrow O^i$

where L_i is the set of members of the ith cluster out of N_L total clusters from some character class C, and L_i^{NEW} for all i, $1 \leq i \leq N_L$ defines the new cluster memberships after each iteration of the while loop. λ_{L_i} is the set of trained parameters for a hidden Markov model to represent the cluster L_i, and HMM_Train(S) is the algorithm which trains these parameters using a set of training data, S. O^i is the observed string of events for the ith character of class C in the training set, and $P(O^i|\lambda_{L_j})$ is the probability that this character matches the model for cluster L_j and is obtained using the Viterbi algorithm. N is the total number of training examples from class C.

5.4 Results

An evaluation of our recognition system was conducted using three different datasets: a set of 12,291 handwritten digits, a set of 80,524 handwritten lowercase characters, and a set of 22,301 handwritten uppercase characters. These datasets were produced by a group of over 100 writers, and contains a combination of discretely written and handsegmented cursive characters. Each of these datasets was randomly divided into half training and half test data. Lexemes were identified within the training data using the string matching-based clustering method described in Chapter 3. Table 5.1 shows the recognition accuracies of our system obtained using a nearest neighbor classifier in which the full training set acts as templates, and three different levels of lexeme modeling: a template-based approach in which a single template per lexeme is used as the reference set in a nearest neighbor classifier, a single hidden Markov model per character class, and lexeme models built using data from lexeme identification. In

Table 5.1: Overall writer-independent results for three different classification problems.

Classifier	Digits		Lowercase Chars.		Uppercase Chars.	
	No. of Models	Acc.	No. of Models	Acc.	No. of Models	Acc.
String Matching Full Set	6148	97.74%	40,269	89.12%	11,158	84.29%
String Matching Reference Set	27	77.39%	84	50.11%	67	44.58%
Single Lexeme HMM	10	95.62%	26	84.43%	26	84.04%
Mult. Lexeme HMM	27	98.0%	84	86.85%	67	89.47%

the template-based approach, each reference template is selected as the character that is closest to the center of that lexeme's cluster. All lexemes are identified and models are trained using each of the three training sets: digits, lowercase, and uppercase, and then each classifier is tested using the corresponding test sets.

While the use of reference templates to represent the lexemes greatly reduces the computational requirements of the string matching measure, these templates do not represent the intra-class variance very well. On the other hand, hidden Markov models are well suited for this task and, in the case of both digits and uppercase characters, even outperformed string matching using the full set of training data as templates. In addition, these lexeme models significantly improve the accuracy over that obtained with a single model per character class. Overall, the best results of 98.0% recognition accuracy on digits, 86.85% accuracy on lowercase characters, and 89.47% accuracy on uppercase characters were obtained using a recognizer in which a hidden Markov model was trained for each lexeme. These models form a compact representation of the data, while obtaining similar recognition results to that of using the full training set as templates in our string matching nearest neighbor classifier. The throughput of the HMM classifier is approximately 7.2 characters per second for digits, 1.0 characters per second for lowercase characters, and 1.8 characters per second for uppercase characters. These timings are reported on a 296MHz Sun UltraSparc workstation.

5.4.1 Lexeme Identification by HMM-based Clustering

Perrone and Connell [78] integrated HMM-based clustering in the IBM Pen Technologies Group handwriting recognition engine [72] to identify lexemes. This is an unconstrained handwritten word recognizer, based on HMMs in which the state observations are represented with a mixture of Gaussian densities. These densities are taken from a pool of densities estimated over the entire feature space, making the representation slightly different than the Gaussian densities derived for our HMMs. These experiments showed a drop in error rate of as much as 8% within the first 2 iterations when tested on 4,920 words in unsegmented sentences taken from 188 different writers.

Based on the success of this work [78], the advantages of identifying lexemes in our continuous parameter HMM model recognizer through the use of HMM-based clustering was explored. Figure 5.2 shows the recognition accuracies obtained for the HMM-based clustering algorithm, using 40,269 training examples, and tested on an independent set of 40,255 lowercase handwritten characters from over 100 writers. Due to the computational requirements of this algorithm, experiments were limited

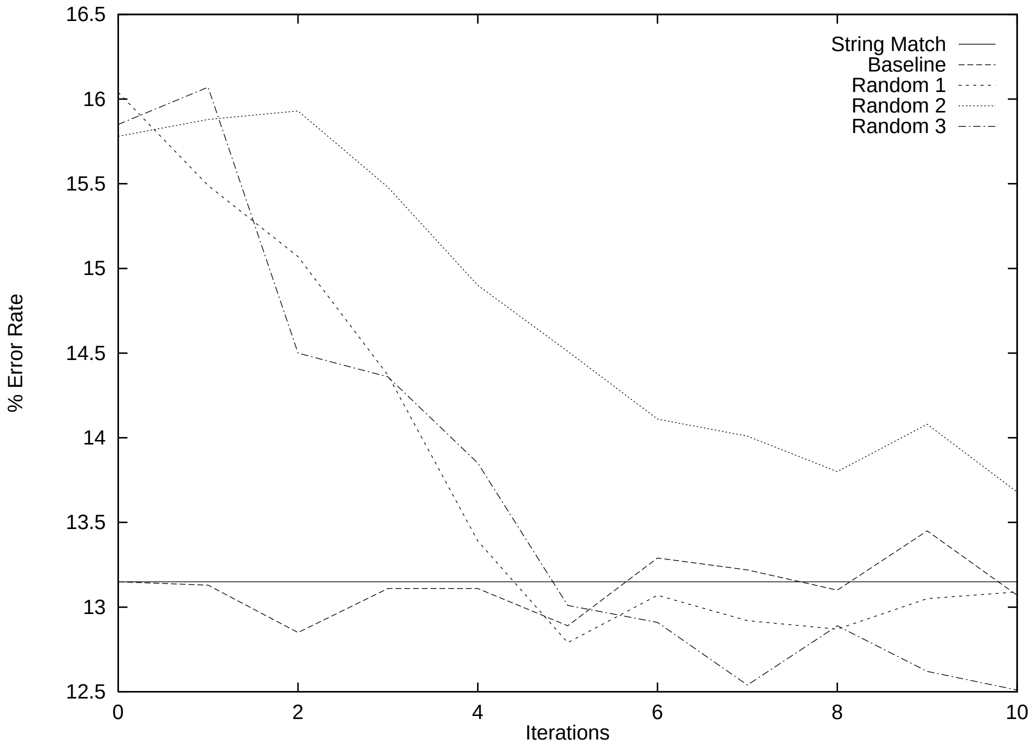


Figure 5.2: Performance of HMM-based clustering over 10 iterations. The "baseline" case shows the accuracies obtained at each iteration of the algorithm when the lexemes at *iteration 0* are defined using string matching as described in Section 3.4. The results on this initial set of clusters is also shown as a horizontal line (labeled "String Match") for comparative purposes. Three additional cases in which the lexemes at *iteration 0* were randomly assigned are shown as "Random 1", "Random 2", and "Random 3".

to 10 iterations. *Iteration 0* represents the accuracy of the initial set of clusters. For the example labeled as "baseline", these initial clusters have been defined in the string matching space as described in Section 3.4. As the figure shows, at *iteration 2* the best recognition accuracy for the "baseline" experiment has been achieved where there is a 2.25% reduction in error rate from the initial set of lexemes. Since the algorithm may potentially be limited by poor initial clusters (which could cause the algorithm to converge toward a local minima), three additional clustering experiments were run in which the initial set of clusters are defined randomly. For each of these experiments, the number of clusters was set to be the same as the "baseline" case, and the cluster membership assignments were uniformly distributed across the training data. As can be seen in Figure 5.2, the random initialization trials quickly converge toward our best results after a few iterations, and, in the case of the "Random 1" and "Random 3" trials, obtain a clustering that consistently outperforms the "baseline" case at each iteration. While the difference in error rate between these random trials and the "baseline" trial are not very large, it does indicate that the HMM-based clustering algorithm can be used to obtain a good clustering without relying on a sophisticated method of initializing the clusters. The best overall error rate achieved is 12.51% obtained at *iteration 10* of the "Random 3" trial. This is a 4.9% reduction in error rate compared to the error rate obtained using the string matching method of lexeme identification.

5.5 Summary

We have presented a handwriting character recognition system based on hidden Markov models. Such parametric models can represent the distribution of training data more compactly than non-parametric methods. The identification of lexemes can be used to decompose the complex feature space of handwritten characters into multiple, simple spaces, which can be represented with simpler hidden Markov models. The recognition accuracy of such a system has been shown to be superior to that modeled with a single HMM model per character class. In addition, we have proposed a method of refining these models, referred to as HMM-based clustering, in which the cluster membership is driven by the trained models themselves. This modification has been shown to reduce error rates by 4.9% on a lowercase character classification task.

Chapter 6

Information Fusion

The fusion of different information sources can be of great advantage to a recognition system, providing a greater classification accuracy than any classifier built upon a single source of information. In this chapter, the use of classifiers built from different feature sets is explored in which the individual feature sets attempt to capture some unique information that is not present in the other feature sets. A method of combining the outputs of these classifiers is discussed, and results are presented on the problem of alphanumeric character recognition. In Chapter 7, this method will then be used to improve the accuracy, while reducing the computational requirements, for word recognition.

6.1 Introduction

Information fusion is a field of study that has received much attention recently. When multiple sources of information exist regarding a decision making problem, the goal is to make the best decision possible given these different sources. This is a challenging problem since it requires handling potentially conflicting information, having an understanding of the uncertainties related to the sources of information, and possibly handling missing information. Classifier combination is a form of information fusion in which the sources of information are the outputs of a set of classifiers.

Much work has appeared in the literature on classifier combination [39, 46, 49, 53, 97]. The type of combination method used is often driven by the type of information that is available at the output of the component classifiers. This may be a single classification label, a ranked set of labels, or a set of posterior probabilities. The success of a combination of classifiers depends on their degree of “independence”. Therefore, component classifiers trained from different features sets and/or using different modeling techniques are desired. Methods of bagging [12] and boosting [31, 85, 97] train different classifiers from a single source of training data. Bagging accomplishes this by subsampling the data with replacement, while boosting trains a set of classifiers in series, such that classifier i focuses on the misclassifications of the combination of classifiers 1 though $i-1$.

6.2 Component Classifiers

Classifiers were constructed for each of the five different feature sets shown in Table 6.1. As a preprocessing step the location of each character was normalized by first detecting the baseline of a character and then subtracting the baseline height from the character height, such that at the y coordinate of each sample point is taken

Table 6.1: Characteristics of the different classifiers used.

Name	Local	Crit. Pt.	Spatial/Dir	End-point	COG
Classifier Type	HMM	HMM	NN	NN	NN
Feature Type	Dynamic	Dynamic	Static	Static	Static
Size Normalized?	Yes	No	Yes	Yes	Yes
Multistroke Model?	No	No	Yes	Yes	Yes

with respect to the baseline. In order to increase the reliability of baseline detection accuracy, the baseline was detected for an entire horizontal line of written characters.[1] This method is described in more detail in Section 7.2.1.

6.2.1 Local Features

These features provide information on the dynamics of writing at a very high resolution. They are the Δx, y, $\sin(\theta)$, and $\cos(\theta)$ features described in Section 5.2.1. While these features are good for representing very local structures in the handwritten strokes, they do not necessarily represent the global structural properties well.

6.2.2 Critical Point Features

The inability of the *Local* features to represent global structures has motivated a second dynamic feature set to focus on such structures. These features measure the curvature and orientation of stroke segments centered around a critical point, as well as the vertical location of the critical point. In addition, some local properties are measured within a neighborhood of the critical point. For each critical point, i, corresponding to an original sample point $S(i)$, the features extracted can be summarized

[1]This baseline detection was done at IBM previous to our processing of the data.

as follows:

1. The angle, θ, between two line segments formed by connecting critical point i to $i+1$ and i to $i-1$. The angle is measured such that $0 \leq \theta \leq 180$.

2. $\sin(\phi)$ and $cos(\phi)$, where ϕ is the angle between the x-axis and the direction in which the center of concavity of θ is pointing.

3. Δx and Δy measured from critical point $i-1$ to $i+1$.

4. The vertical location, y, of the critical point, i.

5. Δx and Δy for each consecutive pair of two points from sample point $S(i)-10$ to sample point $S(i)+10$.

6. $\sin(\psi)$ and $\cos(\psi)$ for each consecutive triple, $(k-1, k, k+1)$, of points from sample point $S(i)-10$ to sample point $S(i)+10$. ψ is calculated as the angle formed by the line segments k to $k-1$, and k to $k+1$. The angles are measured such that $0 \leq \psi \leq 180$.

The features are pictorially demonstrated in Figure 6.1. Together, these features form an 83-dimensional feature vector for each resampled point, which is reduced to 60 dimensions by Principal Component Analysis [42]. For the computation of these features, characters were not size normalized in the manner described in Section 3.2. This is intended to make this feature set suitable for word recognition when character segmentation is not known. Size normalization is instead accomplished for an entire horizontal line of written characters in order to simulate the type of size information that would be available for normalization if the character were to appear in a word. This method is described in more detail in Section 7.2.1.

6.2.3 Spatial/Directional Features

The previously described features make use of dynamic stroke information, but do not adequately capture the spatial relationships. Many offline character recognition features capture this type of information (see [20]), and therefore such features should compliment the information encoded in the online features. A set of features, motivated by the offline features described in Takahashi [102], have been designed for the recognition of online characters.

The character is size normalized such that it's bounding box is a standard size. For some characters, their small size and location can best be used to distinguish them from other characters, therefore we wish to preserve this information. This is accomplished by placing the following restrictions on the bounding box:

1. The bottom of the box cannot be higher than the baseline.

2. The top of the box cannot be lower than the midline.

3. The width of the box cannot be thinner than a minimum width which is one half the distance between the baseline and the midline. If this is the case, the box width is enlarged equally on both the left and right sides, such that it this minimum width.

The bounding box is divided into a 5×5, grid. Features are calculated independently for each of these grid sectors, thereby capturing spatial location information

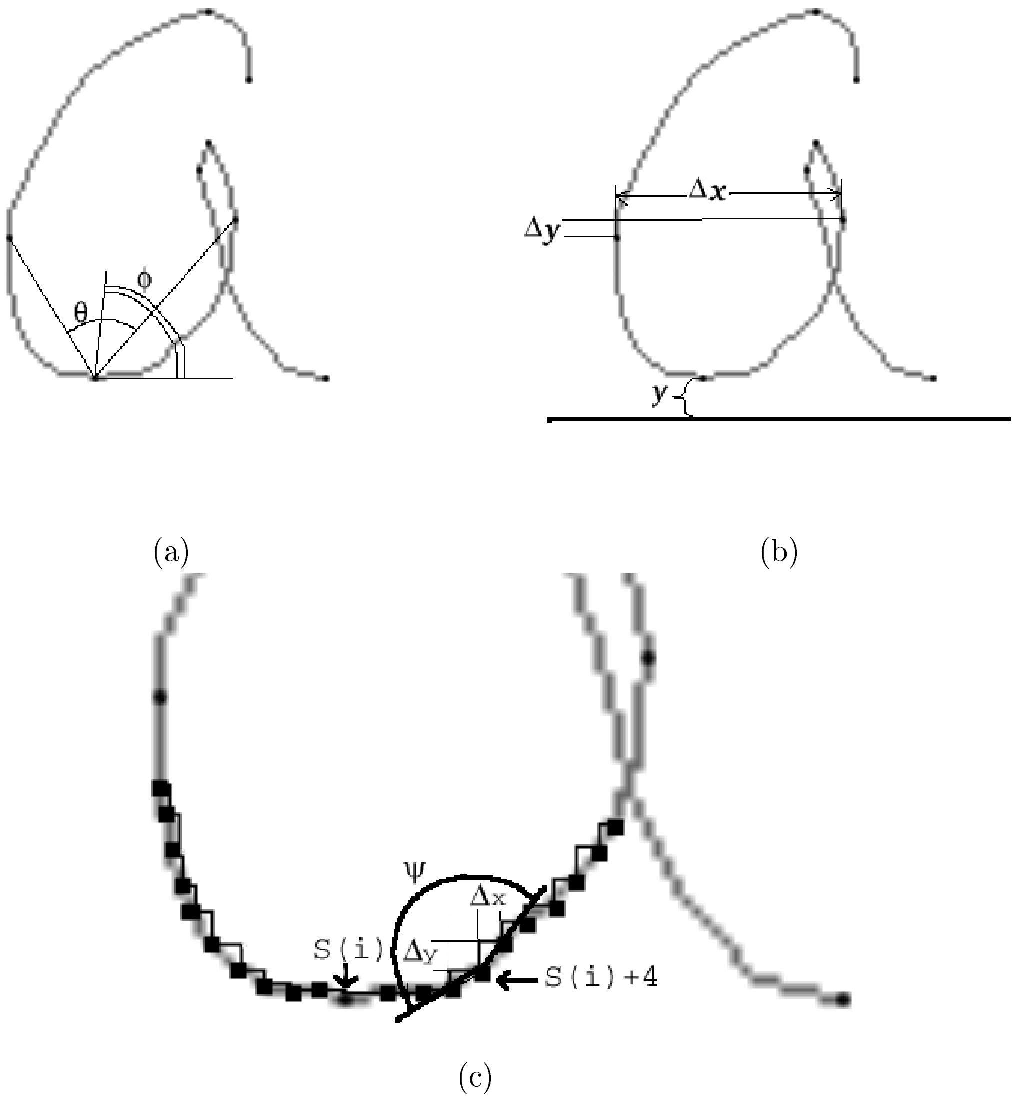


Figure 6.1: An example of *Critical Point* features: (a) calculation of structural features θ, and ϕ, (b) y (in relation to baseline shown), and critical point (Δx, Δy), (c) calculation of local neighborhood features Δx, Δy, and ψ.

about the properties measured. All strokes segments (from one sample point to the next) within a sector are quantized such that their direction is measured as either North-South, East-West, Northwest-Southeast, or Northeast-Southwest. For each of these directions, the total stroke length traveled in that direction is measured for each grid sector, forming a 100-dimensional feature vector (5×5 sectors $\times 4$ directions). Figure 6.2 shows an example of this feature extraction method.

One additional piece of information that can be made use of is the stroke segmentation. We extract a single 100-dimension feature vector for each stroke in the character, forming a $(100 \times N)$-dimensional vector, where N is the number of strokes forming the character. Classification is accomplished through the use of a nearest neighbor classifier, in which an N-stroke character is only compared against N-stroke templates.

Since a nearest neighbor classifier requires a number of comparisons that increases linearly with the training set size, a way of identifying a more compact representation is desired. Vector quantization [33] is a method of reducing a set of training vectors to a smaller set of representative vectors, with an emphasis on minimizing some distortion that is introduced through this representation. The OLVQ1 algorithm of the Learning Vector Quantizer (LVQ) package [55] is used to obtain these representative vectors. This is a self organizing map [54] which finds representative vectors by iteratively maximizing the nearest neighbor classification accuracy on the training set.

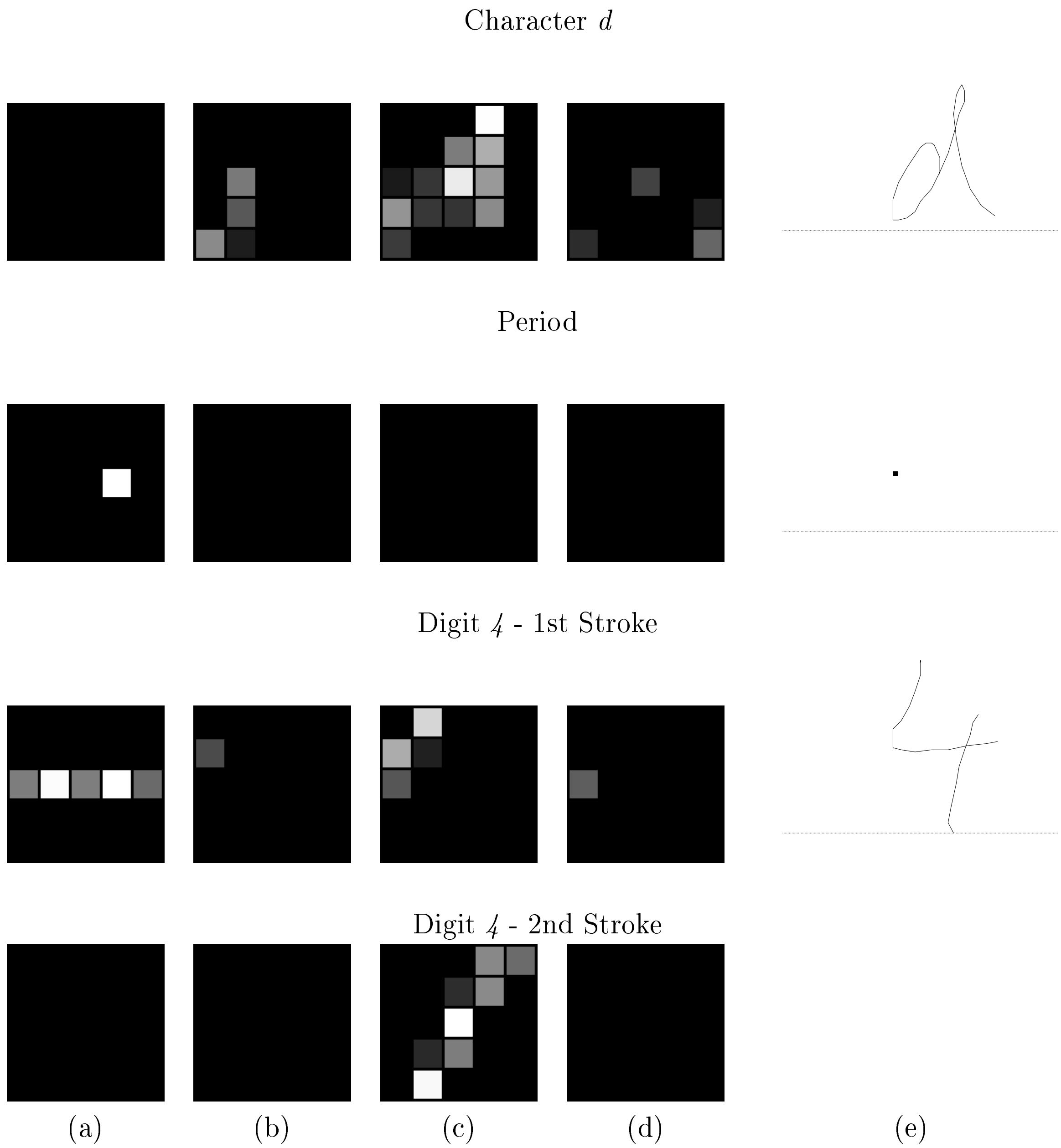


Figure 6.2: An example of spatial/directional features extracted from the single stroke character *d*, a ".", and the two-stroke digit *4*. Intensities indicate the magnitude of the features in each sector of the 5×5 grid in (a) horizontal direction, (b) vertical direction, (c) southwest-northeast diagonal, and (d) northwest-southeast diagonal. The original character is also shown (e) in relation to the baseline.

6.2.4 Stroke End-point Features

These are simple features which focus on the spatial relationship between the starting and ending points of each stroke. After size normalization, a pair of features in calculated for each stroke forming that character: the Δx and Δy from the starting point of the stroke to the ending point of the stroke. This forms a $(2 \times N)$-dimensional feature vector, where N is the number of strokes forming the character. Classification is accomplished through the use of a nearest neighbor classifier, in which an N-stroke character is only compared against N-stroke templates. LVQ is used to reduce the number of template comparisons required per classification, as described in Section 6.2.3.

6.2.5 Center of Gravity Features

Another simple large-scale structural characteristic is the location of each stroke in the character. This is encoded as the center of gravity of each stroke, (x, y). This forms a $(2 \times N)$-dimensional feature vector, where N is the number of strokes forming the character. A nearest neighbor classifier is created using a set of quantized vectors trained using LVQ as in Section 6.2.3. Characters are first size normalized.

6.3 Classifier Combination

Our classifier combination problem involves both HMM and nearest neighbor classifiers. Therefore, posterior probabilities are not available from all the classifiers, and at best they provide a set of ranked classification labels. Given a ranked list of N

class labels from a classifier, the possibility still exists of any class label being the true class label. We can express the probability that the actual class label is C_i as

$$P(C_i|Q_j(O)) = P(C_i|Q_{j,1}(O), Q_{j,2}(O), \ldots, Q_{j,N}(O)), \;\; 1 \leq i \leq N \tag{6.1}$$

where O is the observed pattern, and $Q_{j,k}(O)$ is the kth best class label in the ranking given by classifier j. Estimation of this posterior probability may not be feasible as it requires examples in which a classifier returns every possible ranking for each given truth value. Therefore, this probability can be simplified to

$$P(C_i|Q_j(O)) = P(C_i|Q_{j,1}(O), Q_{j,2}(O), \ldots, Q_{j,M}(O)), \tag{6.2}$$

for some value of M where $M \leq N$.

Under the assumption that the information, $Q_j(O)$, provided by different classifiers is statistically independent, the final classification is taken as the class C_i which maximizes the following posterior probability:

$$P(C_i|O) = \frac{\prod_{j=1}^{R} P(C_i|Q_j(O))}{[P(C_i)]^{R-1}}. \tag{6.3}$$

where R is the number of component classifiers involved in the combination. If we assume that these posterior probabilities do not deviate dramatically from the prior probabilities, it has been shown [53] that this can be expressed as

$$P(C_i|O) = P(C_i)\prod_{j=1}^{R}(1 + \delta_{ij}) \approx P(C_i) + P(C_i)\sum_{j=1}^{R}\delta_{ij}, \tag{6.4}$$

where $\delta_{ij} << 1$ is the deviation of the posterior probability, $P(C_i|Q_j(O))$, from the prior, $P(C_i)$. This can then be further simplified as

$$P(C_i|O) = (1 - R)P(C_i) + \sum_{j=1}^{R} P(C_i|Q_j(O)), \tag{6.5}$$

which under the assumption of equal priors, can be approximated by

$$P(C_i|O) = \sum_{j=1}^{R} P(C_i|Q_j(O)). \tag{6.6}$$

The summation form of this posterior probability is more robust to missing information than the product form, since the product form will force the posterior probability to go to zero if a single classifier's posterior probability is zero.

This classifier combination technique is similar to techniques based on confidence measures in that $P(C_i|Q_j(O) = C_i)$ is a measure of confidence in the ability of classifier j to correctly classify a character of class C_i. However, this technique also makes use of the confidence in the common misclassifications of each classifier. For example, $P(C_i|Q_j(O) = C_e)$ is the probability that the true class is C_i when classifier j claims, in error, that it is class C_e, $i \neq e$. If this error occurs frequently, this information will still contribute (albeit, probably not as much as a correct classification) toward the combined posterior probability of the correct class. The advantage of this is that consistent misclassifications by a single classifier can still contribute to the correct combined classification.

As a natural extension to this method, the classification $Q_j(O)$ returned by a classifier j can be in the form of a lexeme identity rather than a character class

Table 6.2: Top 10 classification results on 57,541 alphanumeric characters. Case-dependent (62 classes) results are reported.

Features	Accuracy (% within top N)									
	1	2	3	4	5	6	7	8	9	10
Local	76.45	91.66	95.61	97.01	97.80	98.32	98.66	98.90	99.10	99.23
Critical Point	68.81	82.32	87.67	90.60	92.34	93.70	94.67	95.39	96.01	96.49
Spatial / Directional	64.92	75.92	80.79	83.70	85.74	87.25	88.43	89.33	90.15	90.84
Stroke End-point	21.27	27.32	31.75	35.06	38.47	41.78	44.71	47.94	50.38	52.57
Center of Gravity	23.60	33.41	38.83	43.85	48.45	52.79	56.45	60.0	62.82	65.21

identity. In the following experiments, $Q_j(O)$ is the lexeme identity for the HMM classifiers, while it is the character class identity for the nearest neighbor classifiers.

6.4 Results

Classification results for each of the classifiers described in the previous section are shown in Tables 6.2 and 6.3. Each classifier was constructed for the task of distinguishing between 62 different characters classes: 26 lowercase letters, 26 uppercase letters, and 10 digits. The training set consists of a total of 57,575 characters randomly selected from a pool of data written by over 100 writers, and results are reported on an independent set of 57,541 characters randomly selected from this same pool. Results are also reported when the case (uppercase vs. lowercase) is ignored (e.g., a classification of 'e' or 'E' are both considered the same). In most cases, this can be considered a very minor confusion since it will not change the meaning of a word.

Table 6.3: Top 10 classification results on 57,541 alphanumeric characters. Case-independent (36 classes) results are reported.

Features	Accuracy (% within top N)									
	1	2	3	4	5	6	7	8	9	10
Local	84.32	94.02	96.48	97.63	98.25	98.68	98.96	99.16	99.32	99.41
Critical Point	71.81	84.96	89.96	92.51	94.03	95.21	95.99	96.57	97.08	97.47
Spatial / Directional	69.10	79.53	84.24	87.04	89.04	90.54	91.65	92.51	93.25	93.86
Stroke End-point	23.60	30.53	36.35	41.41	45.69	49.99	53.35	57.39	60.83	63.43
Center of Gravity	26.23	37.74	43.97	49.68	54.77	59.59	63.74	67.55	70.57	73.06

The best results using a single feature set are obtained using the *Local* feature set, which achieve a 76.45% accuracy for case-dependent classification, and a 84.32% accuracy for case-independent classification. Also of note are the *Critical Point* features, for which the correct character is found within the top 4 hypotheses over 90% of the time, and the *Spatial/Directional* features, for which the correct character is found within the top 9 hypotheses over 90% of the time. It is apparent (as would be expected) that the much simpler *Stroke End-point* and *Center of Gravity* features are not very useful by themselves, however their potential to add a new piece of information in a classifier combination can be considered.

Our classifier combination technique can be used to combine the component classifiers in any combination. Table 6.4 presents the classification accuracies for each of these possible combinations. For these experiments, $M = 1$ such that classification is based on only the single best classification from each classifier. It should be noted that this technique can also be used for a component classifier, and has the potential to increase the accuracy of that component classifier by focusing on its common misclassifications. These results are also shown in Table 6.4, and as can be seen, the component classifier accuracies are improved for all the classifiers with the exception of the classifier based on Stroke End-point features.

The single best combination of two classifiers, is the combination of the *Local* features and the *Critical Point* features. While the *Center of Gravity* features did not produce an impressive classification accuracy when used alone, when added to any other combination of classifiers, it results in an increase in accuracy (except when combined with the other two offline features, in which case the accuracy remains the same). This indicates that these features may capture some information not present in the other features. The same is true of the *Stroke End-point* features when combined with any of the online features. However, the results are mixed when combined with other offline features, possibly indicating a reinforcement in the ambiguities of these classifiers. The best results are obtained when the two online classifiers are combined with the *Spatial/Directional* features and one or both of the other two offline classifiers, producing an overall recognition accuracy of 84.0%.

Table 6.5 presents the case-dependent accuracies within the top N choices ($1 \leq N \leq 10$) for the five best classifier combinations shown in Table 6.4. It is apparent from this table that when more than two choices are to be considered, the much simpler combination of the *Local*, *Critical Point*, and *Spatial/Directional* features performs the best. For this combined classifier, the correct classification falls within the top 10 choices approximately 99% of the time.

Table 6.4: Classifier combination accuracies on 57,541 alphanumeric characters. Case-dependent (62 classes) results are reported.

Local	Crit. Pt.	Spatial/Dir	End-point	COG	Accuracy
X					77.9%
	X				69.7%
		X			64.9%
			X		22.1%
				X	25.1%
X	X				80.2%
X		X			78.4%
X			X		78.0%
X				X	79.0%
	X	X			73.8%
	X		X		69.9%
	X			X	70.2%
		X	X		64.7%
		X		X	64.8%
			X	X	29.0%
X	X	X			83.7%
X	X		X		81.1%
X	X			X	81.2%
X		X	X		79.0%
X		X		X	79.0%
X			X	X	78.6%
	X	X	X		74.7%
	X	X		X	75.1%
	X		X	X	70.7%
		X	X	X	64.7%
	X	X	X	X	75.6%
X		X	X	X	79.4%
X	X		X	X	81.8%
X	X	X		X	84.0%
X	X	X	X		84.0%
X	X	X	X	X	84.0%

Table 6.5: Accuracies within top 10 choices for the 5 best classifier combinations.

N	Classifiers				
	Local, Crit. Pt., Spatial/Dir, End-points, COG.	Local, Crit. Pt., Spatial/Dir, End-points.	Local, Crit. Pt., Spatial/Dir, COG.	Local, Crit. Pt., Spatial/Dir.	Local, Crit. Pt., COG, End-points.
1	84.02	83.96	83.97	83.72	81.75
2	91.85	91.65	91.73	91.57	91.11
3	95.02	95.03	95.10	95.22	94.13
4	96.42	96.54	96.54	96.89	95.59
5	97.19	97.29	97.31	97.54	96.35
6	97.61	97.65	97.73	97.96	96.87
7	97.92	97.96	98.01	98.30	97.25
8	98.14	98.24	98.25	98.54	97.61
9	98.31	98.48	98.47	98.75	97.88
10	98.51	98.69	98.59	98.92	98.04

The form of our classifier combination has the potential to make use of more than just the top choice from each classifier (the case for $M = 1$). However, as was mentioned in Section 6.3, it is not clear that incorporating knowledge for $M > 1$ is practical due to the rapidly increasing data requirements to train these probabilities as M increases. Table 6.6 presents the results of a classifier combination where $M = 2$, for one of our best classifier combinations, and each of the classifiers in this combination scored independently with this method. An examination of these results indicates that there is no increase in accuracy by the inclusion of the second best choice for either the *Critical Point* or *Spatial/Directional* classifiers (in fact, there is a reduction in accuracy). However, the *Local* features classifier obtains an increase in accuracy of approximately 1.5 percentage points. This improvement is also apparent in the combination of these three classifiers, which achieves an accuracy of

Table 6.6: Top 10 classification accuracies for classifier combinations with M=2.

N	Classifiers			
	Local	Crit. Pt.	Spatial/Dir	Local, Crit. Pt., Spatial/Dir.
1	78.43%	68.67%	64.12%	84.89%
2	89.80%	78.83%	76.12%	92.71%
3	91.98%	81.75%	80.71%	95.72%
4	92.65%	82.96%	82.98%	96.96%
5	92.99%	83.72%	84.62%	97.52%
6	93.17%	84.19%	85.82%	97.89%
7	93.27%	84.52%	86.76%	98.14%
8	93.32%	84.82%	87.44%	98.33%
9	93.36%	85.07%	87.93%	98.46%
10	93.40%	85.26%	88.41%	98.54%

approximately 85%. In addition, the correct classification is found approximately 97% of the time in the top 4 choices.

6.5 Summary

In this chapter, a method of information fusion through the combination of multiple classifiers built on different feature sets has been presented. These feature sets have been designed such that they focus on different information about the characters, such as viewing the character from an online vs. an offline perspective, the scale of structures that the features attempt to represent, the degree to which the characteristics captured are spatially dependent, and the relationship between different high-level structures such as the center of gravity of strokes and the distance between the endpoints of a stroke. Our classifier combination method focuses on the frequency of confusions for each classifier, and it has been demonstrated that this creates a potential to improve the accuracy of even a component classifier. The ability of this method to allow classifiers to contribute to the combined classification, weighted by their certainties of each possible class, facilitates improvements in accuracy even for weak classifiers, such as those based on the *stroke end-point*, and *center of gravity* features, as long as they capture some useful information not captured by the other features in use. The best combined classifier accuracy achieves approximately 84.0% for the 62-class alphanumeric character recognition problem (26 lowercase, 26 uppercase, and 10 digits). This is a 32% reduction in error as compared to the best single classifier accuracy. In addition, it was shown that the correct classification among the top 10 choices in approximately 99% of cases. This was further improved by taking into account the top 2 choices from each classifier in the combination, to achieve an accuracy of approximately 85% with the correct class among the top 4 choices approximately 97% of the time.

Chapter 7

Word Recognition

7.1 Introduction

The main goal of handwriting recognition is to correctly recognize individual words selected from a large lexicon. While it is not the goal of this thesis to produce a commercial grade word recognition system, it is of great interest to demonstrate the effects of lexeme modeling, lexeme-based writer-adaptation, and information fusion when applied to the word recognition problem. To this end, a word-recognition system has been constructed that is capable of recognizing words from a small vocabulary, and results are reported for 10 writers. The complexity of the inter-writer variations, however, are apparent in this data, and the challenge is to robustly handle such variations. Section 7.2 presents an overview of the word recognition system, including a description of the preprocessing steps, the models used, and methods of reducing the search space. Characteristics of the word-level data used for these experiments are discussed in Section 7.3, and the experimental results are presented in Section 7.4.

7.2 System Overview

Figure 7.1 presents an overview of the word recognition system. Each handwritten item is presented to the system as a single horizontal line of pre-segmented words. In a sentence-level recognizer, word-level segmentation can be achieved by way of spatial clustering of the handwritten strokes for well separated words, and by extension of the single-word recognizer to handle word pairs, for pairs of words that are not well separated. However, for this study word segmentation is assumed to be complete and accurate.

7.2.1 Preprocessing

This section describes the preprocessing steps taken that are specifically for word recognition. Other preprocessing steps are described in Section 3.2.

Certain preprocessing steps are first applied to the entire line of writing before recognition is attempted for each individual word in the line. In particular, these are the midline and baseline detection, and slant detection. Since short (possibly single character length) words do not give a very robust estimation of these values, they are estimated for the entire line of writing and then individual words are normalized using these values.

Baseline/Midline Detection and Normalization

Normalization for the size and location of words presents a more difficult challenge than for individual characters. Since character models are used for recognition, we

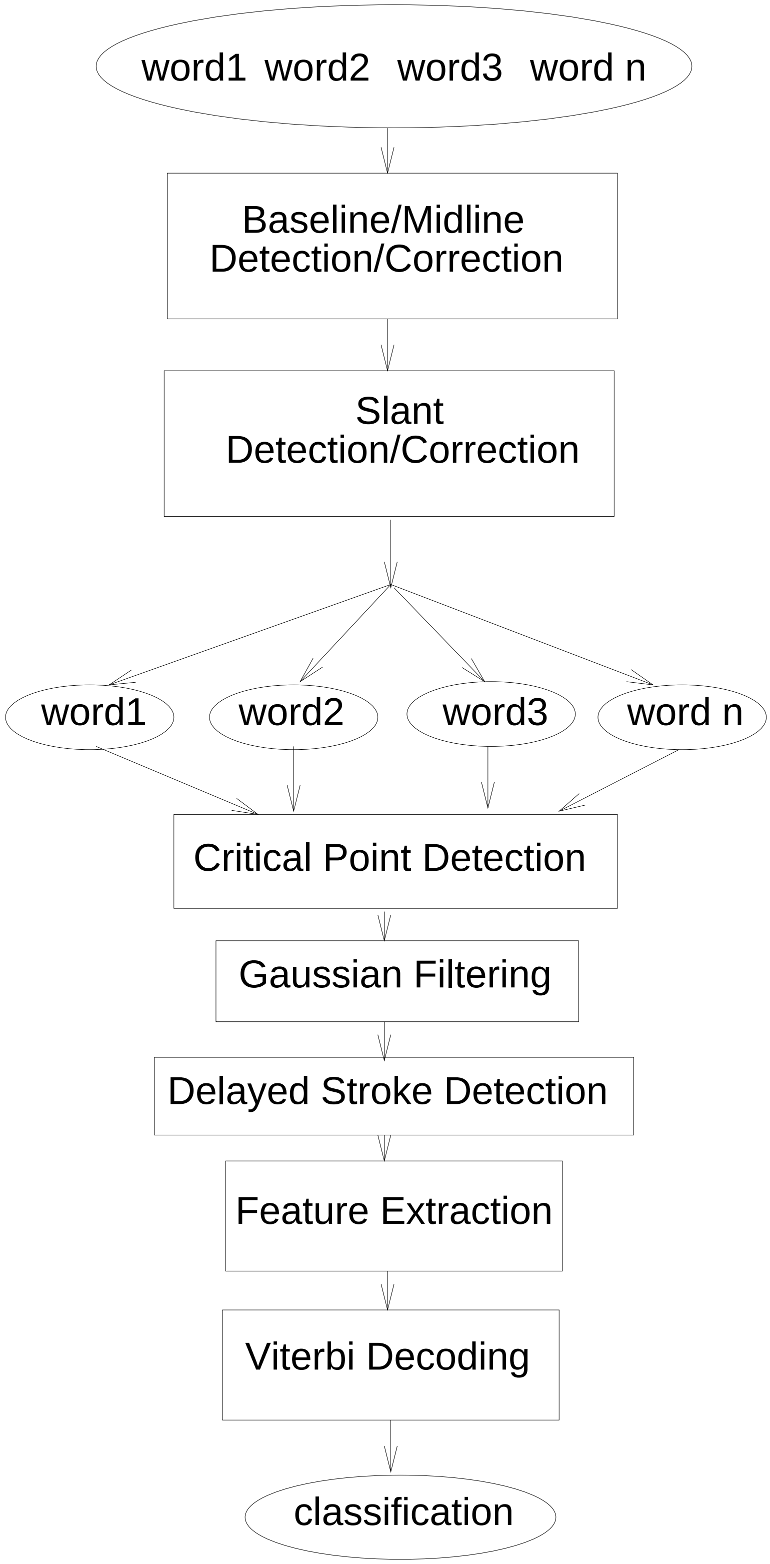


Figure 7.1: Word recognition system.

must ensure that the word is normalized such that it's individual characters are at the same scale and position with respect to some reference as those used to train the character models. However, without any knowledge of the character segmentation, simple character-based size normalization is not possible. In addition, it may be desirable to retain some of the scale and position information with respect to the word as a whole, since this can be useful information for indicating ascenders and descenders, or for indicating the difference between a comma and an apostrophe. We therefore strive to normalize the word as a whole based on the baseline (the line on which the writing "sits" and below which the descenders extend) and the midline (the line which separates the vertically middle region of writing from the ascender region).

A common offline method is used to detect the baseline and midline [7]. The written strokes are viewed as an image, and horizontal projections are taken forming a histogram as shown in Figure 7.2. The largest amount of ink should occur in the middle section of writing (the space between the baseline and midline), and therefore there should be a large hill in the horizontal projection profile at this point, with small tails above and below indicating the ink of ascenders and descenders. This peak is detected as the center of gravity of a smoothed version of the profile. The midline and baseline heights are then, respectively, adjusted upward and downward until they have each covered 20% of the total area of this smoothed profile. The locations of the baseline and midline are then refined as follows:

1. The mean height of the smoothed profile (referred to as the *peak mean*) found between the current midline and baseline is calculated.

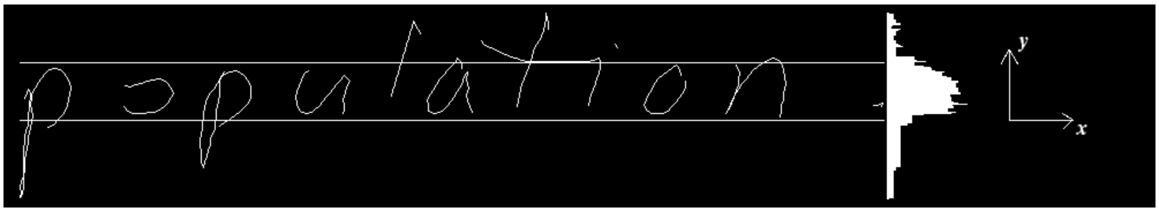


Figure 7.2: An example of baseline/midline detection.

2. The baseline is moved downward until the height of the smoothed profile falls below a threshold percentage of the peak mean.

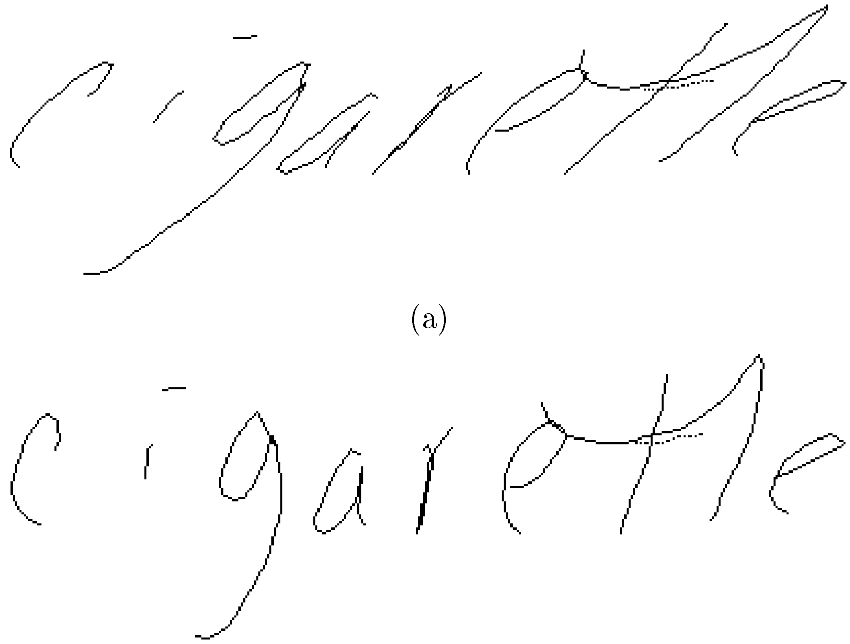
3. The midline is moved upward until the height of the smoothed profile falls below a threshold percentage of the peak mean.

Figure 7.2 shows an example of the detection of the baseline and the midline of a handwritten word.

Following baseline and midline detection, the word is normalized such that the distance between the baseline and midline is set to a standard height, and the word is translated such that the baseline is at $y = 0$ in the coordinate system.

Slant Detection and Correction

The amount of slant in handwritten characters often varies from person to person, and may even vary for a single writer (e.g., possibly dependent on the speed of writing). This is a variation that can be removed by detecting the amount of slant and then correcting for it. Slant detection is accomplished by measuring the average angle of incline of vertically oriented strokes in the writing. This is much easier to measure in online handwriting then offline since the online strokes can be viewed as offline strokes



(a)

(b)

Figure 7.3: Slant detection and correction (a) before slant correction and (b) after slant correction.

that have been perfectly segmented from the background, and perfectly thinned to a single pixel thickness. The average incline is calculated by measuring the total distance traveled and incline of each stroke segment with an incline between -157.5 and 157.5 degrees from vertical orientation. This average incline is then used to deslant the writing. Figure 7.3 shows an example of slant correction.

Delayed Stroke Detection

Delayed strokes create a difficult problem when attempting to recognize a word. Given a known delayed stroke in a word, and not knowing any of the character identities in the word, the delayed stroke creates a large search space in which we must attempt to match this stroke with any other single stroke, or sequence of consecutive strokes to

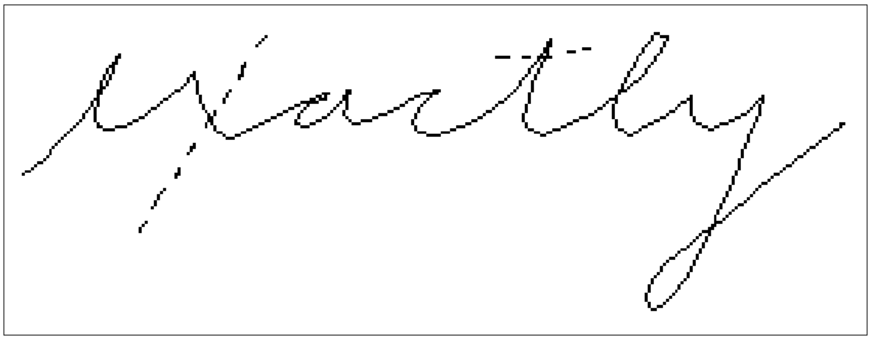


Figure 7.4: An example of delayed stroke detection. Strokes that have been identified as potentially delayed are shown as dotted lines.

form a candidate character, regardless of it's order in the stroke sequence. The number of possible options that must be searched when allowing any number of strokes in a word to be considered as a delayed stroke becomes extremely large. Therefore, we can greatly reduce the search space by limiting the strokes that are considered as delayed strokes by some initial pruning method. This is accomplished by marking strokes as potentially delayed if they extend leftward past the rightmost extension of any earlier stroke. Figure 7.4 shows a word example with the potentially delayed strokes shown as dotted lines.

7.2.2 Modeling

The proposed word-level recognition system makes use of character models trained from a combination of discrete printed characters and handsegmented cursive characters. Since a word is presented to the system without specifying an explicit character segmentation, we have three options:

1. Segment the word into individual characters for subsequent character recognition.

2. Construct a model for each word in the lexicon.

3. Perform character segmentation and recognition concurrently.

Reliable segmentation of a word into individual character components is a very difficult problem, and failure to provide reasonable segmentation options to the subsequent classifier will almost certainly ruin any chance of a correct character classification, and therefore a correct word classification. Construction of a model for each word in the lexicon becomes infeasible as the size of the lexicon grows. The data collection process would be enormous if multiple examples of each word are to be collected from a large number of writers. In addition, the storage requirement for these models would grow linearly with the size of the lexicon (e.g., a 20,000 word lexicon would require a minimum of 20,000 different models to be constructed). The third option, concurrent segmentation and recognition, can be seen as a case of the first method, segmentation followed by recognition, in which all possible segmentations are attempted. The computational requirements of this method are much greater, however.

Hidden Markov models are nicely suited for concurrent segmentation and recognition. By chaining character models together, an HMM word model can be defined. Through the use of a grammar, we can describe a graph in which each node is a character HMM, and a single start and single end node is defined such that the sequence of character models on any path through the graph from the start node to the end node defines a word in the lexicon (see Figure 7.5). Since the Viterbi algorithm finds the best path through an HMM (i.e., the path with the highest likelihood), this graph

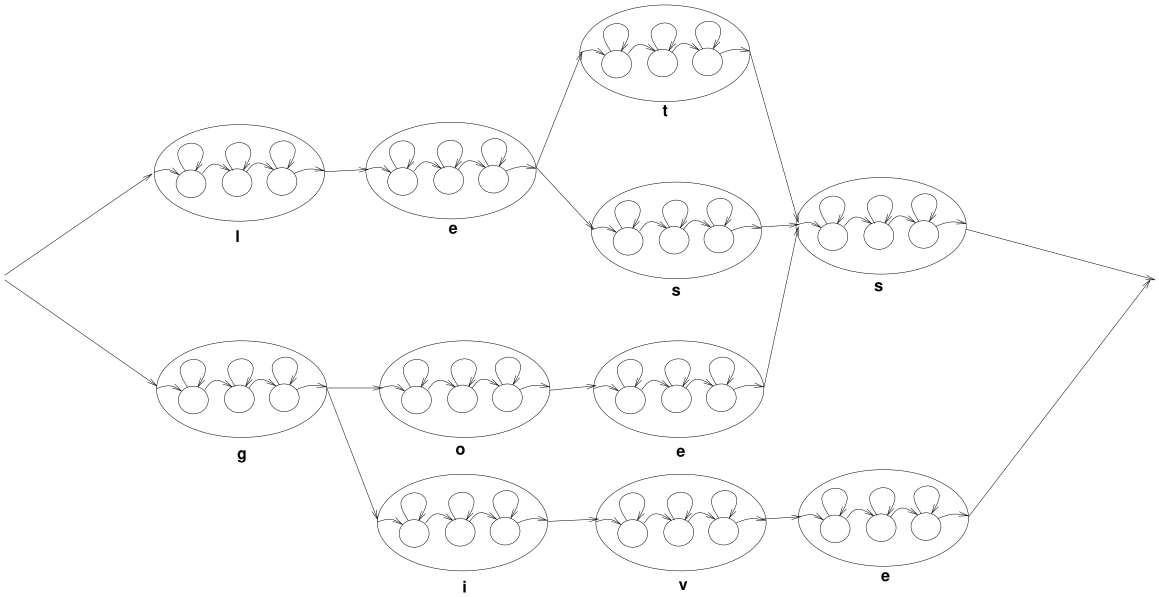


Figure 7.5: Example of word modeling by connecting character models. This grammar has the ability to recognize the words *lets*, *less*, *goes*, and *give*. Note that each of the models shown from a single class (e.g., all the "s" models) are actually only stored as a single model.

can be treated as a single HMM and the best path can be backtraced such that the sequence of character models followed determines the word classification.

In order to best handle segmentation of strokes in multistroke characters, and delayed strokes, our word recognition technique makes use of character stroke models in which a single model is trained for each stroke of each lexeme. This allows for restrictions to be placed on the alignment of strokes to models:

- The start/end of a stroke must be aligned with a transition from one stroke model to the next. While this does not allow for reconnecting strokes that have been broken into two or more pieces, it was found that, at the word level, the benefits of this endpoint alignment outweighed any losses due to this simplification.

- In general, stroke model transitions need not be aligned to the start/end of a stroke, if the stroke is the first stroke in a character. This allows for characters to be connected as in cursive writing.
- Certain stroke models must always be aligned with the start/end of a stroke. For example, the dot of an 'i' or 'j' must be a single stroke which cannot span multiple stroke models (i.e., it is never connected, even in cursive writing).
- All digit and symbol models must start/end on stroke boundaries. In other words, they cannot be connected to another character.

With the separation of character lexeme models into stroke models, it also makes sense to sub-divide the lexeme definitions based on the stroke counts in each training pattern. For example, if a particular lexeme contains characters formed by a single stroke, and characters formed by two strokes, these examples should be separated and a model should be trained to represent each of these.

7.2.3 Feature Set

Due to the large number of paths that must be searched in word recognition, the computational requirements for classification can easily become very large. A common method of reducing computational requirements is by multi-stage classification, using a simpler and less accurate classifier to reduce the search space to a number of more probable choices, and then rescoring these choices with a more complex and accurate classifier. This method is known as *cascading* [62, 68, 72]. Classification time with hidden Markov models grows rapidly as the number of states and the number of sample points grow. Therefore, the *Critical Point* features described in Section 6.2.2, which contain a single sample point for each critical point, are used to perform an initial recognition and character segmentation.

Although not implemented in this study, it should be noted that once classification is accomplished using these simple features, the character segmentation can be "inferred" by tracing back through the best path as defined by the Viterbi algorithm during word classification. For example, in the case of a two-character word, the time t in the temporal sequence of features $(O_1, O_2, \ldots, O_t, \ldots, O_T)$ in which the path makes a transition from the first character model at time t to the next character model at time $t+1$ defines the segmentation of this sequence into the sequence that makes up the first character, $(O_1, \ldots, O_t)$, and the sequence that makes up the second character, $(O_{t+1}, \ldots, O_T)$. Given this segmentation, we can now rescore the characters of this word by a more reliable technique, such as the *Local* features described in Section 5.2.1. By producing an ordered list of the top N word classifications using the *Critical Point* features, we can then select the best choice among these words after rescoring. The choice of N should be sufficiently large such that the correct word label is included a large percentage of the time. However, N should be small enough so as to minimize the number of computationally expensive *Local* feature matchings.

7.2.4 Delayed Stroke Handling

The handling of delayed strokes can easily cause the search space to increase far beyond reasonable boundaries. Even when candidate delayed strokes have been identified apriori, a large number of positions in which a delayed stroke can be inserted may exist in the word, each doubling the number of search paths at that point as a path with and without the delayed stroke inserted must be searched. In our method, when the next model in a search path is of a stroke that can occur as a delayed stroke (e.g., dot of *i*, cross of *t*, etc.), we must create a new path that branches off of the current path, skips this state, and marks it to be matched to a delayed stroke that should be written later in the sequence. In order to keep the number of paths that must be searched from growing too large, a number of restrictions are placed on the potential insertion location of these delayed strokes:

- Only certain characters are permitted to have delayed strokes. Currently, these characters are 'T', 'X', 'i', 'j', 't', and , 'x'.

- Only strokes other than the first stroke of a character are permitted to be delayed strokes.

- A delayed stroke for a particular character can only be inserted at a point immediately after the last non-delayed stroke of the character.

- In a search path, if a character is encountered that may contain a delayed stroke, there must exist a number of delayed strokes later in the sequence greater than the number of characters expecting delayed strokes currently in this search path, and one of those delayed strokes must extend within a certain horizontal distance of the non-delayed portion of the character.

Search Path Pruning

During Viterbi decoding of the best path through the graph of models, our method actually postpones the classification of the delayed stroke until it is encountered in it's natural temporal order. This allows for a fair comparison of all partially completed search paths for path pruning. Currently, during the Viterbi search, paths are pruned for the following reasons:

- The difference between the log likelihood of that path and the best path falls below a predefined threshold.
- The aspect ratio of a character, at the point of transition from one character model to the next, is not within 3 standard deviations of the mean for that character (or lexeme) class.

7.3 Test Data Set

Handwritten sentences were collected. Segmentation points for individual words were defined within each line through a process of model alignment using the truth values for the entire line of writing. These segmentations were then hand-verified and corrected. Detection of baseline, midline, and slant was then done for the entire line of writing, rather than for each individual word, to give a more robust estimate.

Experiments were conducted on a set of 8 writers, each writing between 571 - 614 words from a lexicon of 483 different words (see Table 7.1) containing upper and lowercase characters, digits, and symbols. These writers were not included in the training data and were instructed to write in their natural form. As a result, the 8 writers represent a large mix of cursive, discretely handprinted, and connected handprint characters in a variety of styles. Examples of three words from 4 different writers are shown in Figure 7.6 to demonstrate the degree of variation in the writing.

An examination of Figure 7.6 reveals some interested characteristics of the different writing styles. Writers 2 and 4 write primarily in a connected/cursive style, while writers 3 and 5 write in a discrete/printed style. Writer 1 uses a mostly printed style, but occasionally makes use of a connected style as can be seen in the word "than". Another characteristic worth noting is the dots (delayed strokes for letter 'i') written by writer 5. These large circular dots appear throughout this writer's data, and there exists no such examples for the other 7 writers. In fact, this type of dot is not well represented in the current set of lexemes identified during training, creating a potential difficulty when performing writer-independent recognition. Another characteristic of note is the cross of the 't' connecting to the following 'h' in the word "than" for writer 1. This is an example of the sort of care that must be taken when placing restrictions on the possible character connectivities so as not to eliminate commonly occurring possibilities from consideration.

Another handwriting characteristic is *overwriting*. Overwriting typically occurs when the writer is not satisfied with the quality of one or more of the characters he or she has written. The writer attempts to rectify this by writing the character again directly on top of the already existing character. Some examples of overwriting are shown in Figure 7.7. Overwriting is extremely difficult to deal with since it effectively means that any character may have a set of delayed strokes which are duplicates of the

Table 7.1: Words in the word classifier lexicon.

"Metro "nugget" #3 $1 $12 $2000 $4,300 $43,000+ $500 & '88. '92. (& (7.6 (By ∗36∗ - / 0.01% 0.26% 0.9%. 1% 1/4 10 100 12 13 14 150 1600 1969-70]. 1971 1975, 1980 1982-1992. 2% 20,000 22 23% 23%. 24 27 28%, 3.45% 34. 38% 38,146,000+ 4% 4.5 40% 42.3 45, 5%). 50% 60% 70% 72} 8 81% 9% 92% 94% < = > @ A Admit Advancing African Age Almost America America's American American! Americans Americans' Americans. An And Antarctica. Anytime Are Area" Arrests Asia; Average BROWN Bankruptcy Barbers Be Before? Below Between Birth Bread. Bush. By C:\~j6\^zq|, Can Chicken Choosing City Color Come Countries Cox, Current DOG. Daily, Debt. Did Divorced) Divorced). Do Dues-paying Each Education} Europe Europe. Exceeds Exhibit Expectancy Experience. FOX Fact: Federal Female Finds For From Gallons Gets Gov't. Graduates Greenland Growth Growth. Half Highest How I've Immigrants In Insurance Insurance. Is JUMPED Jan's Jogging John Just Keep Kept Know LAZY Land Last Led Less Life Lives Longest Male Males} Married May Median Medical Men! Men, Miami Mike's More Mr. Music Music. National Nationally, Nearby New News: Now. OVER Objectively Objectively, Occur Ocean Of On One Only Owned Park Per Person Population Prices Prisoners Projected QUICK Quarry Quarterly, Rates Recorded Reserves Seconds September? Shoppers Show Single Slowly, Smith Southern Southernmost Spend Spending States Statistics Student Sufficient THE Tenure; Than That The To Today, U.S. U.S.'s U.S.A. United Uses Vice Visits Volunteer Was Water Water. When While Xenon Year Year, Year. Yearly, Years' Years,] Years? York Zaire Zinc [In [Now] [down 'German,' 'strategic' a about above accounts added ads. adults adults? after ago. aims airplanes all-sports ancestry and annually. apple approx. are as average away be become billion blindness block break building-construction bulk bus-load by can, carbon children cigarette citizens claimed cm) coal come common companies consumption? conveyed countries? cut data. day declined decreased degrees, disk, documented don't economy edged education effectively employment energy? equals exactly exceed expects fallen falls fields. filings fiscal fleet for frequent from fuel. gal. gas goes government granted grew guy halved harvest's have helped home. households huge idled if ignite imports in in. include inert. info, invoke is isn't jet job jobs just know lady! law likely little logs many master's} memberships million mixed monoxide moral more most much music name? natural not now number of oil okay? on only ordered other outman over patents people people. percent performed phase population. possibly pounds. presently president presidents production provide public quickly radio rain ranks realize receive recently, restaurants revenue routine sales salt. she since sixteen sizable size slides small some spends stations. strikes successful support swiftly system tax) technically televisions ten than the their think thought three time; times times, to topmost total tourists tuna twenty typical union up up! value was water week. when which who why will with wood working world worse written year. years years, years. you {79, {Females, {Management {bachelor's

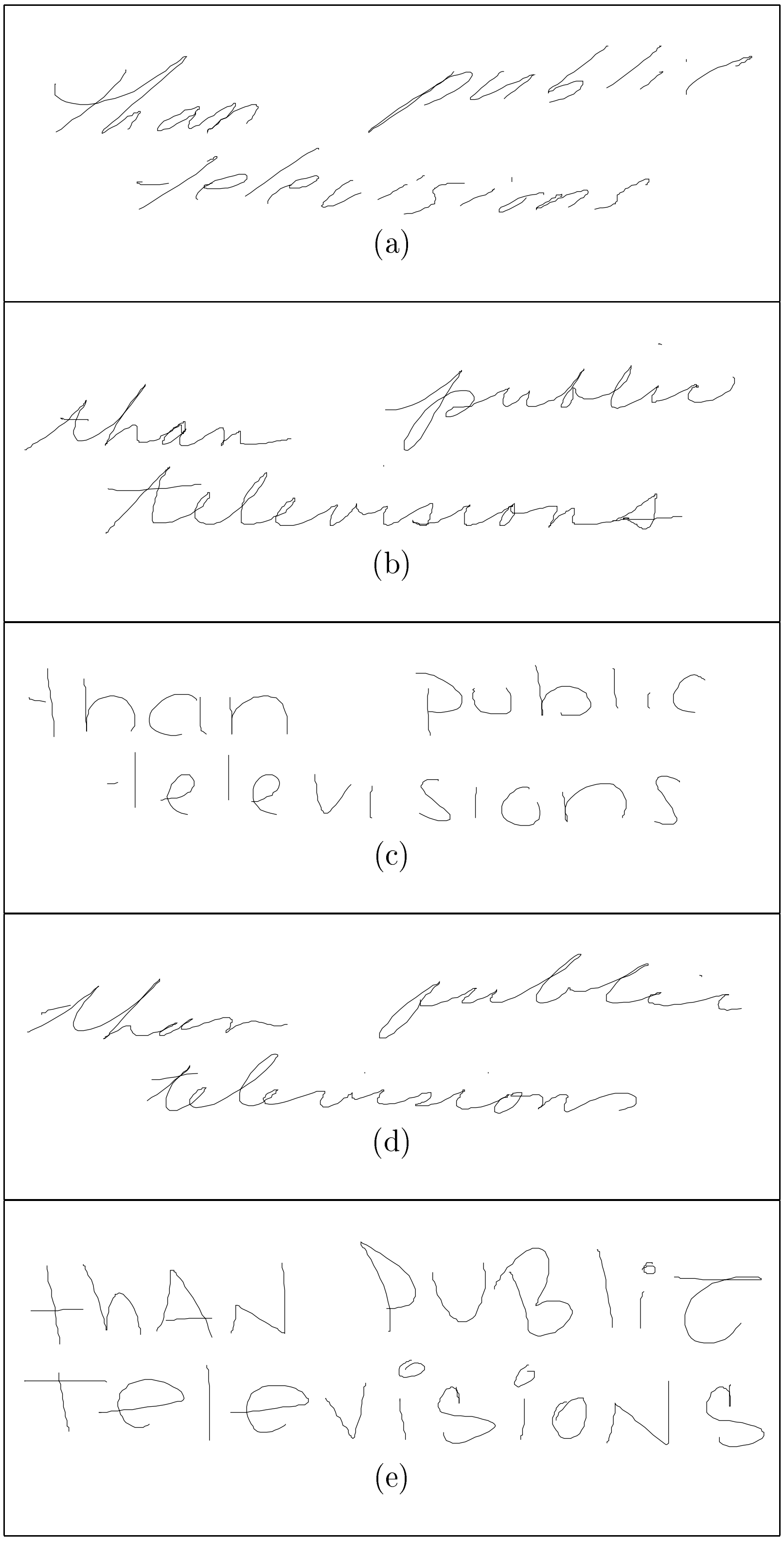


Figure 7.6: Variations between writers for handwritten words. (a) Writer 1, (b) Writer 2, (c) Writer 3, (d) Writer 4, and (e) Writer 5.

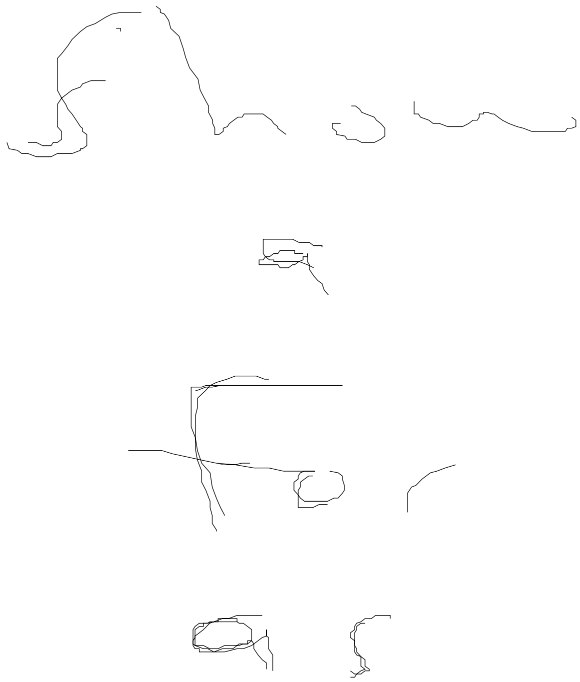


Figure 7.7: Some examples of handwritten words containing "Overwriting".

original strokes. Therefore, the search requirements would drastically increase. Currently, overwriting is considered an open problem, and we do not explicitly attempt to handle it in our classifier.

7.4 Results

For these experiments, a set of models were trained to recognize characters from 93 different classes: 26 lowercase characters, 26 uppercase characters, 10 digits, and 31 symbols. Lexemes were identified within these classes using a set of 122,410 characters containing both discretely written handprinted and cursive characters. Table 7.2 shows the amount of data available and the number of lexemes identified for each character class.

Table 7.2: The number of examples available and number of lexemes identified for each character class.

Class	Examps	Lex
A	1007	4
B	876	5
C	929	4
D	950	3
E	1020	5
F	833	6
G	828	5
H	759	3
I	824	7
J	733	3
K	793	4
L	830	3
M	858	8
N	878	6
O	923	3
P	848	2
Q	809	5
R	905	3
S	1055	3
T	887	4
U	775	4
V	742	3
W	849	3
X	809	5
Y	812	4
Z	769	2

Class	Examps	Lex
a	5013	7
b	2287	3
c	3811	3
d	3102	4
e	5930	4
f	2046	4
g	3037	5
h	2445	4
i	4515	8
j	1173	4
k	2088	5
l	3080	4
m	2865	3
n	4793	4
o	4136	4
p	2510	4
q	1619	5
r	4307	4
s	4126	4
t	4383	5
u	2984	4
v	2173	3
w	2156	5
x	1826	5
y	1985	3
z	2134	4

Class	Examps	Lex
!	682	2
”	287	3
#	86	3
$	93	2
%	91	5
&	96	5
’	683	2
(	181	2
)	195	2
	86	6
+	86	2
,	683	3
-	578	3
.	767	2
/	90	2
:	568	2
;	184	2
<	93	2
=	96	3
>	93	3
?	682	4
@	90	2
[	93	3
\	90	2
]	93	2
ˆ	90	2

Class	Examps	Lex
0	1217	4
1	1203	3
2	1210	2
3	1338	5
4	1333	4
5	1339	4
6	1348	5
7	1106	6
8	1105	4
9	1092	4

<table>
<tr><th>Class</th><th>Examps</th><th>Lex</th></tr>
<tr><td>‘</td><td>90</td><td>4</td></tr>
<tr><td>{</td><td>86</td><td>6</td></tr>
<tr><td>|</td><td>90</td><td>2</td></tr>
<tr><td>}</td><td>86</td><td>3</td></tr>
<tr><td>˜</td><td>86</td><td>3</td></tr>
</table>

Table 7.3: Word recognition accuracies using curvature/orientation features and a single model per character class.

Writer	No. Test	Accuracy	
	Examples	Case Sensitive	Case Insensitive
#1	571	57.1%	59.9%
#2	614	60.9%	62.7%
#3	577	41.9%	44.9%
#4	605	55.2%	59.8%
#5	610	57.2%	60.0%
#6	599	47.9%	49.4%
#7	609	34.3%	39.6%
#8	591	30.1%	32.1%
Total	4776	$\mu = 48.1\%$ $\sigma = 10.8\%$	$\mu = 51.1\%$ $\sigma = 10.6\%$

Table 7.3 shows the word recognition accuracies obtained using only the *Critical Point* features, and where only a single model is defined per character class (1 lexeme). Both case sensitive (e.g., an uppercase 'A' and a lowercase 'a' are treated as different character classes), and case insensitive accuracies (e.g., 'A' and 'a' are both treated as the same character class) are reported.

Table 7.4 shows the word recognition accuracies obtained, where the number of lexemes is as shown in Table 7.2. A comparison of Table 7.3 and Table 7.4 show a drop in average error rate of 32.0% for case-dependent word recognition, and 35.4% for case-independent word recognition. In addition, the recognition accuracies using multiple lexemes achieves a higher accuracy than the single lexeme classifier in the case of every writer with the exception of writer #2. The throughput of this system is approximately 0.34 words per second on a 296MHz Sun UltraSparc workstation.

As can be seen from these tables, there is a lot a variance in the recognition

Table 7.4: Word recognition accuracies using curvature/orientation features and multiple lexeme models per character class.

Writer	No. Test Examples	Accuracy	
		Case Sensitive	Case Insensitive
#1	571	79.9%	83.9%
#2	614	57.5%	58.5%
#3	577	72.3%	77.3%
#4	605	69.4%	75.2%
#5	610	84.1%	87.9%
#6	599	48.2%	49.4%
#7	609	51.7%	57.8%
#8	591	55.2%	58.2%
Total	4776	$\mu = 64.7\%$ $\sigma = 12.6\%$	$\mu = 68.4\%$ $\sigma = 13.3\%$

accuracies obtained between writers. Much of this can be attributed to the extent a writer's lexemes are represented by the set of writer-independent models in our system. Other reasons may be due to the current limitations of our system to decode certain characteristics. For example, the handwritten words produced by Writer #8 contain a very large amount of overwriting. Currently, there is no special facilities for handling this type of writing, and our system is doomed to fail for most of these words.

An examination of Table 7.4 reveals that there is a large variation in the recognition accuracy for individual writers. Our best results are for Writer #5, with an 84.1% accuracy for case-dependent word recognition, and our worst results are for Writer #6, for which this accuracy is 48.2%. Poor results such as this may be due to either a large class overlap in the lexemes used by this writer, or due to a lack of examples of the writer's lexemes in the writer-independent training set. If we choose to ignore differences in the case of the letters, these accuracies are increased by 3.7 percentage points on an average. For some writers, such as Writers #3, #4, and #7, this produces a significant increase in accuracy, indicating that the writer forms some uppercase or lowercase characters that are very similar to their complementary case equivalent. In addition, Writer #8, which was identified as having a large amount of overwriting, obtained a case-dependent recognition accuracy of only 55.2%. Figure 7.8 shows some examples of correctly classified and misclassified words.

In a word recognition system, it is also of interest to evaluate the quality of the alternative word identities that a word recognizer can produce. If the top word choice is incorrect, and the user wishes to correct this word in his document, it would be more convenient for the user to be able to select the correct word from a list of top choices rather than re-enter the word by pen or keyboard. In addition, the ability of the system to place the correct word among the top N word alternatives is crucial if we are attempting to improve the accuracy of the top choice through a cascading of classifiers, or the classifier combination techniques discussed in Chapter 6. Table 7.5 presents the percentage of cases in which the correct word identity is ranked among the top N choices by the *Critical Point* features, for $1 \leq N \leq 10$.

As can be seen in Table 7.5, the number of correct word classifications found in the top 10 best choices is much more promising. An interesting case is Writer #4, which jumps from an accuracy of 69.4% for the top choice, to 88.1% within the top 10. Our best results is for Writer #5, which achieves an accuracy of approximately 95% within the top 8 choices. Most writers which did not receive a good top choice accuracy (writers #2, #6, #7, and #8) still remain the worst performers when the

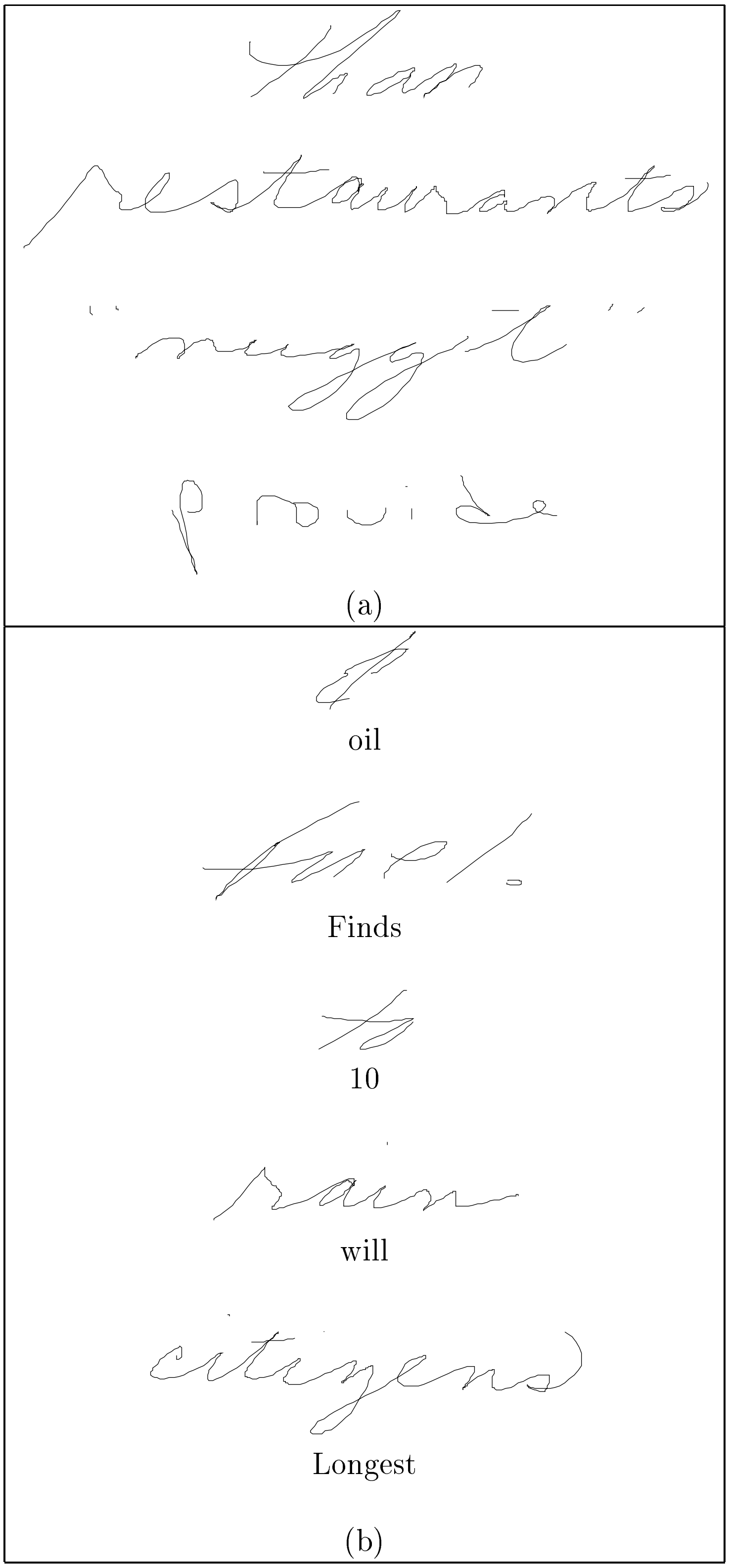


Figure 7.8: Some word recognition examples: (a) correctly classified words and (b) misclassified words and the incorrect label assigned by the classifier.

Table 7.5: Percentage of cases in which the correct word is within the top 10 choices for words from different writers. (The total number of words in our lexicon is 483).

Writer	Accuracy (%) Within Top N									
	1	2	3	4	5	6	7	8	9	10
#1	79.9	84.9	86.9	87.6	87.7	87.9	88.1	88.4	88.8	89.0
#2	57.5	62.7	66.0	68.2	69.4	70.5	71.8	73.0	73.9	74.4
#3	72.3	79.9	82.0	82.8	83.9	85.1	85.1	85.6	86.0	86.1
#4	69.4	79.8	83.5	84.5	86.3	86.8	87.3	87.4	87.6	88.1
#5	84.1	89.8	91.8	92.5	93.4	93.9	94.4	94.6	94.6	94.8
#6	48.2	58.6	60.1	61.8	62.6	63.8	64.9	65.8	67.3	67.8
#7	51.9	63.1	66.7	68.5	70.0	72.2	73.4	74.4	75.7	75.9
#8	55.2	63.3	68.2	72.1	73.6	75.1	76.1	77.0	77.3	77.7

top 10 choices are considered. This would indicate that there is some fundamental characteristics about their writing that are not being captured by our system. This may be due to either (a) certain system implementation restrictions, or (b) a lack of representation of the writer's lexemes in our writer-independent set of lexemes.

7.5 Summary

In this chapter, a system for the recognition of handwritten words has been described. This system makes use of character models that represent 93 different classes (10 digits, 26 uppercase and 26 lowercase characters, and 31 symbols), and a grammar which describes how these character models can be chained together to form the words in the lexicon. Results were presented for 8 writers tested on a writer-independent recognizer. The ability of this recognizer to capture the large variations in the handwritten characters was shown by the identification and modeling of lexemes in the writer-independent data. For these experiments, a coarse-level feature set, the *Crit-*

ical Point features, was used to provide the first stage classification and character segmentation due to their accuracy vs. complexity tradeoff. This achieved a case-dependent recognition accuracy of between 48.2% and 84.1% for an average of 64.7% on the 8 different writers. In addition, the percentage of time that the correct classification is within the top 10 choices indicates that a second stage classification using the *Local* features could greatly improve these results.

Chapter 8

Devanagari Script Recognition

The ability of our feature sets, lexeme identification method, and information fusion technique to generalize to other languages is explored in this chapter. Devanagari is a script used for several major languages such as Hindi, Sanskrit, Marathi and Nepali, and is used by more than 500 million people. Unconstrained Devanagari writing is more complex than English cursive handwriting due to the possible variations in the order, number, direction and shape of the constituent strokes. In addition, the large size of the Devanagari alphabet makes a keyboard interface difficult to use, thereby increasing the importance of alternative yet more natural computer interfaces. In this chapter, an experiment in the recognition of online Devanagari characters is presented, in which a combination of classifiers is used to obtain a good classification accuracy as described in Chapter 6.

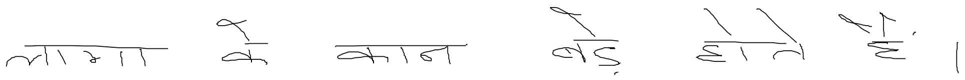


Figure 8.1: A sentence written in Devanagari.

8.1 Introduction

Although much work has been reported in the literature for off-line recognition of Devanagari script [5, 16, 17, 98, 99], to the best of our knowledge, this is the first time an on-line recognition of Devanagari has been attempted.

Devanagari script is a two-dimensional composition of symbols attached to the top, bottom, left or right of a main character. These composite characters are joined together by a horizontal line, called “Shirorekha” (see Figure 8.1). The Devanagari alphabet is based on approximately 93 symbols: 13 vowels, 36 consonants, 28 pure consonants (half-forms) and 16 modifier symbols. In addition, a number of conjunct forms exist in which a pure consonant is combined with a consonant to form a new character. Figure 8.2 shows forty major Devanagari characters. These form the basis of many other symbols in the alphabet. Since the script composition in Devanagari is two-dimensional and the number of symbols is large, its input using a keyboard is cumbersome. This makes the on-line pen computing environment very attractive for Devanagari input.

Text in Devanagari is similar to cursive English writing. A word needs to be segmented and decomposed into recognizable units. It usually consists of three vertical regions: a mid-region containing main characters, an upper modifier symbol region and a lower modifier symbol region. In this study, we have considered only the main characters that occur in the mid-region. An off-line recognition system usually makes use of a histogram to separate these strips as was accomplished for English writing in Section 7.2.1.

The representation of Devanagari characters as online data makes stroke segmentations explicit, which can be used to assist the process of segmenting a word into characters. However, the degree of variations in shape, order, and direction of Devanagari characters is very large. Compared to cursive English, Devanagari contains a larger number of strokes. In addition, the strokes are combined and broken based upon the writer's style of writing which is usually acquired at the time of learning the script. For example, the same stroke may be written from top-to-bottom/left-to-right or bottom-to-top/right-to-left by two different writers. Similarly, the Shirorekha may be broken and become part of down-going/up-going strokes. As an additional complication in the online stroke space, a part of the stroke may be retraced in the opposite direction. These variations are very common as can be seen from the sample on-line data (Figure 8.3). Some of these characters cannot be correctly recognized even by humans without using contextual information.

8.2 System Overview

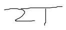
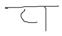
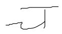
The Devanagari recognition task is accomplished through the use of a combination of 5 different classifiers, in which each classifier's error characteristics determine it's level of contribution to the final classification. This group of classifiers is made up of two hidden Markov model classifiers, which focus on two different levels of local features

अ	इ	उ	ऊ	ए
“a”	“i”	“u”	“uu”	“e”
(01)	(02)	(03)	(04)	(05)
क	ख	ग	घ	त्र
“ka”	“kha”	“ga”	“gha”	“tra”
(06)	(07)	(08)	(09)	(10)
च	छ	ज	झ	ञ
“cha”	“chha”	“ja”	“jha”	“ña”
(11)	(12)	(13)	(14)	(15)
ट	ठ	ड	ढ	ण
“Ta”	“Tha”	“Da”	“Dha”	“Na”
(16)	(17)	(18)	(19)	(20)
त	थ	द	ध	न
“ta”	“tha”	“da”	“dha”	“na”
(21)	(22)	(23)	(24)	(25)
प	फ	ब	भ	म
“pa”	“pha”	“ba”	“bha”	“ma”
(26)	(27)	(28)	(29)	(30)
य	र	ल	व	श
“ya”	“ra”	“la”	“va”	“sha”
(31)	(32)	(33)	(34)	(35)
ष	स	ह	क्ष	ज्ञ
“shha”	“sa”	“ha”	“kSha”	“jña”
(36)	(37)	(38)	(39)	(40)

Figure 8.2: Forty major Devanagari characters with English pronunciations and category numbers.

Figure 8.3: Some handwritten examples of the 40 different Devanagari characters.

Table 8.1: Poorly discriminated character pairs.

Character Pair	Distinguishing Characteristic
 (31)  (26)	top of upper left-most curve and curvature of lower left corner
 (31)  (11)	gap in vertical line on right side
 (31)  (11)	the almost non-existent vertical stroke in 31
 (30)  (08)	curve at top of lower left stroke
 (29)  (30)	circle at start of lower left stroke
 (24)  (09)	circle at start of lower left stroke
 (21)  (25)	circle at start of lower left stroke

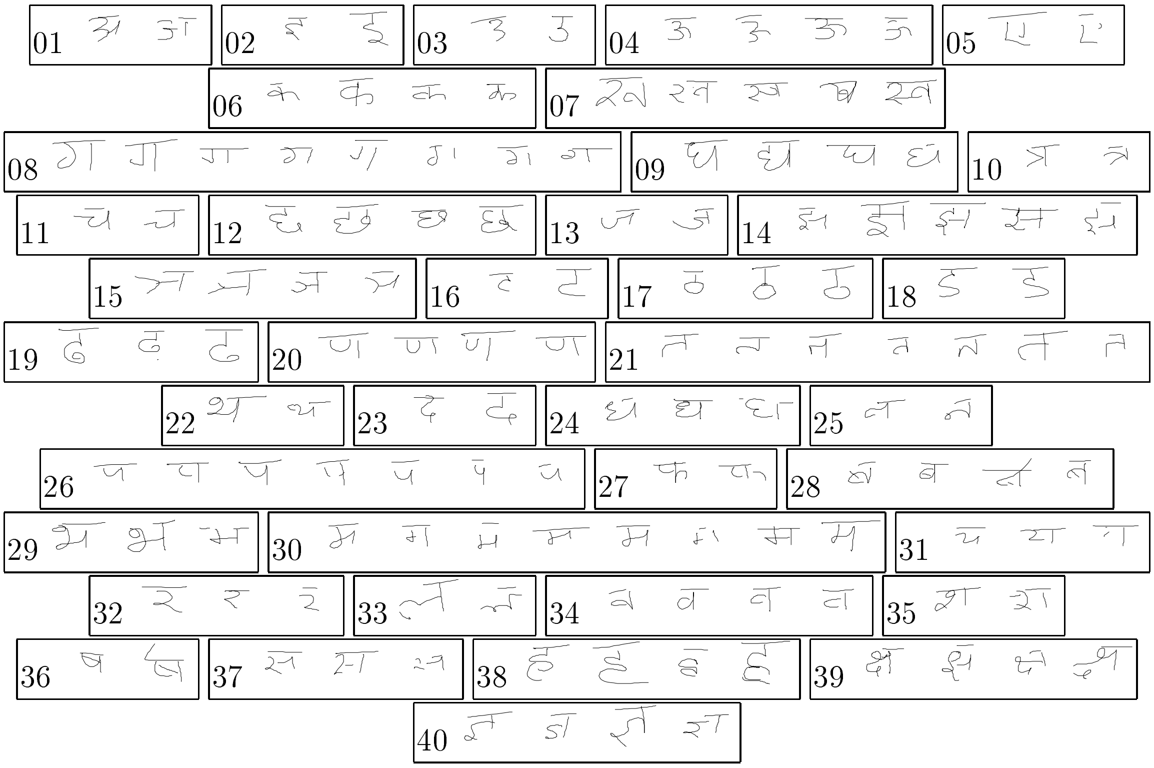


Figure 8.4: Prototypical examples of each lexeme from each of the 40 Devanagari characters.

along the trace of the on-line sample point sequence, and three nearest neighbor classifiers that focus on off-line features. In order to better represent the complex intra-class variations that exist, lexemes are identified within each character class, and these are represented by separate models (for HMM classifiers). A typical example of each of these writing styles (the pattern closest to the cluster center) can be seen in Figure 8.4.

An examination of the handwritten characters in Figure 8.4 reveals some of the complexities of the Devanagari classification problem. Poor writing, and possibly input device errors have made the third and fourth examples of character 07 unreadable, along with the third example of 38, the third example of 31, and the fourth

Table 8.2: Feature sets used in each classifier.

Classifier No.	Feature Type	Features
1	on-line	dx, y, $sin(\theta)$, $cos(\theta)$ for each sample point
2	on-line	curvature and orientation for each critical point
3	off-line	stroke direction histogram for each box of 5x5 grid
4	off-line	dx and dy from beginning to end of each stroke
5	off-line	center of gravity of each stroke

Table 8.3: Five different classifiers used in classifier combination.

Classifier No.	Model	Accuracy
1	Hidden Markov Model	69.2%
2	Hidden Markov Model	43.1%
3	Nearest Neighbor	77.4%
4	Nearest Neighbor	44.7% (1.3% rej)
5	Nearest Neighbor	40.7%

example of 26. In addition, the fourth example of 09 and the first example of 24 are virtually identical, while the third example of 07 and the third example of 37 seem to be better representatives of each other's character class, demonstrating the large intra-class variability. Some other character pairs with low levels of discrimination are shown in Table 8.1. It should also be pointed out that certain character pairs, in particular characters 10 and 15, are known to be difficult for humans to distinguish in isolation.

Tables 8.2 and 8.3 give a summary of the different classifiers and feature sets used. Since the handwritten strokes are captured as a sequence of (x, y) sample points, we can make use of this trace of the writing to capture information on how the strokes were formed. *Classifier 1* extracts features for each equidistant resampled point along the curve of writing. These features are the *Local* features described in Section 6.2.1.

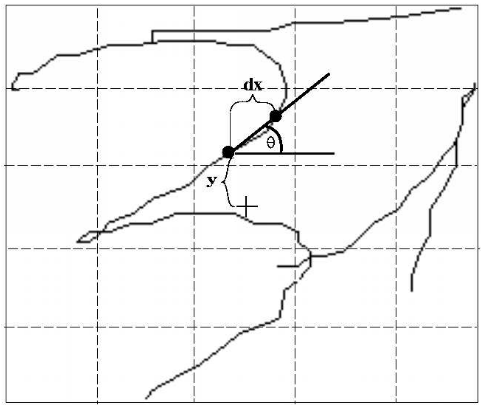


Figure 8.5: Features for classifiers 1 and 3. The "+" marks the center of the character. Features are calculated between every pair of sample points. Classifier 3 quantizes θ and measures the total distance traveled in each direction for each box in the grid.

Classifier 2 uses the following five features extracted at each local (x, y) extrema point, p_i, (referred to as *critical points*) not including the endpoints of a stroke:

1. The angle, θ_c, formed between the two segments, one from the previous critical point, p_{i-1}, to the current point, and one from the current to the next critical point, p_{i+1}, $0 \leq \theta_c \leq 180$.

2. The orientation of the curve, is captured by the sine and cosine of the angle of incline for the normal to the tangent of this critical point,

3. The change in x and the change in y from the previous critical point, p_{i-1}, to the next critical point, p_{i+1}.

This is a variation on the *Critical Point* features described in Section 6.2.2. *Classifier 3* attempts to capture the directional properties of the written strokes, local to different spatially separate parts of the character. *Classifier 4* and *Classifier 5*, two additional off-line classifiers, are respectively the *stroke endpoint* and *center of gravity* features described in Sections 6.2.4 and 6.2.5. Figure 8.5 presents an example of the *Local* and spatial/directional features extracted from a handwritten Devanagari character.

As discussed in Section 6.3, the final classification can be taken as the class C_i which maximizes the following posterior probability:

$$P(C_i|O) = \sum_{j=1}^{5} P(C_i|Q_j(O)), \tag{8.1}$$

where O is the observed character, $Q_j(O)$ is the function which the jth classifier uses to classify an observed character, O, and $P(C_i|Q_j(O))$ is the probability that the true class of O is class C_i given that classifier j has classified the character as class $Q_j(O)$. These conditional probabilities are estimated by examining the resubstitution accuracies for the HMM classifiers, and the leave-one-out accuracies for the nearest neighbor classifiers.

8.3 Results

We have attempted to recognize the 40 major Devanagari characters (Figure 8.2) that occur in the core-strip of the writing. For this study, the modifier symbols and conjunct formation are not considered. Five samples of each character from twenty different writers were collected using an IBM/Cross CrossPad. It should be noted that all our subjects have been living in the United States for some time and are only casual writers of Devanagari. The sample data has a mixture of writing styles as the subjects come from different geographical regions in India.

A combination of hidden Markov models and nearest neighbor classifiers are used for classification, capturing different levels of on-line and off-line information. Three samples of each writer are used for training and the remaining two samples have been used for testing. A recognition rate of 69.2% is achieved using a traditional HMM-based on-line recognizer. An examination of the substitution errors, revealed that the HMM model did not capture many of the structural properties needed for classification. This motivates the use of multiple classifiers built on features of different scales in an attempt to capture some of the structural properties of the characters.

Our classifier was tested using a total of 1600 characters from 40 classes: 2 samples for each character, independent of the training data, from each of the 20 writers. Table 8.3 shows the accuracies obtained for each classifier independently. The combined classifier achieves an accuracy of 86.5% with no rejects. Since these classifiers can be run in parallel, the classification time is defined by the most time-consuming classifier, which is *Classifier 1* due to its use of a large number of local on-line features. The currently unoptimized code processes characters at a rate of approximately 1.5 characters per second on a 250MHz UltraSparc.

The contribution of the two on-line classifiers can be easily underestimated if we look at the recognition accuracies of individual classifiers (Table 8.3). While *Classifier 3* has the best single classifier accuracy, the addition of the other two off-line classifiers (*Classifier 4* and *Classifier 5*) does not increase the overall accuracy unless we also include at least one of the two on-line classifiers (*Classifier 1* or *Classifier 2*). However, any of the off-line classifiers can be used in combination with *Classifier 1* for a significant increase in recognition accuracy. (between 2.3% and 6.8% above the highest single classifier accuracy of the two). In the full combination of five classifiers, our poorest performance is on character 31 which is often confused with character 22. This appears to be a reasonable confusion. Other common confusion pairs are (25, 21), (19, 17), and (29, 30), each of which are structurally similar.

Table 8.4: Percentage of test characters for which the correct class falls within the top 10 choices.

N	1	2	3	4	5	6	7	8	9	10
Accuracy	86.5%	92.1%	94.7%	95.6%	96.8%	97.5%	98.2%	98.3%	98.5%	98.7%

The accuracies reported here are on discrete characters. Many of the common confusions can be corrected through the use of context when these characters are found in a word (of course, this requires a correct segmentation of a word into constituent characters). The potential benefits of such a method depend on how frequently the correct character class can be found as the second or third choice. Table 8.4 shows the percentage of times the correct classification is found within the top 5 choices. The successful use of context to select between the top two or three character choices has the potential to significantly boost the performance of our classifier.

8.4 Summary

A system for the recognition of unconstrained online Devanagari characters has been presented. This is the first such work reported to the best of our knowledge. A combination of classifiers capturing both on-line and off-line features has been described yielding a classification accuracy of 86.5% with no rejects. In addition, approximately 95% of the time the correct classification can be found in the top 3 choices.

Therefore, further improvements in performance are expected by making use of the word-level contextual information. In Chapter 6 this approach was shown to produce good results on the English character recognition problem, and therefore these results indicate the ability of the recognition system to handle different languages and symbols.

Chapter 9

Writer-Dependent Systems

9.1 Introduction

In the previous chapters, results have been presented on writer-independent handwriting recognition tasks. In other words, a recognition system was trained on the handwritten data of several writers for the purpose of achieving a good recognition accuracy on a large number of writers. However, the accuracy of a recognition system, for any particular writer, can be improved if character models are adjusted to better match the characteristics of that writer's handwriting rather than using the more generalized models trained for a large number of writers. Such a writer-dependent system can be constructed for a user in one of two ways: by collecting enough data from that user to create robust models of his handwriting, or by adapting the models of a writer-independent recognition system to better fit the handwriting of those users from whom a relatively small amount of data bas been collected.

For practical purposes, a handwriting recognition system designer may want to specify the minimum amount of data that a user must supply for training his models. While a user may be willing to dedicate up to a couple of hours in providing training data if there is some assurance that it may improve recognition accuracies, we cannot expect individual users to provide the same amount of training data as in the writer-independent case. In addition, if the within-class variability in a user's writing can be characterized by more than one style, it may be appropriate to separate these styles into different lexeme models. An increase in the number of models creates a demand for more training data, and therefore we should ensure that we use the available data to the best of our ability.

In this chapter, we explore the role of lexeme modeling for writer-dependent handwriting recognition. This includes the issue of whether or not it is appropriate to model a single writer's handwriting with multiple models per character class, as well as how knowledge of the lexemes present in a writer-independent dataset can be used to assist the writer-adaptation process.

9.2 Writer-Dependent Lexeme Identification

It is, given enough data, possible to identify lexemes within a writer's handwritten data using the same method as the writer-independent case, however the amount of data required form good clusters within a character class make this method infeasible for the experiments presented here.

9.3 Writer-Adaptation

If we assume that each writer can be expected to provide only a small amount of data during the training phase, this data may not be sufficient to identify lexemes or train multiple models per character. Therefore, we perform writer adaptation by identifying which writer-independent lexemes best match the writing style of a particular writer. These relatively small number of models are then *retrained* using the writer's data. Our goal is to make an efficient use of the data that is available by using writer-independent models to compensate for the small amount of writer-dependent data.

Our method of adaptation (previously introduced in [23]) begins by classifying each input character provided by the writer into one of the writer-independent lexeme categories. All labeled characters provided by the writer are compared against the writer-independent lexeme models of the same class (e.g., all the 'a' models), and the most similar model gives the lexeme identity. In this way, each writer-specific character is given a label corresponding to one of the writer-independent lexemes. Counts, n_{L_i}, are kept, for each lexeme L_i, of the number of times a character has been identified as lexeme L_i. Lexeme models are then retrained using the newly labeled lexeme data from the user. If the number of examples available from the writer for a particular lexeme is less than a threshold n_T, the corresponding writer-independent lexeme is retained instead.

Figure 9.1 gives a flow diagram of the proposed recognition system. Lexeme modeling is done using the full set of writer-independent data, and is broken into two

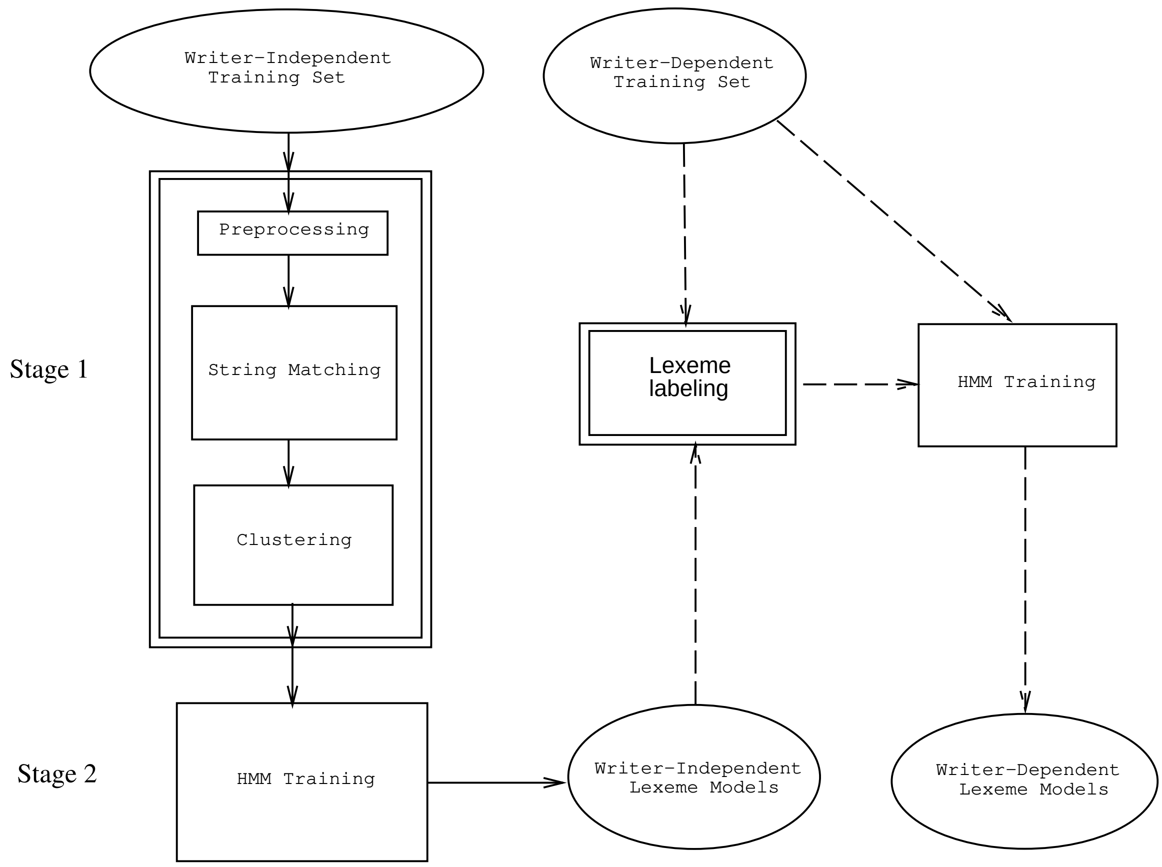


Figure 9.1: An overview of the model training process. Dashed lines show the flow of events for writer adaptation. The number of states has reduced from 16 to 14.

stages. Stage 1 performs an initial clustering of the lexemes using a string matching measure and a squared-error clustering algorithm, as described in Section 3.4. Stage 2 defines a continuous hidden Markov model for each of these clusters, using the data within each cluster as the training data. At this point, we have a writer-independent set of lexeme models. These models are then adapted to individual writers in the dataset. In Figure 9.1 the dashed lines show the flow of events for this writer adaptation process.

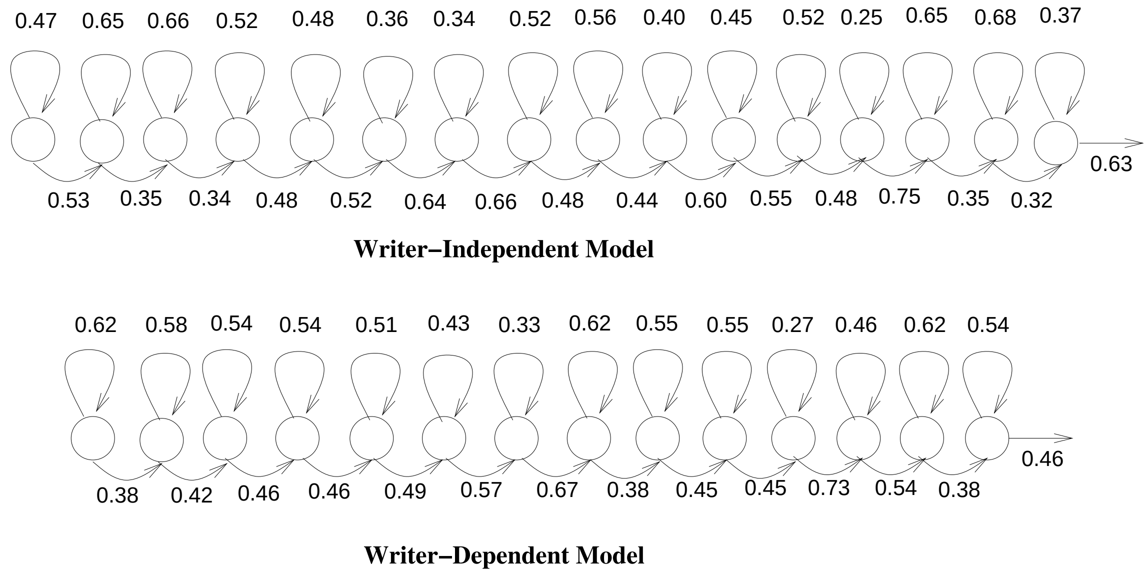


Figure 9.2: The parameters of a hidden Markov model trained for a lexeme of the character 'e' before and after writer adaptation.

9.4 Results

Figure 9.2 shows a hidden Markov model for a lexeme of the character 'e' before and after writer adaptation has been performed. This shows that model parameters have been modified to adapt to the user's handwriting.

Figure 9.3 shows the set of lexemes that were identified for the digits written by "Writer #5", along with the number of his training examples that were identified with each lexeme. Due to the small amount of digit data available from each writer, and the time required to train models, evaluation of writer adaptation was conducted using the resubstitution method. In addition, the number of writers in this experiment was limited to six writers whose training data contained examples of all ten digit classes. The performance of both the writer-independent recognition system, and the newly adapted writer-dependent system are shown for each of the six writers in Table 9.1.

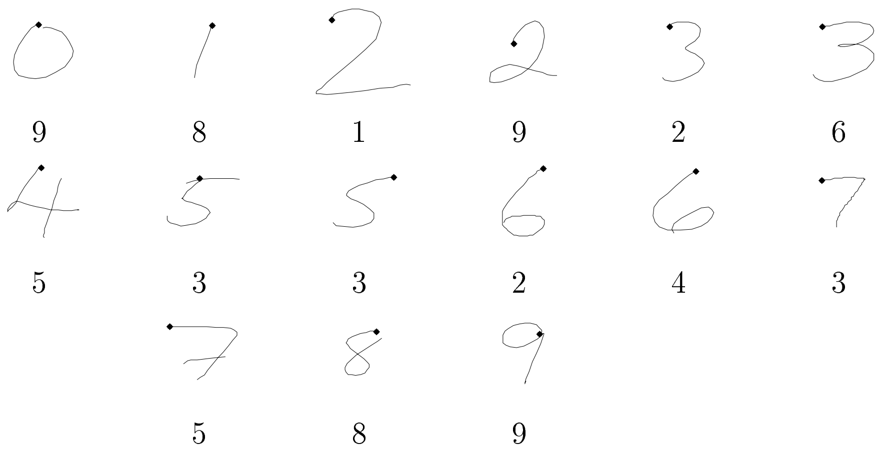


Figure 9.3: Lexemes identified in digits written by "Writer #5", and the number of occurrences matching that lexeme in that writer's data. Note that for this writer, we have identified 15 lexemes (out of a total of 26 lexemes for the writer independent data).

For a comparison, a recognition system was trained in which the writer's data was used to train a single model per character class. As was done in the multiple lexeme model case, character classes for which there was not enough data to train a model were represented by the corresponding writer-independent model for that character.

Table 9.2 shows the accuracies obtained for writer-adaptation of lowercase characters for 11 different writers. The data from each writer has been randomly split into independent training and test sets. An open question in writer adaptation is whether individual writers can be said to have more than one writing style for a particular character. In the context of our classification problem, this can be interpreted to mean the following: do disparities exist within a character class for a single writer, such that the character class can be more adequately modeled using multiple lexeme models? Table 9.2 shows that, for every writer, the writer-dependent models produce better recognition accuracies than the writer-independent models. In addition, an

Table 9.1: Writer-dependent digit recognition accuracies using different writer-adaptation methods.

User	No. Digits	Writer Independent	Writer Dependent	
			Single Lexeme	Mult. Lexemes
Writer #1	95	94%	100%	100%
Writer #2	107	91%	99%	100%
Writer #3	108	86%	96%	98%
Writer #4	64	91%	91%	94%
Writer #5	77	81%	94%	95%
Writer #6	111	90%	100%	100%
Overall	562	88.8%	97.2%	98.2%

important observation is that there is never a loss of accuracy by using multiple lexeme models and in most cases this resulted in a significant increase in accuracy. One exception to this is Writer #2, in which the multiple lexeme model accuracy is less than the single lexeme model accuracy by a single character error. This is not considered a significant difference. Overall, the use of multiple writer-dependent lexeme models reduced the error rate by an average of 16.3% over that achieved using a single writer-dependent model per character class. Furthermore, this is a 53.8% reduction in error compared to the classification accuracies obtained by the writer-independent recognizer. The best improvement occurred for "Writer #20", with a 50.7% reduction in the error rate compared to the single writer-dependent model case, and a 70.8% reduction in error compared to the writer-independent case.

9.4.1 Word Recognition

The ability of our writer adaptation technique was next tested for the word recognition system described in Chapter 7. Writer-independent models used in the adaptation

Table 9.2: Lowercase letter recognition accuracies using different writer-adaptation methods.

Writer	No. Chars Train / Test	Writer Indep. Acc. (%)	Writer Dep. Acc. (%) Single Lexeme	Mult. Lexemes
#1	1155 / 1125	84.82	91.11	92.89
#2	1121 / 1094	83.75	95.34	95.25
#3	1190 / 1171	68.53	82.66	84.54
#4	1117 / 1117	70.73	83.26	84.15
#6	1247 / 1223	69.11	80.95	82.33
#7	1177 / 1158	84.15	92.40	94.04
#13	1223 / 1195	75.06	85.69	88.20
#17	1070 / 1049	85.37	91.42	92.66
#18	1339 / 1312	85.06	92.61	93.98
#19	1279 / 1253	55.25	74.38	78.21
#20	1250 / 1226	78.23	87.11	93.64
Overall	13168 / 12923	76.16	86.85	88.99

process were trained on 122,410 examples from 93 character classes. A set of discretely written characters in a handprinted style, and a second set in a cursive style were collected from each writer in the experiment. Table 9.3 shows the amount of data that is available from each writer. Each data set from each writer covers all 93 classes, with a minimum of 5 examples per class for each handprinted and cursive style characters in almost all cases. This is a very small amount of data for training. Experiments conducted by Subrahmonia et al. [100] indicate that a minimum of 500 words of training data, or as much as 2000 words of training data could be required to train adequate writer-dependent models for a word recognition system which makes use of models of 93 different character classes. If we assume an average of 5 characters per word, this would be approximately 2,500 to 10,000 characters required to train these models. As can be seen in Table 9.3, we have significantly less data than this.

Table 9.3: Number of character samples available from different writers of both hand-printed and cursive styles.

Writer	No. Handprint	No. Cursive
#1	487	480
#2	487	489
#3	443	464
#4	487	490
#5	442	322
#6	473	432
#8	476	469

Furthermore, the data is split into at least two different styles, handprint and cursive, effectively dividing the data available to any lexeme model into less than half of the total data for that character.

Tables 9.4 and 9.5 show the word-level accuracies for the same writers used in experiments of Chapter 7, for both case-dependent and case-independent classification. Due to the absence of character-level data for Writer #7, this writer has been omitted from these experiments. A simple modification has been made to the algorithm to accommodate for a lack of adequate training data: a copy of all original writer-independent models are retained along with the newly retrained models to form the final recognition system. This is to allow the system to capture writing styles that may not have occurred in the training data simply due to their relatively low frequency of occurrence. Nonetheless, those lexemes for which models have been retrained should still dominate classification when a test pattern is an example of that lexeme.

The first thing that is apparent from Tables 9.4 and 9.5 is that any attempt to retrain a single lexeme model for each character using just the newly acquired data

Table 9.4: Case-dependent word recognition accuracies after writer-adaptation.

Writer	No. Test Examps	Writer Indep. Acc.	Writer Dep. Acc.	
			Single Lexeme	Mult Lexeme
#1	571	79.9%	45.2%	79.2%
#2	614	57.5%	12.9%	58.8%
#3	577	72.3%	36.4%	74.7%
#4	605	69.4%	28.3%	72.4%
#5	610	84.1%	47.9%	83.8%
#6	599	48.2%	17.7%	50.8%
#8	591	55.2%	47.2%	62.8%
Total	4132	67.1%	33.8%	69.4%

Table 9.5: Case-independent word recognition accuracies after writer-adaptation.

Writer	No. Test Examps	Writer Indep. Acc.	Writer Dep. Acc.	
			Single Lexeme	Mult Lexeme
#1	571	83.9%	47.6%	83.2%
#2	614	58.5%	13.8%	59.8%
#3	577	77.3%	38.6%	79.4%
#4	605	75.2%	31.4%	78.0%
#5	610	87.9%	49.8%	87.7%
#6	599	49.4%	19.5%	51.9%
#8	591	58.2%	49.4%	66.0%
Total	4132	70.6%	35.9%	72.8%

Table 9.6: Writer-adaptation for a single writer with a larger set of training examples. Case-independent accuracies are reported.

User	No. Train Characters	No. Test Words	Writer Independent	Writer Dependent	
				Single Lex.	Mult. Lex
# 10	2450	612	72.1%	62.1%	78.6%

produces very bad results. This is due to a lack of training data, and therefore this experiment can be used to really test the limitations of our writer-adaptation method. Writer adaptation achieves an average reduction in error rate of 7.0% for the case-dependent word classification task, and 7.5% error rate reduction for the case-independent task. While this is not a large improvement in performance, given the very small amount of data available for adaptation, it is still significant. Perhaps even more interesting is the fact that the recognition rate only drops by a miniscule amount due to adaptation in two cases, writers #1 and #5. The best results obtained were for writer #8, which achieved an 18.7% reduction in error.

In order to demonstrate the scalability of the writer-adaptation algorithm when tested on words, a new set of training characters were collected from a single writer. Rather than requesting both cursive and handprint characters, the writer was allowed to write in only his preferred style (which was handprint), and a much larger quantity of data was collected than the previous word-level experiments. Table 9.6 presents the results of this experiment. While there is still not enough training data to construct a single model per character class that outperforms the writer-independent recognizer, the use of the writer-adaptation algorithm to form multiple lexeme models per character class can be seen to reduce the error rate by 21.7%.

9.5 Summary

We have presented a hidden Markov model based system that performs writer-adaptation by making use of a writer-independent set of writing style models (lexemes) to identify different writing styles in the writer-dependent data. These models are then used to train new models for this subset of lexemes. Lexemes that are present in the writer's data, but that do not contain an adequate number of training examples are replaced with the writer-independent models. This technique is compared to a system in which a single model per character class is trained from the writer's data. The results presented show that, on average, error rate reductions of 37.5% for digits from 6 writers, and 16.3% for lowercase characters from 11 writers are achieved compared to accuracies achieved using a single writer-dependent model. Other experiments, in which a very small amount of data was available for the adaptation process, produced word-level error rate reductions of 7.0% for case-dependent word classification, and 7.5% for case-independent word classification compared to writer-independent classification accuracies. Moreover, even with such a small amount of data available for adaptation, no significant increase in the error rate is ever introduced by this adaptation method. In addition, the scalability of this method for word recognition was demonstrated on a single writer with a significantly larger, but still small, training set. This produced a 21.7% reduction in error rate compared to the writer-independent case.

Chapter 10

Conclusions

Many challenges need to be overcome in the field of online handwriting recognition, if handwriting is to evolve into a reliable method of data entry. In particular, a recognition system must be able to accommodate the large variations that exists between writing styles. The adaptation of models to the writing style(s) of a single writer has the potential to greatly increase accuracies for that writer. However, the amount of data a single writer is typically willing to provide is not enough to build robust models. Therefore, an adaptation method must best use knowledge it has gained from a large set of writers to assist it in creating robust models for individual writers. In addition, the combination of multiple sources that capture different types of information about the handwritten characters is becoming a common way of capturing and integrating different types of variations in the writing. Finally, these concepts are just as important to languages other than those based on the Roman alphabet. In particular, Devanagari script is used by hundreds of millions of writers, and is potentially very well suited for a pen interface. This Chapter presents a summary of the contributions made in this thesis, along with a discussion of some future areas of research is likely to advance the state-of-the-art.

10.1 Contributions

The contributions of this thesis can be summarized as follows:

- A method of identifying different character writing styles (referred to as lexemes) in online data has been presented that makes use of a string matching measure and a squared error clustering algorithm. This method also makes use of a cluster dispersion measure to automatically choose an appropriate number of lexemes.

- A decision tree classifier was constructed which makes use of distances to prototypical lexemes to make decisions at each node of the tree. This method has been shown to achieve a classification accuracy of 86.9% on the 36-class alphanumeric character recognition problem (10 digits, 26 lowercase characters). The use of a decision tree provided a 99.4% reduction in classification time as compared to a full nearest neighbor classifier, while retaining 97.7% of the recognition accuracy.

- A hidden Markov model classifier was constructed in which a single model was trained for each lexeme class, where the lexemes have been based on the string matching measure. This method has been shown to achieve a recognition accuracy of 98.0% on digits, 86.85% on lowercase characters, and 89.47% on uppercase characters. These results are a significant improvement over those obtained when only a single model is trained for each character class: 95.62% on digits, 84.43% on lowercase characters, and 84.04% on uppercase characters. In addition, the classification accuracy for the combined 62-class problem has been shown to be 76.45%, with the correct classification choice within the top 3 choices over 95% of the time.

- A method of defining lexemes in the hidden Markov model probability space was demonstrated. This method performs a K-Means clustering in which the distance of a pattern to a cluster is the likelihood that the pattern was generated by a HMM trained to model that cluster. It was shown that, compared to the clustering method based on the string matching measure, a 4.9% decrease in error rate can be achieved. Moreover, the algorithm has been shown to obtain a good clustering, similar to the best clustering using the string matching measure, without the requirement of any sophisticated cluster initialization.

- A method of combining multiple classifiers based on both online and offline features that capture different levels of structural characteristics of handwriting was demonstrated. This method was demonstrated for 5 different classifiers that capture different online and offline properties of the writing, as well as different scales of structural information. The focus of this technique is on the misclassification characteristics of each classifier, and therefore this approach also improves the accuracy of individual classifiers. The best classifier combination achieves a recognition accuracy of 84.0% on the 62-class alphanumeric character recognition problem. This is a 32% reduction in error compared to the best single classifier accuracy.

- The application of character lexemes to the recognition of words was explored. An experiment involving a lexicon of 483 words made up of 93 different character classes (10 digits, 26 uppercase and 26 lowercase characters, and 31 special symbols) was conducted. The use of lexemes was shown to reduce the error rate by 32.0% for case-dependent word recognition and 35.4% for case-independent word recognition. Final writer-independent classification accuracies obtained on 8 different writers ranged from 48.2% to 84.1% with an average of 64.7%.

- The application of our recognition system to the classification of Devanagari script was explored. This system achieves an accuracy of 86.5% on 40 common Devanagari characters with the correct classification occurring in the top 3 choices approximately 95% of the time. This is the first time any result has been reported on online Devanagari recognition to our knowledge.

- A writer-adaptation technique was described in which knowledge about lexemes that have been identified for a large number of writers are used to help identify lexemes within a single writer's handwritten data. This method requires very little data from the writer for which the models are adapted. Experiments on lowercase handwritten characters show that average error rate reductions of 53.8% as compared to writer-independent classification accuracies, and 16.3% as compared to classification using a single writer-dependent model are achieved. The latter result indicates that a single writer tends to write in multiple styles that can best be described using separate models. In the extreme, it was shown that for less than 10 examples per character class divided among a minimum of two different writing styles, our adaptation technique can still produce error rate reductions for word recognition by an average of 7.5%, and has been shown to reduce error rates by as much as 18.7% with this small amount of training data. For a larger training set, this method has been shown to reduce the error rate by 21.7%. In no instance did the adaptation technique increase the error rate of the recognition system by a significant amount.

10.2 Future Work

This section presents some areas in which our work could be extended to improve the online handwriting recognition accuracy.

10.2.1 Classifier Combination for Word Recognition

Our current word-level results make use of the *curvature/orientation* features. An improvement on these results should be possible through the application of a combination of classifiers. While this is difficult during the word recognition process, due to the fact that character segmentations are unknown until recognition is complete, combined classifier classification scores could be used to reliably rescore the top classification choices from the current classifier.

10.2.2 Devanagari Script Recognition

Results have been presented on Devanagari script recognition for a set of 40 characters. For these results to be applicable for Devanagari word recognition, we require the following:

- A method of segmenting Devanagari words into characters. Alternatively, an approach that accomplishes concurrent segmentation and recognition could be attempted, such as was accomplished for English word recognition. The large number and order of strokes in Devanagari makes this a challenging problem however.
- The extension of our models to handle modifiers. This will increase the complexity of the character models.
- A word-level size and location normalization procedure may be required if segmentation and recognition will be done concurrently. Our current method of English word normalization might be of use for this task with some minor adjustment. For example, the Shirorekha should be easily found, providing a reliable estimate of the mid-line.

10.2.3 The HMM-based Clustering Algorithm

In Section 5.3, a method was shown in which the K-Means algorithm was used to define a clustering algorithm in which a hidden Markov model represents each cluster. This approach, referred to as the HMM-based clustering algorithm, has been shown to produce a reasonable clustering on randomly initialized examples where the number of clusters is predefined. However, as is the case with most clustering algorithms, the potential exists for this algorithm to converge toward a local minima if the initial set of clusters is poorly chosen, and therefore never achieve an solution near the globally optimal solution. This can be alleviated by avoiding making any "hard" cluster membership decisions in the early iterations of clustering, and gradually allowing patterns to commit more and more to a single cluster. Methods of simulated annealing [52], deterministic annealing [40, 90, 107], and self annealing [28, 88] have been applied to non-temporal Gaussian mixtures, and may be useful techniques when properly adapted to temporal data. Figueiredo [28] has presented a method in which the number of clusters is defined by starting with an initially large number of clusters, some of which will be annihilated during the clustering process.

10.2.4 Writer-Adaptation

The writer-adaptation approach described in Section 9.3 has been shown to produce increases in classification accuracy at the word level when only a very small amount of training data is available from the user. An evaluation of this method for a slightly larger training set has produced very good results for the character recognition problem, and a single instance of the word recognition problem. In order to test the full potential of this adaptation technique for word recognition, an evaluation using a somewhat large set of training data from each of a large set of writers is of interest. In addition, a method of selecting character examples from the writer-independent training set to supplement the data available for writer-dependent model training could be implemented. Once lexeme labels are established for the new writer's data, character examples could be randomly selected from the data used to train the writer-independent model of that lexeme, thereby selecting supplementary examples from the writer-independent data that are very similar to each writer-dependent pattern.

Bibliography

Bibliography

[1] http://www.artcomp.com

[2] Special Issue on Scientific Visualization, *IEEE Computer*, vol. 32, no. 12, Dec. 1999.

[3] L.R. Bahl, P.F. Brown, P.V. deSouza, and R.L. Mercer, "Maximum Mutual Information Estimation of Hidden Markov Model Parameters for Speech Recognition," *Proc. ICASSP'86*, Tokyo, Japan, pp. 49-52, Oct. 1986.

[4] L. R. Bahl, F. Jelinek, and R. L. Mercer, "A Maximum Likelihood Approach to Continuous Speech Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 5, no. 3, pp. 179-190, March 1983.

[5] V. Bansal and R.M.K. Sinha, "On how to Describe Shapes of Devanagari Characters and Use Them for Recognition," *Proc. 5th Int. Conf. Document Analysis and Recognition*, Bangalore, India, pp. 410-413, Sept. 1999.

[6] H. Beigi, "Pre-Processing the Dynamics of On-Line Handwriting Data, Feature Extraction and Recognition," *Proc. 5th Int. Workshop on Frontiers in Handwriting Recognition*, Colchester, England, pp. 255-258, Sept. 1996.

[7] H. Beigi, K. Nathan, G. J. Clary, and J. Subrahmonia, "Size Normalization in On-Line Unconstrained Handwriting Recognition," *Proc. ICIP'94*, Austin, Texas, pp. 169-173, Nov. 1994.

[8] E.J. Bellegarda, J.R. Bellegarda, D. Nahamoo, and K.S. Nathan, "A Probabilistic Framework For On-Line Handwriting Recognition," *Proc. 3rd Int. Workshop on Frontiers in Handwriting Recognition*, Buffalo, New York, pp. 225-234, May 1993.

[9] Y. Bengio, Y. LeCun, C. Nohl, and C. Burges, "LeRec: A NN/HMM Hybrid for On-Line Handwriting Recognition," *Neural Computation*, vol. 7, no. 6, pp. 1289-1303, June 1995.

[10] S. Bercu and G. Lorette, "On-Line Handwritten Word Recognition: An Approach Based on Hidden Markov Models," *Proc. 3rd Int. Workshop on Frontiers in Handwriting Recognition*, Buffalo, New York, pp. 385-390, May 1993.

[11] J.J. Brault and R. Plamondon, "Segmenting Handwritten signature at Their Perceptually Important Points," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 15, no. 9, pp. 953-957, Sept. 1993.

[12] L. Breiman, "Bagging Predictors," Technical Report 421, Dept. of Statistics, Univ. of California at Berkeley, 1994.

[13] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*, Wadsworth, Belmont, CA, 1984.

[14] D.J. Burr, "Designing a Handwriting Reader," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 5, no. 5, pp. 554-559, Sept. 1983.

[15] K.F. Chan and D.Y. Yeung, "Elastic Structural Matching for On-Line Handwritten Alphanumeric Character Recognition," *Proc. 14th Int. Conf. Pattern Recognition*, vol. 2, Brisbane, Australia, pp. 1508-1511, Aug. 1998.

[16] B. B. Chaudhuri and U. Pal, "A Complete Printed Bangla OCR System," *Pattern Recognition*, vol. 31, no. 5, pp. 531-549, May 1997.

[17] B.B. Chaudhuri and U. Pal, "An OCR System to Read Two Indian Language Scripts: Bangla and Devanagari," *Proc. 4th Int. Conf. Document Analysis and Recognition*, Ulm, Germany, pp. 1011-1015, Aug. 1997.

[18] K.-F. Chan and D.-Y. Yeung, "Elastic Structural Matching for On-line Handwritten Alphanumeric Character Recognition," *Proc. 14th Int. Conf. Pattern Recognition*, Brisbane, Australia, pp. 1508-1511, Aug. 1998.

[19] A.K. Jain and J. Coggins, "A Spatial Filtering Approach to Texture Analysis," *Pattern Recognition Letters*, vol. 3, pp. 195-203, 1985.

[20] S.D. Connell, "A Comparison of Hidden Markov Model Features for the Recognition of Cursive Handwriting," *Master's Thesis*, Dept. of Computer Science, Michigan State University, May 1996.

[21] S. Connell and A.K. Jain, "Learning Prototypes for On-Line Handwritten Digits," *Proc. 14th Int. Conf. Pattern Recognition*, Brisbane, Australia, pp. 182-184, Aug. 1998.

[22] S. Connell and A.K. Jain, "Template-based Online Character Recognition," *to appear in Pattern Recognition.*

[23] S. Connell and A.K. Jain, "Writer Adaptation of Online Handwritten Models," *Proc. 5th Int. Conf. Document Analysis and Recognition*, Bangalore, India, pp. 434-437, Sept. 1999.

[24] J. Crettez, "A Set of Handwritten Families: Style Recognition," *Proc. 3rd Int. Conf. on Document Analysis and Recognition*, Montreal, Canada, pp. 489-494, Aug. 1995.

[25] D.L. Davies and D.W. Bouldin, "A Cluster Separation Measure," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 1, no. 2, pp. 224-227, April 1979.

[26] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis*, Wiley-Interscience, 1973.

[27] R. P. W. Duin, D. de Ridder, and D. M. J. Tax, "Experiments With Object Based Discriminant Functions; A Featureless Approach to Pattern Recognition," *Pattern Recognition Letters*, vol. 18, no. 11-13, pp. 1159-1166, 1997.

[28] M.A.T. Figueiredo and A.K. Jain, "Unsupervised Selection and Estimation of Finite Mixture Models," *to appear in ICPR'2000*, Barcelona, Sept. 2000.

[29] M.A.T. Figueiredo, H.M.N. Leitao, and A.K. Jain, "On Fitting Mixture Models," in *Energy Minimization Methods in Computer Vision and Pattern Recognition*, M. Pellilo and E. Hancock, eds., pp. 54-69, Springer Verlag, 1999.

[30] G.D. Forney, "The Viterbi Algorithm," *Proc. of the IEEE*, vol. 61, pp. 268-278, 1973.

[31] Y. Freund and R.E. Shapire, "Experiments With a New Boosting Algorithm," *Proc. 13th Int. Conf. Machine Learning*, pp. 148-156, 1996.

[32] http://www.wiley.co.uk/college/busin/icmis/oakman/outline/chap12/misc/graffiti.htm

[33] R. M. Gray, "Vector Quantization," *Readings in Speech Recognition*, Alex Waibel and Kai-Fu Lee, eds., Morgan Kaufmann, pp. 75-100, 1990.

[34] S.A. Guberman, I. Lossev, and A.V. Pashintsev, "Method and Apparatus for Recognizing Cursive Writing from Sequential Input Information," *US Patent WO94/07214*, March 1994.

[35] W. Guerfali and R. Plamondon, "Normalizing and Restoring On-Line Handwriting," *Pattern Recognition*, vol. 26, no. 3, pp. 419, March 1993.

[36] I. Guyon, P. Albrecht, Y. Le Cun, J.S. Denker, and W. Hubbard, "Design of a Neural Network Character Recognizer for a Touch Terminal," *Pattern Recognition*, vol. 24, no. 2, pp. 105-119, Feb. 1991.

[37] I. Guyon, L. Schomaker, R. Plamondon, M. Liberman, and S. Janet, "UNIPEN Project of On-Line Data Exchange and Recognizer Benchmarks," *Proc. 12th Int. Conf. Pattern Recognition*, Jerusalem, Israel, pp. 29-33, Oct. 1994.

[38] S.C. Hinds, J.L. Fisher, and D.P. D'Amato, "A Document Skew Detection Method Using Run-Length Encoding and the Hough Transform," *Proc. 10th Int. Conf. Pattern Recognition*, Atlantic City, NJ, pp. 464-468, June 1990.

[39] T.K. Ho, J.J. Hull, and S.N. Srihari, "Decision Combination in Multiple Classifier Systems," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 16, no. 1, pp. 66-75, Jan. 1994.

[40] T. Hofmann and J. Buhmann, "Pairwise Data Clustering by Deterministic Annealing," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 19, no. 1, pp. 1-14, Jan. 1997.

[41] J.M. Hollerback, "An Oscillation Theory of Handwriting," *Biological Cybernetics*, vol. 39, pp. 139-156, 1981.

[42] H. Hotelling, "Analysis of A Complex of Statistical Variables Into Principal Components," *JEP*, vol. 24:417-441, pp. 498-520, 1933.

[43] J. Hu, A. S. Rosenthal, and M. K. Brown, "Combining High-Level Features with Sequential Local Features for On-Line Handwriting Recognition," *Proc. Italian Image Process. Conf.*, Florence, Italy, pp. 647-654, Sept. 1997.

[44] J. Hu, M. K. Brown, and W. Turin, "HMM Based On-Line Handwriting Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 18, no. 10, pp. 1039-1045, Oct. 1996.

[45] X.D. Huang and M.A. Jack, "Semi-Continuous Hidden Markov Models For Speech Signals," *Readings in Speech Recognition*, Alex Waibel and Kai-Fu Lee, eds., Morgan Kaufmann, pp. 340-345, 1990.

[46] Y.S Huang, C.Y. Suen, "A Method of Combining Multiple Experts for the Recognition of Unconstrained Handwritten Numerals," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 17, no. 1, pp. 90-94, Jan. 1995.

[47] A.K. Jain, L. Hong, S. Pankanti, and R. Bolle, "An Identity-Authentication System Using Fingerprints," *Proc. of the IEEE*, vol. 85, no. 9, pp. 1365-1388, Sept. 1997.

[48] A.K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice-Hall, 1988.

[49] A.K. Jain, R.P.W. Duin, and J. Mao, "Statistical Pattern Recognition: A Review," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 4-37, Jan. 2000.

[50] A.K. Jain and D. Zongker, "Representation and Recognition of Handwritten Digits using Deformable Templates," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 19, no. 12, pp. 1386-1391, Dec. 1997.

[51] http://www.cic.com

[52] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, pp. 671-680, 1983.

[53] J. Kittler, "On Combining Classifiers," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 20, no. 3, pp. 226-238, March 1998.

[54] T. Kohonen, "The self-organizing map," *Proc. of the IEEE*, vol. 78, no. 9, pp. 1464-1480, Sept. 1990.

[55] T. Kohonen, J. Kangas, J. Laaksonen, and K. Torkkola, " LVQ_PAK: A Program Package for the Correct Application of Learning Vector Quantization Algorithms," *Int. Joint Conference on Neural Networks*, Baltimore, pp/ I725-730, June 1992.

[56] A. Kosmala, J. Rottland, and G. Rigoll, "Improved On-Line Handwriting Recognition Using Context Dependent Hidden Markov Models," *Proc. 4th Int. Conf. Document Analysis and Recognition*, Ulm, Germany, pp. 641-644, Aug. 1997.

[57] K.-F. Lee, "Hidden Markov Models: Past, Present, and Future," *Proc. Eurospeech'89*, vol. 1, Paris, pp. 148-155, Sept. 1989.

[58] S.E. Levinson, L.R. Rabiner, and M.M. Sondhi, "An Introduction to the Application of the Theory of Probabilistic Functions of a Markov Process to Automatic Speech Recognition," *Bell System Technical Journal*, vol. 62, no. 4, pp. 1035-1074, April 1983.

[59] X. Li, R. Plamondon, and M. Parizeau, "Model-based On-line Handwritten Digit Recognition," *Proc. 14th Int. Conf. Pattern Recognition*, Brisbane, Australia, pp. 1134-1136, Aug. 1998.

[60] X. Li and D.-Y. Yeung, "On-line Handwritten Alphanumeric Character Recognition Using Dominant Points in Strokes," *Pattern Recognition*, vol. 30, no. 1, pp. 31-44, Jan. 1997.

[61] X. Li, M. Parizeau, and R. Plamondon, "Segmentation and Reconstruction of On-Line Handwritten Scripts," *Pattern Recognition*, vol. 31, no. 6, pp. 675-684, June 1998.

[62] S. Madhvanath and V. Govindaraju, "Serial Classifier Combination For Handwritten Word Recognition," *Proc. 3rd Int. Conf. Document Analysis and Recognition*, vol. II, Montreal, Canada, pp. 911-914, Aug. 1995.

[63] E. Mandler, R. Oed, and W. Doster, "Experiments In On-Line Script Recognition," *Proc. 4th Scandinavian Conf. Image Analysis*, pp. 75-86, June 1985.

[64] S. Manke, M. Finke, and A. Waibel, "NPen++" A Writer Independent, Large Vocabulary On-Line Cursive Handwriting Recognition System," *Proc. of the 3rd Int. Conf. Document Analysis and Recognition*, Montreal, Canada, pp 403-408, Aug. 1995.

[65] S. Manke and U. Bodenhausen, "A Connectionist Recognizer for On-Line Cursive Handwriting Recognition," *Proc. ICASSP'94*, vol. 2, Adelaide, Australia, pp. 633-636, April 1994.

[66] S.S. Marwah, S.K. Mullick, R.M.K. Sinha, "Recognition of Devanagari Characters Using A Hierarchical Binary Decision Tree Classifier," *Proc. IEEE Int. Conf. Systems, Man and Cybernetics*, New York, pp. 414-20, 1984.

[67] G. McLachlan and T. Krishnan, *The EM Algorithm and Extensions*, John Wiley & Sons, New York, 1997.

[68] K. Mohiuddin and J. Mao, "A Comparative Study of Different Classifiers For Handprinted Character Recognition," in *Pattern Recognition in Practice IV*, E.S. Gelsema and L.N. Kanal, eds., Elsevier Science, The Netherlands, pp. 437-448, 1994.

[69] G. Nagy, "Twenty Years of Document Image Analysis in PAMI," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 38-62, Jan. 2000.

[70] V.S. Nalwa, "Automatic On-Line Signature Verification," *Proc. of the IEEE*, vol. 85, no. 2, pp. 215-239, Feb. 1997.

[71] K.S. Nathan, J.R. Bellegarda, D. Nahamoo, and E.J. Bellegarda, "On-Line Handwriting Recognition Using Continuous Parameter Hidden Markov Models," *Proc. ICASSP'93*, vol. 5, Minneapolis, MN, pp. 121-124, May 1993.

[72] K.S. Nathan, H.S.M. Beigi, H. Subrahmonia, G.J. Clary, and H. Maruyama, "Real-Time On-Line Unconstrained Handwriting Recognition Using Statistical Methods," *Proc. ICASSP'95*, vol. 4, Detroit, MI, pp. 2619-2623, May 1995.

[73] F. Nouboud and R. Plamondon, "On-Line Recognition of Handprinted Characters: Survey and Beta Tests," *Pattern Recognition*, vol. 25, no. 9, pp. 1031-1044, Sept. 1990.

[74] L. O'Gorman and R. Kasturi, *Document Image Analysis*, IEEE Computer Society Press, 1995.

[75] Palm Computing, "Graffiti Reference Card," [Online] Available http://www.3com.com, Oct. 1999.

[76] http://www.paragraph.com

[77] I. Pavlidis, R. Singh, and N.P. Papnikolopoulos, "On-Line Handwriting Recognition Using Physics-Based Shape Metamorphosis," *Pattern Recognition*, vol. 31, no. 11, pp. 1589-1600, Nov. 1998.

[78] M.P. Perrone and S.D. Connell, "K-Means Clustering For Hidden Markov Models," *to appear in 7th Int. Workshop on Frontiers in Handwriting Recognition*, Amsterdam, The Netherlands, Sept. 2000.

[79] R. Plamondon, D. Lopresti, L.R.B. Schomaker, and R. Srihari, "On-Line Handwriting Recognition," *Encyclopedia of Electrical and Electronics Engineering*, J.G. Webster, ed., vol. 15, pp. 123-146, Wiley, 1999.

[80] R. Plamondon and F.J. Maarse, "An Evaluation of Motor Models of Handwriting," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 19, no. 5, pp. 1060-1072, May 1989.

[81] 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.

[82] L. Prevost and M. Milgram, "Automatic Allograph Selection and Multiple Classification for Totally Unconstrained Handwritten Character Recognition," *Proc. 14th Int. Conf. Pattern Recognition*, Brisbane, Australia, pp. 381-383, Aug. 1998.

[83] L. Prevost and M. Milgram, "Non-supervised Determination of Allograph Subclasses for On-line Omni-scriptor Handwriting Recognition," *Proc. 5th Int. Conf. Document Analysis and Recognition*, Bangalore, India, pp. 438-441, Sept. 1999.

[84] W. Postl, "Detection of Linear Oblique Structures and Skew Scan in Digitized Documents," *Proc. 8th Int. Conf. Pattern Recognition*, Paris, France, pp. 687-689, Oct. 1986.

[85] J.R. Quinlan, "Bagging, Boosting, and C4.5," *Proc. 13th National Conference on Artificial Intelligence*, Portland, OR, Aug. 1996.

[86] J.R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kauffman, 1993.

[87] L. R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proc. of the IEEE*, vol. 77, no. 2, pp. 257-286, Feb. 1989.

[88] A. Rangarajan, "Self Annealing." in *Energy Minimization Methods in Computer Vision and Pattern Recognition*, M. Pellilo and E. Hancock, eds., pp. 229-244, Springer Verlag, 1997.

[89] G. Rigoll, A. Kosmala, and D. Willett, "A New Hybrid Approach to Large Vocabulary Cursive Handwriting Recognition," *Proc. 14th Int. Conf. on Pattern Recognition*, Brisbane, Australia, pp. 1512-1514, Aug. 1998.

[90] K. Rose, E. Gurewitz, and G. Fox, "A Deterministic Annealing Approach to Clustering," *Pattern Recognition Letters*, vol. 11, no. 11, pp. 589-594, Nov. 1990.

[91] P. Scattolin and A. Krzyzak, "Weighted Elastic Matching Method for Recognition of Handwritten Numerals," Vision Interface '94, pp. 178-185, 1994.

[92] M. Schenkel, I. Guyon, and D. Henderson, "On-Line Cursive Script Recognition Using Time Delay Neural Networks and Hidden Markov Models," *Proc. ICASSP'94*, vol. 2, Adelaide, Australia, pp. 637-640, April 1994.

[93] L.R.B. Schomaker and H.-L. Teulings, "A Handwriting Recognition System based on the Properties and Architectures of the Human Motor System," *Proc. Int. Workshop on Frontiers in Handwriting Recognition*, Montreal, pp. 195-211, April 1990.

[94] L.R.B. Schomaker, E.H. Helsper, H.L. Teulings, and G.H. Abbink, "Adaptive recognition of online, cursive handwriting," *Proc. 6th Int. Conf. Handwriting and Drawing*, Paris, France, pp. 19-21, July 1993.

[95] G. Seni, R.K. Srihari, and N. Nasrabadi, "Large Vocabulary Recognition of On-Line Handwritten Cursive Words," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 18, no. 7, pp. 757-762, July 1996.

[96] A.W. Senior, J. Subrahmonia, and K. Nathan, "Duration Modeling Results For An On-Line Handwriting Recognizer," *Proc. ICASSP'96*, Atlanta, Georgia, pp. 3483-3486, May 1996.

[97] R.E. Shapire, Y. Freund, P. Bartlett, and W.S. Lee, "Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods," *The Annals of Statistics*, vol. 26, no. 5, pp. 1651-1686, 1997.

[98] R.M.K. Sinha and V. Bansal, "On Devanagari Document Processing," *Proc. Int. Conf. on Systems, Man and Cybernetics*, Vancouver, BC, pp. 1621-1626, Oct. 1995.

[99] R.M.K. Sinha and H.N. Mahabala, "Machine Recognition of Devanagari Script," *IEEE Trans. Systems, Man and Cybernetics*, vol. 9, no. 8, pp. 435-441, Aug. 1979.

[100] J. Subrahmonia, K. Nathan, and M. Perrone, "Writer Dependent Recognition of On-Line Unconstrained Handwriting," *Proc. ICASSP'96*, vol. 6, Atlanta, Georgia, pp. 3478-3481, May 1996.

[101] J. Makhoul, T. Starner, R. Schwartz, and G. Chou, "On-Line Cursive Handwriting Recognition Using Speech Recognition Methods," *Proc. ICASSP'94*, vol. 5, Adelaide, Australia, pp. 125-128, April 1994.

[102] H. Takahashi, "A Neural Net OCR Using Geometrical and Zonal-Pattern Features," *Proc. 1st Int. Conf. Document Analysis and Recognition*, pp. 821-828, 1991.

[103] C.C. Tappert, "Adaptive On-Line Handwriting Recognition," *Proc. 7th Int. Conf. on Pattern Recognition*, Montreal, Canada, pp. 1004-1007, July-Aug. 1984.

[104] C.C. Tappert, "Cursive Script Recognition by Elastic Matching," *IBM Journal of Research and Development*, vol. 26, no. 6, pp. 765-771, June 1982.

[105] C.C. Tappert, C.Y. Suen, 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.

[106] H. Teulings and L. Schomaker, "Unsupervised Learning of Prototype Allographs in Cursive Script Recognition," *From Pixels to Features III: Frontiers in Handwritten Recognition*, S. Impedovo and J. Simon, eds., North-Holland, pp. 61-73, 1992.

[107] N. Udea and R. Nakano, "Deterministic Annealing EM Algorithm," *Neural Networks*, vol. 11, pp. 271-282, 1998.

[108] A.J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimal decoding algorithm," *IEEE Trans. Information Theory*, vol. IT-13, pp. 260-269, 1967.

[109] L. Vuurpijl and L. Schomaker, "Finding Structure in Diversity: A Hierarchical Clustering Method for the Categorization of Allographs in Handwriting," *Proc. 4th Int. Conf. Document Analysis and Recognition*, Ulm, Germany, pp. 387-393, Aug. 1997.

[110] T. Wakahara, "On-Line Cursive Script Recognition Using Local Affine Transformation," *Proc. 9th Int. Conf. Pattern Recognition*, pp. 1133-1137, Nov. 1988.

[111] T. Wakahara, "Adaptive Normalization of Handwritten Characters Using Global/Local Affine Transformation," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 20, no. 12, pp. 1332-1341, Dec. 1998.

[112] T. Wakahara, H. Murase, and K. Odaka, "On-Line Handwriting Recognition," *Proc. of the IEEE*, vol. 80, no. 7, pp. 1181-1194, July 1992.

[113] K.Y. Wong, R.G. Casey, and F.M. Wahl, "Document Analysis System," *IBM J. Research and Development*, vol. 26, no. 6, pp. 647-656, Nov. 1982.

[114] L.S. Yaeger, B.J. Webb, and R.F. Lyon, "Combining Neural Networks and Context-Driven Search for Online, Printed Handwriting Recognition in the Newton," *AI Magazine*, pp. 73-89, Spring 1998.

[115] K. Yamasaki, "Automatic Allograph Categorization Based on Stroke Clustering for On-Line Handwritten Japanese Character Recognition," *Proc. 14th Int. Conf. Pattern Recognition*, Brisbane, Australia, pp. 1150-1152, Aug. 1998.