On the Recognition of Handwritten Mathematical Symbols

by

Xiaofang Xie

Graduate Program
in
Computer Science

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

Faculty of Graduate Studies
The University of Western Ontario
London, Ontario
Dec 2007

© Xiaofang Xie 2007

THE UNIVERSITY OF WESTERN ONTARIO
FACULTY OF GRADUATE STUDIES

CERTIFICATE OF EXAMINATION

Chief Advisor	Examining Board
Stephen M. Watt	Masakazu Suzuki
	Greg Reid
Advisory Committe	
	Mark Perry
	Marc Mareno Maza

The thesis by
Xiaofang Xie

entitled

On the Recognition of Handwritten Mathematical Symbols

is accepted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Dec. 14, 2007	Xing Jiang
Date	Chairman of Examining Board

ABSTRACT

We have examined the problem of machine recognition of handwritten mathematical symbols. We focus on the case where ink-stroke information is available, as it would be collected from a digital pen. We have examined a number of problems: handwriting variant analysis, feature extraction, grouping sets of characters, encoding handwritten mathematical symbols and building recognizers.

One of the difficulties of handwritten mathematical symbol recognition lies in the variability of the symbols. We have performed handwriting variance analysis and identified the factors contributing to the variants. Based on the analysis of 800M data in a format that includes symbol names, start time, end time, x and y coordinates and pressure, we developed an allomorph set for each mathematical symbol. We then used them to build models. We have examined and selected different features of handwritten mathematical symbols and proposed new algorithms for feature extraction are proposed. For well-known features such as loops, our algorithms can perform better than the algorithms in the literature. We group the large set of mathematical symbols according to different features. The prototypes for comparison are pruned by comparing within each group instead of on the whole symbol set. The implementation of prototype pruning in the elastic recognizer has resulted in improved recognition speed.

We have designed two encoding schemes: encoding the handwritten symbols based on the curvature or encoding the handwritten symbols with equal length. The encoded segments are associated with the states of a hidden Markov model (HMM). Separately, we have designed a multi-path and multi-model HMM topology and integrated inter-stroke and space information into HMM. Our HMM topology achieves better recognition rate than earlier HMM.

We have designed and implemented a handwritten mathematical symbol recognition system that includes two recognizers: an elastic matching-based recognizer and a hidden Markov model-based recognizer. The architecture includes data preprocessing, data analysis, symbol representation and recognition.

Key words: Handwriting Recognition, Feature Extraction, Symbol Encoding/Decomposition, Variance Analysis, Elastic Matching, Hidden Markov Model, Vector Quantization, Subspace .

To my parents and my husband . . .

ACKNOWLEDGEMENTS

I would like to express my appreciation to my supervisor, Dr. Stephen Watt for providing me with the opportunity to work on this topic and for his guidance and encouragement throughout years of my studies. I feel especially indebted to the friendly people of the Symbolic Computation Laboratory (SCL) for the spirit of collaboration being instrumental in the completion my thesis. Special thanks go to Jack Polihronov for reading my thesis. I would like to acknowledge the help of the departmental secretaries during the years of my study in the Computer Science department. Words could not express my thankfulness to my dear husband Long for his love, support and encouragement.

TABLE OF CONTENTS

CERTIFICATE OF EXAMINATION **ii**

ABSTRACT **iii**

DEDICATION **v**

ACKNOWLEDGEMENTS **vi**

TABLE OF CONTENTS **vii**

LIST OF TABLES **xii**

LIST OF FIGURES **xiii**

Chapter 1 Background and Motivation **1**

- 1.1 Motivation 1
- 1.2 Statement of the Problem 3
- 1.3 Review of Handwritten Character Recognition 4
- 1.4 Review of Handwritten Mathematics Recognition 5
- 1.5 Main Innovative Elements of the Present Work 6
- 1.6 Outline of the Thesis 7

Chapter 2 Overview of Handwritten Mathematics Recognition **9**

- 2.1 Difficulties 9
- 2.2 Recognition Process 10

2.3 Segmentation of Symbols 10

2.4 Recognition of Segmented Symbols 11

2.5 Structural Analysis 13

Chapter 3 Handwriting Recognition Components of the Current Work 15

3.1 Architecture of the Handwriting Recognition System 15

3.2 Other Approaches for Handwriting Recognition 16

Chapter 4 Experimental Data for Mathematical Handwriting Recognition 19

4.1 Data Collected at the ORCCA Lab 19

  4.1.1 Data Collected with an IBM CrossPad 19

  4.1.2 Data from Tablet PC 20

  4.1.3 The ORCCA Data Sample 21

  4.1.4 Data from Unipen 21

4.2 Different Digital Ink Format Conversion 22

Chapter 5 Preprocessing 24

5.1 Dehooking 24

5.2 Smoothing 26

5.3 Re-sampling 28

5.4 Stroke Re-direction 29

5.5 Stroke Re-ordering 30

5.6 Size Normalization 31

Chapter 6 Mathematical Handwriting Variant Analysis 33

6.1 Introduction 33

6.2 Variant Analysis 33

  6.2.1 Number of Strokes 34

  6.2.2 Beginning and Ending Hooks 34

6.2.3 Cursively Connecting Segments 35
6.2.4 Cusps Changing to Small Loops 35
6.2.5 Angles Changing to Loops 36
6.2.6 Omitting Character Tails 36
6.2.7 Wavy Lines 37
6.2.8 Quick and Cursive Handwriting 37
6.3 Factors Which Are Not Considered 38
6.4 Allomorph Variation Among Characters 38
6.5 Using of Variance Analysis in Handwriting Recognition 38

Chapter 7 Feature Extraction 40

7.1 Introduction 40
7.2 Published Studies on Features of Written Symbols 40
7.3 Feature Families 41
7.3.1 Geometric Features 42
7.3.2 Ink-Related Features 47
7.3.3 Directional Features 48
7.3.4 Global Features 49

Chapter 8 Prototype Pruning in Elastic Recognition 50

8.1 Prototype Pruning by Feature Extraction 50
8.2 Elastic Matching 51
8.3 Experimental Results 53
8.4 Conclusions 57

Chapter 9 Hidden Markov Model Based Recognition 58

9.1 Introduction 58
9.2 Elements of a Hidden Markov Model 60
9.2.1 Discrete Markov Process 60

9.2.2 Hidden Markov Models 61
9.2.3 The Three Basic Problems of HMM 62
9.2.4 Solutions to the Three Basic Problems of HMM 63

Chapter 10 A HMM for Mathematical Characters 67
10.1 The Architecture of Our Hidden Markov Model Recognizer 67
10.2 Modeling of Handwritten Mathematical Symbols 69
10.2.1 Decomposing Mathematical Symbols 69
10.2.2 Representing Handwritten Symbols by Using Features 76
10.3 Calculating Observation Symbols 82
10.4 HMM Topology 85
10.4.1 Number of States 85
10.4.2 Initial Observation Distribution 86

Chapter 11 Implementation, Experiments and Results 89
11.1 Implementation of the HMM Based Recognizer 89
11.1.1 Ink Processing 89
11.1.2 Ink Representation 91
11.1.3 Model Parameter Selection and Implementation 92
11.1.4 Scaling 93
11.1.5 Training 94
11.1.6 Recognition 94
11.2 The Mathematical Symbol Database Used in the Present Work 95
11.3 Experiments and Results 97

Chapter 12 Other Recognition Methodologies 102
12.1 Introduction 102
12.2 Subspace Classification 102
12.3 Implementation of a Subspace Recognizer 104

12.3.1 How to Build the Subspace 104

12.3.2 Recognition 104

12.4 Discussion 105

Chapter 13 Combining Recognizers and Applying Context Rules 107

13.1 Combination of Recognizers 107

13.2 Individual Classifier Generation 107

13.3 Combination Schemes 109

13.4 Combining Dictionary-Based Prediction with Recognition 111

13.4.1 Building Database of Mathematical Context 112

13.4.2 Predicting Characters from the Mathematical Context 113

13.4.3 Combining Prediction and Recognition 114

Chapter 14 Conclusion and Future Work 115

14.1 Conclusion 115

14.2 Future Work 116

Appendix A Samples of Allomorph of Handwritten Symbols 118

Appendix B Dynamic Programming for Elastic Matching 124

REFERENCES 128

VITA 136

LIST OF TABLES

8.1 Results in the case when features are not used 56
8.2 Results in the case when features are used 56
8.3 Comparison with Kurtzberg's Results 56

11.1 ORCCA Mathematical Symbol Data Set 96
11.2 Dynamic States vs. Constant States: The Case of Positional Feature 99
11.3 Dynamic States vs. Constant States: The Case of Chain-Code Based Features 99
11.4 Combining Feature Sets 100
11.5 Different Code Book Sizes: The Case of positional features 100
11.6 Different Code Book Sizes: The Case of Chain Code Based Feature . 100
11.7 Gaussian Distribution vs. Random Distribution: The Case of Chain Code Based Feature 100
11.8 Number of States: The Case of 7 States, 5 States and 3 States 101

13.1 Subscripts of $\sum$, from [54] 114

LIST OF FIGURES

2.1 Similarity of Mathematical Symbols 10

2.2 Ambiguity of Mathematical Expression (from [75]) 10

2.3 X-Y Projection Cut, from Ha [24] 11

2.4 Elastic Matching 12

2.5 Markov Model 13

2.6 Structural Analysis 13

2.7 Indistinguishable Characters by the Projection Profile Cutting method 14

3.1 Architecture of the Handwriting Recognition System 18

4.1 Tablet PC Data Sample 22

5.1 Examples of Hooks 25

5.2 Definition of theta in the case of hooks 25

5.3 (a) before dehooking (b) after dehooking 26

5.4 (a) the symbol “slash” before dehooking (b) after dehooking 26

5.5 (a) before dehooking (b) after dehooking 27

5.6 (a) the symbol “plus” before dehooking (b) after dehooking 27

5.7 (a) before smoothing 27

5.8 (b) after average smoothing (c) after Gaussian smoothing 28

5.9 Stroke Re-ordering, from Nicholas [44] 31

6.1 Different Number of Strokes 34
6.2 Hooks 35
6.3 Connected Strokes 35
6.4 Cusps becoming Loops 36
6.5 Angles becoming Loops 36
6.6 Omitting Little Tails 37
6.7 Straight Line Variance 37
6.8 Simplified Symbols 37
6.9 Loop to Cusp Example 38
6.10 Writing Style Examples 39

7.1 (a) Cusps in "h" (b) Cusps 44
7.2 (a) Example 1 for Loop Detection (b) Example 2 for Loop Detection 45
7.3 Modified Sweepline Algorithm 45
7.4 Point Density 47
7.5 Chain Code 48
7.6 Initial-End Direction 48
7.7 Writing Angle 49
7.8 Initial-End Position 49

8.1 Elastic Matching (from [66]) 53
8.2 Symbols for Experiments 54

9.1 Markov Chain with Four States 60
9.2 Three State HMM 62

10.1 HMM Recognition Diagram 68
10.2 Chinese Strokes (from [77]) 69
10.3 Curve-based Elements 70

10.4 Curve-based Decomposed "3" 73
10.5 Curve-based Decomposed "β" 73
10.6 Encoding Elements 74
10.7 Equal Length Decoding of the Symbol "3" 75
10.8 Equal Length Decoding of the Symbol "β" 75
10.9 Pendown Features 76
10.10 Penup Features 76
10.11 Spatial Relation 78
10.12 HMM and Initial Distribution for Symbol "α" 80
10.13 Combined HMM 81
10.14 Combined Features 81
10.15 2D Vector Quantization 85
10.16 HMM States and Segments 86
10.17 Combined HMM 88

11.1 Diagram of the Major Classes of the Recognizer (1) 90
11.2 Diagram of the Major Classes of the Recognizer (2) 91

12.1 Projection into Subspace 104
12.2 Build Subspace 105
12.3 Subspace Recognition 105

13.1 Context - Ambiguity in Mathematics(1) 112
13.2 Context - Ambiguity in Mathematics(2) 112
13.3 Context - Ambiguity in Natural Language, from [54] 113
13.4 Trie for Symbol "i", from [54] 113

Chapter 1

Background and Motivation

1.1 Motivation

Handwriting recognition has been studied for about fifty years [31] [63] and has been used in hand-held devices [66], post office scanners *etc* [38]. With the recent development of electronic tablets, pen-tip movement can be captured more accurately, making possible the capture of not only x, y coordinates, but pen pressure as well. Pen-based computing combined with handwriting recognition has become an important research topic and has attracted significant attention in the last few decades, being considered as a key development needed for the next generation of PDAs and tablet PCs.

Handwritten mathematics recognition is an important and largely unaddressed branch of handwriting recognition due to the following considerations:

- Mathematics inputting and editing have posed problems due to their two dimensional structure and large sets of symbols. These typically consist of special symbols and Greek letters in addition to English letters and digits. The commonly used keyboard input is thus insufficient for the input of such a large set of symbols, which has led to the desire for other input methods. The most widely used such system is $\LaTeX$. A set of keywords is defined in $\LaTeX$ for the representation of special symbols and characters. However, using $\LaTeX$ may be considered unattractive as it requires the memorization of a number of commands. On the other hand, using pen-based computing technologies, one could simply write mathematical expressions on a tablet and use recognition software to typeset them. Suzuki *et al.* [19] have already designed and implemented such mathematics editors with handwriting interface. Fateman and Zhang [78] have proposed a design involving pen and voice.

- Most scientific and engineering publications contain mathematical symbols and expressions. Recognition of handwritten mathematics would not only require less effort in writing technical documents but could also be used to transfer existing handwritten documents into electronic format and between machines when needed. Therefore handwritten mathematics recognition is one of the key forces that drive the information transformation between human and machine and among machines.
- Recognition of mathematical symbols could provide a friendly user interface for computer algebra systems. For example, Maple has a handwriting interface.
- Recognition of mathematical symbols can be expanded towards the fields of electronics circuitry, chemistry, *etc.*

The Ontario Research Center for Computer Algebra (ORCCA) lab has initiated a research project on mathematics editing, rendering and recognition in 1999 under the leadership by Stephen Watt. The related work includes organizing pen-based environments for processing handwriting [56], communicating mathematics via pen-based computer interfaces [53], building context for pen-based computing [52] *etc.* Various modules were completed by their respective authors within ORCCA framework:

- Arthur Louie [41] has completed the mathematical handwriting analysis for MathML generation.
- Bo Wan [66] has implemented a handwriting recognizer for Pocket PCs.
- Clare So [58] has calculated the frequencies of symbols in large sets of mathematical expressions. This has been used by Smirnova and Watt for prediction in mathematical handwriting recognition [59].
- Kevin Durdle [16] has completed an analysis of digital ink framework to support handwriting recognition.

- Xiaojie Wu [73] studied various digital ink formats and transformation between them.

Other work has included contributions to InkML [29] [71], representing handwritten mathematical symbols through mathematical functions [11], mathematical symbol recognition using support vector machine [28] and mathematical expression analysis for Arabic handwriting [55].

The goal of the present thesis is to analyze different aspects of handwriting recognition and build tools that can operate with large sets of handwritten mathematical symbols.

1.2 Statement of the Problem

We examined the problem of machine recognition of handwritten mathematical symbols. We concentrated on the following areas: reducing the amount of computation, identifying discriminative features between symbols and building recognition models.

Existing recognizers for, *e.g.*, English, Chinese, Japanese and numbers have achieved reasonable processing rates based on small sets of symbols. In Asian languages there are many "characters", but only small sets of strokes. When the lexicon increases, it is challenging to achieve high accuracy and speed. One goal should be to reduce the amount of computation for recognition when a large number of symbols is used. One way to do this would be to group handwritten mathematical symbols into classes, once proper criteria for such grouping have been found. Unknown symbols would first be placed into a group, followed by recognition within the group instead of comparing with the whole set of symbols. One goal of the present work is to identify those characteristics that may be used to separate characters into classes effectively.

Feature extraction can be an important step for recognition, since features distinguish the individual characters. So far, researchers have used a wide range of different features, such as geometry features, global features and so on [27] [47]. One goal of the present work is to analyze and discover features that can best represent handwritten characters. We have decomposed each individual symbol into basic elements.

Raw recognition may then be performed on the basic elements.

The design of hidden Markov models is another key for the success of recognition systems. For this purpose, one needs to determine the proper number of states, the number of observation symbols and the meaning associated with the states. More importantly, the structure of a hidden Markov model is an issue that should also be taken into consideration. We shall give details of these problems and their solutions in later chapters of this thesis.

1.3 Review of Handwritten Character Recognition

Handwriting recognition inherited a number of technologies from optical character recognition (OCR). The main difference between handwritten and typewritten characters is in the variations that come with handwriting. It is also worth noticing that OCR deals with off-line recognition while handwriting recognition may be required for both on-line and off-line signals. (On-line means that data is captured as it is written. For off-line, all the data is collected before processing starts.) Off-line processing is able to use only snapshots of the handwriting without time information. From off-line data, we do not have information on the order of strokes the user has used to write the character. Normally, the input of on-line handwriting consists of traces while the off-line handwriting recognition deals with images. We focus on the on-line problem. While the input of OCR is usually entire documents, the input of handwriting recognizers can be entire handwritten documents or small snippets of digital ink.

Work in this area began as early as the 1950s [62]. In 1950, David Shepard, worked on transferring printed messages into machine language for computer processing in the United States National Security Agency. In 1951, he founded Intelligent Machines Research Corporation (IMR), which delivered the world's first several commercial OCR systems. Since late 1950's, OCR software has been used widely in post offices, banks *etc.* Nowadays, there are a number of publications and software products that claim high OCR recognition rates. Yet the failure of certain real applications shows that the performance problems subsist on composite and degraded documents and there is still room for progress.

The challenge in OCR research is to develop robust methods that remove as much as possible the noise restriction while maintaining high recognition rate [10] [25] [63].

The research activity in handwriting recognition was intense in late 1950's and 1960's, ebbed in the 1970's, and renewed in the 1980's. With the hardware development and better algorithms, it has attracted more attention recently. The first handwriting recognizers successfully used for PDAs were Calligrapher (ParaGraph Inc.) and Graffiti (Palm Inc.). Calligrapher lost market soon after its introduction due to low recognition rate and high price. Graffiti used one stroke to form each character, so users had to be trained to know how to write on the device. The Graffiti alphabet is similar to the earlier Unistroke alphabet of Xerox (and indeed this has led to contention between the companies). Today better recognizers are used by PDAs. For example, the transcriber of Microsoft does not require segmented letters so that users can write naturally. Most commercial recognizers achieve high recognition rate because they use a small set of characters. The achievement of real-time recognition rates while using large sets of symbols is still a challenging research topic [63].

1.4 Review of Handwritten Mathematics Recognition

Handwritten mathematics recognition has been studied for over 30 years [10]. As mathematical expressions appear in large number of scientific documents, without doubt transferring such documents into electronic format requires utilities for recognition of mathematical content. Handwriting input provides natural and convenient way of inputting mathematical text into computer for storage or sharing with others, once again underlining the necessity of an effective mathematical recognition software. The research in handwritten mathematics recognition is driven by a desire to combine the natural advantages of handwritten input with the data processing capabilities of computers [44].

The problem of recognizing handwritten mathematics is significantly different from natural language recognition. There is no pre-defined context which gives constraints to possible symbols. Mathematical symbols are more complicate than Chinese in terms of variation of stroke directions and retraces. Two dimensional structure together with implicit operators makes mathematical expression recognition a challenging problem.

Handwritten mathematics recognition typically consists of symbols recognition and structure analysis. These two stages can go either together or separately. Some researchers, for example, Koschinski *et al.* [33] focus only on the symbol recognition, while other researchers, *e.g.* Blostein *et al.* have been worked on mathematical expression recognition. Symbol recognition methods include template matching [9], statistical approaches [57], neural networks [45], Markov models [37], *etc.* Structural analysis methods have two main directions: syntactic parsing [4] and graph rewriting [20]. In syntactic parsing, a set of syntactic rules is built to parse the expressions. Graph rewriting method treats mathematical expressions as graphs and certain graph reductions are applied to the expressions.

1.5 Main Innovative Elements of the Present Work

The present thesis describes the design and implementation of a handwritten mathematical symbol recognition system. It includes two recognizers: elastic matching based recognizer and hidden Markov model-based recognizer. Our main results are:

- Allomorphs for 270 mathematical symbols have been presented. Allomorphs are the representatives of variations in handwritten symbols. The allomorphs are used in the models for the elastic matching based recognizer. The hidden Markov model also uses the allomorphs for training purposes.
- A set of 12 features which represent the handwritten mathematical symbols has been defined and analyzed. The features are: number of loops, number of self- and intra-stroke intersections, number of cusps, number of strokes, point density, pen-down/pen-up, initial direction, end direction, initial-end direction, writing angle, initial/end position and width-height ratio.

- New algorithms for finding loops and intersections in handwritten symbols have been presented.
- By grouping the large set of symbols according to different features, we have pruned the prototypes by 88.5% and the recognition speed has been improved.
- Two efficient encoding/decomposing schemes have been presented and have been utilized in a hidden Markov model.
- We have designed a multi-path, multi-model HMM topology. The hidden Markov model for each character has multi-path according to its variation. Except for tempo information, inter-stroke information is integrated into HMM. Individual models have been built according to different features, then the models have been combined to form a final model. Better recognition rate has been achieved from this topology.
- A mapping between the states of HMM and the decomposed segments has been created. A Gaussian distribution for the observation sequences in each state has been built based on the mapping, which achieves better recognition result compared to the cases when random and uniform distribution are used.

1.6 Outline of the Thesis

In Chapter 1 and Chapter 2, we describe the motivation of our research and reviewed the completed work in the area of handwritten mathematical recognition. Chapter 3 describes the architecture of our recognition system. Chapter 4 provides a description of the data collection work performed in ORCCA lab including data collection procedures across different tablets such as IBM cross pad and Microsoft tablet PC. Details on the data converters between different digital ink formats are also presented. Chapter 5 outlines the pre-processing performed in the proposed recognition system, while Chapter 6 describes the handwriting variance analysis, literature review as well as an explanation of our approach and a presentation of allograph for mathematical symbols. Chapter 7 defines some features in handwriting recognition including geometric features, ink related features, directional features and global features. The corresponding feature extraction algorithms are described in this chapter as well. Chapter 8 presents the results of prototype pruning using some features given in Chapter 7. Chapter 9 is a review of the theory of hidden Markov Model. Chapter 10 introduces hidden Markov model for mathematical characters. Chapter 11 describes the implementation of a HMM recognizer, also provides experimental results. Chapter 12 introduces recognition methodology based on subspace analysis and presents the implementation of an subspace recognizer together with discussion of this recognition methodology. Chapter 13 provides an overview of combination of recognizers as well as a description of dictionary based handwritten mathematical symbol prediction. Chapter 14 summarizes the thesis results and points to future research directions.

Chapter 2

Overview of Handwritten Mathematics Recognition

In this chapter, we examine the difficulties and processes in handwritten mathematics recognition and review the research in this area.

2.1 Difficulties

Mathematical handwriting differs significantly from other forms of handwriting [33] [55]. First, the set of possible input symbols is extensive, coming from several different alphabets and sources. A full recognizer would have to distinguish about 2000 symbols. Unlike Asian languages with large symbol sets, these symbols are typically composed of a few strokes, with no specific stroke order, and with many symbols being quite similar. For example, φ, g, L, 2, ∂, α, $\propto$, ∞ shown in Figure 2.1. Similar symbols also make expression analysis more difficult. For example, the expression in Figure 2.2 (from [75])can be $(x+3)(y+3)$ or $(x+3xy+3)$. Second, the spatial relation among symbols can use complex context-sensitive two-dimensional rules [11]. Mathematical symbols can be written beside, above, below and inside one another in different sizes. Third, mathematical handwriting can involve large variable operators such as matrix brackets, fraction bars or square roots. This layout and grouping makes mathematical handwriting akin to a blend of drawing and writing. Last, mathematical notation is not formally defined. There are many different communities, each with their own convention. With the evolution of mathematics, new variations and styles are adopted for the notation from time to time.

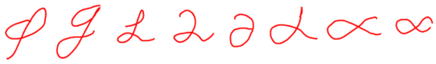


Figure 2.1: Similarity of Mathematical Symbols

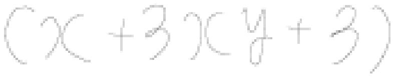


Figure 2.2: Ambiguity of Mathematical Expression (from [75])

2.2 Recognition Process

From the difficulties listed above, we can see that in order to recognize handwritten mathematics well, we need to perform two major tasks: we need to recognize large sets of mathematical symbols and to analyze the two-dimensional mathematical formula structure [9] [76]. We shall focus on the first problem.

2.3 Segmentation of Symbols

The large sets of symbols in mathematical expressions come from different domains and contain different numbers of strokes. For example, "$\sqrt{}$" contains other symbols inside its form. The following is a brief overview of the literature on segmentation technologies: Ha *et al.* [24] used bounding boxes as primitive objects to separate symbols using a "recursive X-Y cut", *i.e.* horizontal projection cut and vertical projection cut, refer to Figure 2.3. This approach must merge or split expressions according to syntax using top down and bottom up cuts.

Okamoto *et al.* [48] used similar segmentation, but used isolated symbol as primitive object.

Smithies *et al.* [57] developed a stroke grouping algorithm to perform segmentation. The system first generated all possible stroke groupings, then examined the confidence level given by a symbol recognizer and selected the best one. This approach is fast but they assumed that any strokes that cross are associated with the same character.

This assumption could introduce errors for overlap strokes from adjacent characters. This requires manual correction.

Faure and Wang presented a modular system for segmenting handwritten mathematical expressions [68]. Their system has two segmentation modules: a data-driven module builds a relation tree for the given expression. Then knowledge-driven segmentation is used for correcting the relation tree.

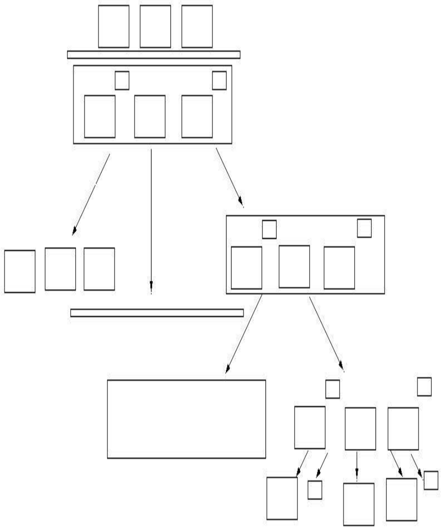


Figure 2.3: X-Y Projection Cut, from Ha [24]

2.4 Recognition of Segmented Symbols

The most common methods used in recognition of handwritten symbols can be classified as follows:

1. Elastic Matching. This method compares the input symbol with symbols in the model. Then the closest distance is computed and the recognition result is the best matched one [26]. (See Figure 2.4)
2. Structural Approaches. Chan [9] *et al.* used this method in their handwritten alphanumeric characters recognition. In this approach, structure information is extracted from the symbols, thereby reducing the number of comparisons.

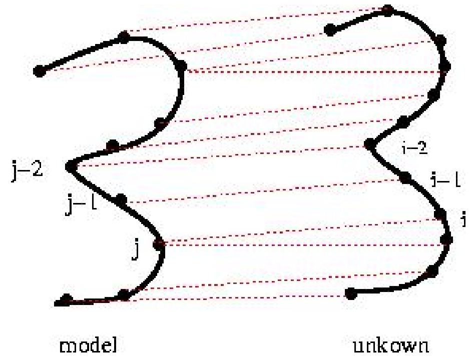
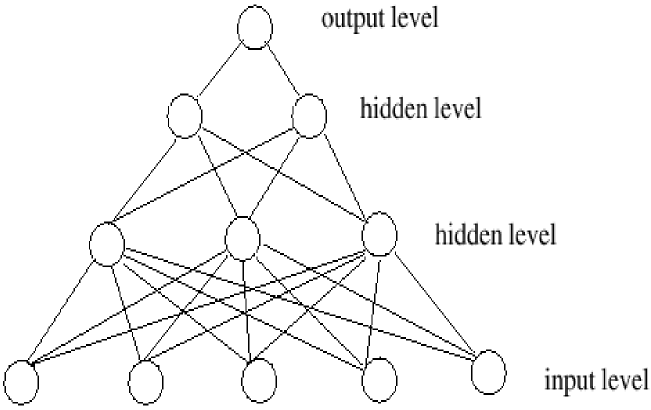
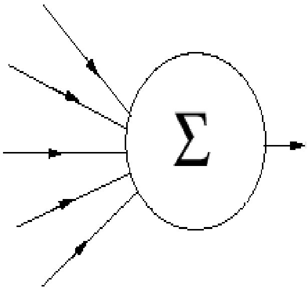


Figure 2.4: Elastic Matching



(a) Neural Network



(b) A Neuron

3. Statistical Approaches. In most of the recognizers, we need to combine certain components in order to get correct results. To recognize segmented characters, we need combine the components that make up a single character. For example, the characters "$i, j, =, !$" each have two strokes. If we have segmented "i" to two strokes, then we would need to combine them for recognition. Smithies *et al.* [57] have used statistical approaches to achieve this.
4. Neural networks. Neural networks have been used successfully in speech recognition. They have been used in handwriting recognition as well. For example Murthy *et al.* [45] applied this method in their handwriting recognition research.
5. Hidden Markov models. This approach does not require the symbols to be segmented before recognition and has been proved to be very effective in the area

of speech recognition. Lee *et al.* [37], Koerich and his colleagues [32], Winker [72] attempted to apply this approach to mathematical expression recognition.

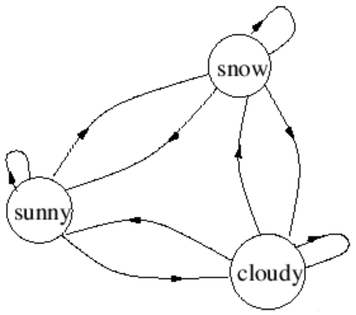


Figure 2.5: Markov Model

2.5 Structural Analysis

The goal of structural analysis is to build a hierarchical relation tree for mathematical expressions. Existing approaches are listed below:

1. Syntactic Methods. Anderson, one of the earliest researchers in this field, used syntactic method [4]. It initiates one ultimate syntactic goal and sub-divides the goal into sub-goals using top-down parsing scheme, followed by the application of certain grammar rules for the division of the goal. A drawback of the method is its low speed due to parsing and poor error handling.

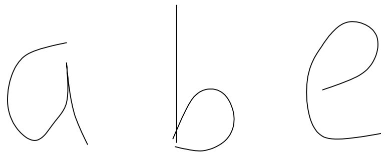


Figure 2.6: Structural Analysis

2. Structure Specification Scheme. This scheme is a restricted version of the syntactic approach. A pattern is divided into several sub-patterns based on the existence of certain operators [76], where each operator has one rule. A disadvantage of this approach is that it can only be applied to the patterns whose

structures are based upon a number of operators. The advantage is that it is more efficient than Anderson's method.

3. Projection Profile Cutting [10]. Vertical cutting is performed first followed by horizontal cutting. The structure analysis is performed prior to symbol recognition. Superscripts, subscripts and square roots require dedicated analysis. A drawback of the method is that it cannot distinguish the expressions as shown in Figure 2.7.

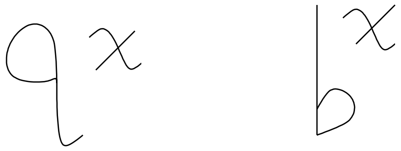


Figure 2.7: Indistinguishable Characters by the Projection Profile Cutting method

4. Graph Rewriting. Blostein *et al.* [20] developed this method for mathematical structural analysis. The information in a mathematical expression is represented by a graph. Then the graph is updated by sets of graph-rewriting rules. The symbols are represented by nodes with attributes representing the location. At the beginning, we only have nodes. Later, we add edges by potential spatial relationship. Recursively, we replace the graph with a new graph by graph rewriting rules. At the end, we only have one node that represents the mathematical expression to be recognized. The advantages of this approach are that there is no backtracking and it has flexible formalism and strong theoretical foundation.
5. Decomposition, based on hierarchy. Chan and Yeung used this method [9], however it needs backtracking. Therefore, it is less efficient comparing with graph rewriting.

In this chapter, we given an overview of handwritten mathematics recognition. We pointed out the difficulties coming from large set of symbols, the similarity between symbols and the complex spatial relations between symbols. We went through the recognition process, *i.e.* segmentation, recognition and structural analysis. Each stage of the process can be performed one after another or simultaneously.

Chapter 3

Handwriting Recognition Components of the Current Work

In this chapter we will give an overview of the components in our handwriting recognition system. The details will be presented in the followed chapters.

3.1 Architecture of the Handwriting Recognition System

This work is part of a larger project for pen-based math, involving expression analysis, dictionary-based methods, and portability layers. In this thesis, we focus on issues related to symbol recognition. Figure 3.1 represents the top-level architecture of our symbol recognition system.

The overall organization includes data collection, preprocessing, feature extraction, prototype pruning, elastic matching recognizer and hidden Markov model recognizer.

In data collection module, a representative set of stroke data for mathematical symbols was collected as traces of (x, y) points, rather than as images. The data collection was performed with the tablet PC and Microsoft tablet SDK. Our database includes mathematical symbols and expressions as well as Unipen data. We include part of Unipen data since thousands of handwritten samples are needed to train our models. A data converter was implemented for the translation of ORCCA data format to Unipen and vice versa.

After collecting the data, various preprocessing methods were applied, such as smoothing, re-sampling, size normalization, *etc.* The preprocessing operations can remove noise caused by ink collection devices or writers. This data was used off-line to develop classification categories for the different mathematical symbols. When we wish to recognize a symbol on-line, given raw stroke data, we perform similar preprocessing, and then send the strokes to a feature extraction coordinator to detect the symbol's features.

A set of discriminative features were used to select the appropriate mathematical symbol class, after which the results of feature extraction were sent to grouping stage, from where the symbols were classified. Grouping reduced computations since the prototypes are pruned.

After grouping, we performed hidden Markov model based recognition as well as elastic matching based recognition. Vector quantization was used to discretize feature vectors and map handwriting input to a sequence of observation symbols. The observation sequence was inputted into a hidden Markov model recognizer. We also tried other recognition method such as subspace method. More details are given in later chapters.

3.2 Other Approaches for Handwriting Recognition

In addition to the above work, we also attempted other approaches in the field of mathematical symbol recognition, which include subspace method, ensembling multiple classifiers and context rules.

There are many methods for pattern recognition. However, not all of them can be used for mathematical symbol recognition due to the difficulties we pointed in Chapter 2. Except for elastic matching and hidden Markov Models, we also explored the subspace method. In subspace method, a pattern is represented by a linear combination from its feature space, which is called the *subspace*. Similar to the hidden Markov model approach, feature vectors have to be extracted from model symbols and unknown symbols. The subspace is built based on the feature vectors so that it represents the pattern and its variations. To recognize an unknown symbol, the distances from unknown feature vector to models' subspaces are calculated, the minimum one is the recognition results.

We built a prototype recognizer using the subspace method. The recognition rate is limited by the choice of method to build the subspace, *i.e.* the linear combination of feature vectors. Further investigation into finding proper subspaces can improve the recognition rate.

Combining multiple classifiers is also an attractive approach. After generating a set of classifiers, various combination schemes can be used for recognition. The widely used combination schemes includes Bayesian combination, score combination, voting, weight voting and Borda count[22]. We studied different ensemble methods including Bagging, AdaBoost, random subspace and architecture variation methods [22].

Ambiguity is one of the difficulties in handwriting recognition. We can rely on context to distinguish the similar symbols. Smirnova and Watt [54] have investigated this area. For completeness, we also summarized their work in this thesis in Section 4, Chapter 13.

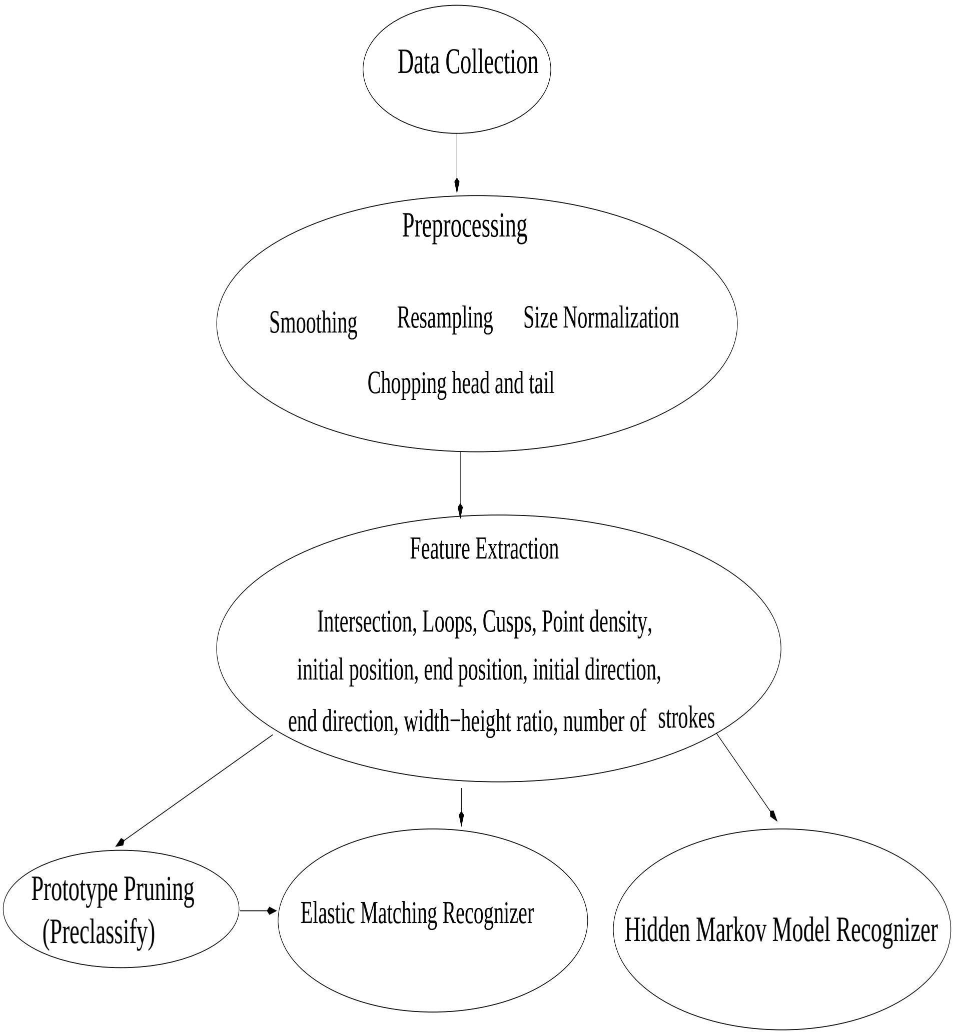

Data Collection
Preprocessing
Smoothing
Resampling
Size Normalization
Chopping head and tail
Feature Extraction
Intersection, Loops, Cusps, Point density,
initial position, end position, initial direction,
end direction, width−height ratio, number of strokes
Prototype Pruning
(Preclassify)
Elastic Matching Recognizer
Hidden Markov Model Recognizer

Figure 3.1: Architecture of the Handwriting Recognition System

Chapter 4

Experimental Data for Mathematical Handwriting Recognition

4.1 Data Collected at the ORCCA Lab

Collection of sufficient handwriting data from different writers is an important part of the recognition process. Many recognition methods such as hidden Markov model based method need substantial amounts of training data. Since standard handwritten mathematics database is not available, we had built our own database.

4.1.1 Data Collected with an IBM CrossPad

Starting in the year 2000, studies on mathematical handwriting recognition at the ORCCA lab have been based on mostly on each authors' own handwriting. In 2002, ORCCA set up the first collection of on-line handwritten samples of mathematical symbols and expressions [16], [73], collected with an IBM CrossPad device. The database was constructed by collecting handwriting samples from 28 volunteers including faculty, graduate and undergraduate students. The mathematical questionnaire used in the study included 301 mathematical characters and 68 mathematical expressions.

The CrossPad is an A4-size portable digital notepad with resolution of 0.01 cm, or 254 dpi (dots per inch). The pen trajectory is recorded 100 times per second as a sequence of points. The data information includes x, y coordinates together with beginning and ending time of the stroke. The x, y coordinates are relative to the original point which is located at the upper-left corner of the notepad.

The CrossPad devices are presently discontinued products and the hardware and software that support these devices are no longer available. It is still possible to handle the data generated by these devices using IBM's Ink Manager and Ink Manager SDK, although the CrossPad's digital ink format is not widely used in handwriting recognition. Therefore, we have converted the CrossPad digital ink into Unipen format.

Time stamps are quite important for on-line handwriting recognition. The CrossPad can record the beginning and ending time of each page and each stroke. The time stamps for each stroke are counted only in seconds, which is useless for most on-line handwriting recognition applications. This limitation is a major drawback of CrossPad devices. Even though the interval time between each points of a stroke can be computed according to the sampling rate, the relation among strokes cannot be accurately captured. Another limitation of CrossPad is that the device is able to record only the pen down trajectory where as it has been shown that pen up trajectory is also important in handwriting recognition.

4.1.2 Data from Tablet PC

Since the IBM CrossPad has inherent limitations a better device was needed for the collection of suitable data. The Tablet PC was be considered to be an appropriate choice. It has an active digitizer (pen-based stylus with a resolution of 600 dpi at least). The Microsoft-developed tablet PC SDK is intended to help users in the development of pen-based applications. We have used an Acer tablet PC in the collection of mathematical handwritten symbols for the second version of the database at ORCCA. The Tablet PC operated under Microsoft XP Tablet Edition and has an appearance of a laptop, enabling the users to write horizontally or vertically on its 14-inch rotary screen. The digitizer of the Tablet PC uses a special stylus sending signals to the display indicating where the stylus is on the screen.

Compared to the IBM CrossPad, the Tablet PC has the following advantages:

- High sampling rate with resolution of about 600 dpi, allowing for more accurate data capture.
- The Tablet PC can detect its stylus status: if pen is off the screen, (*i.e.* pen up stroke), the pen trajectory can also be detected.

- The screen of the Tablet PC is a pressure sensitive device, which permits tracking of pressure data during writing.
- The Tablet PC can capture more detailed pen-related information, such as the x, y tilt angle of stylus.
- Tablet PC is able to provide detailed timing information. Both start and end stroke times can be recorded.

4.1.3 The ORCCA Data Sample

For each symbol and each formula in our questionnaire, we have collected 70 written samples contributed by a group of about 50 individuals. The samples include alphanumeric symbols, Latin letters, Greek letters, scripts and mathematical formulae. The individuals have been selected to have different academic and national backgrounds. Academic backgrounds include mathematics, chemistry, physics, electronical engineering and computer science. The contributors for the data come from Canada, Russia, Japan, China, Iran, France, USA, Romania. The output data file contains character names, Unicode points, x, y coordinates, pressure, a pen-down/pen-up bit as well as time information. Figure 4.1 shows a sample of the data file.

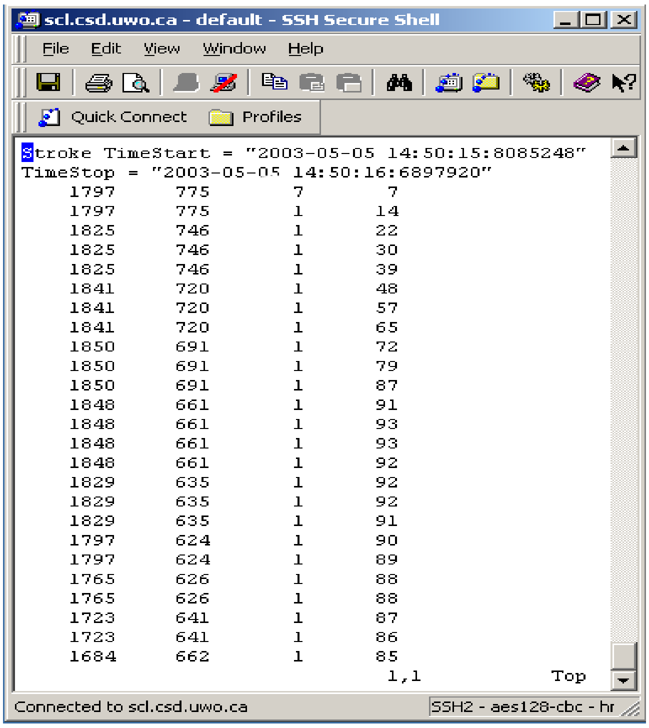
4.1.4 Data from Unipen

Unipen has been first developed in 1993 [23] and is a well-known and widely used digital ink format in the area of handwriting recognition. We have used Unipen data together with our data collection in the training of our recognizers. The Unipen data we used consisted of 16k isolated digits, 28k isolated upper case, 61k isolated lower case and 17k isolated other symbols [23].

4.2 Different Digital Ink Format Conversion

In order to use multiple digital ink formats, we have implemented a number of converters. Since Unipen format is quite widely used, we used Unipen as the standard ink format and have written conversion utilities to and from IBM CrossPad and the Tablet PC formats. This was before InkML and no standard existed. CrossPad ink format was converted into Unipen format using Java and IBM Ink Manager SDK. The conversion is straightforward: starting with the location of the bounding box of each symbol in the "page", followed by recording of the pen trajectory, *i.e.* the x, y coordinates in a Unipen format file.

Converting Unipen format to ORCCA tablet digital ink format requires additional analysis and utilizes the previously prepared database of unicodes of individual symbols in tablet digital ink format. Characters in Unipen format possess only the name of a character, while in ORCCA we use a different symbol naming system, which is platform independent. For example, under ORCCA nomenclature, "a" and "bigA" are the names of small "a" and capital "A" respectively. For this, a map file was prepared containing Unipen character names, corresponding ORCCA nomenclature and



```
Stroke TimeStart = "2003-05-05 14:50:15:8085248"
TimeStop = "2003-05-05 14:50:16:6897920"
    1797      775        7         7
    1797      775        1        14
    1825      746        1        22
    1825      746        1        30
    1825      746        1        39
    1841      720        1        48
    1841      720        1        57
    1841      720        1        65
    1850      691        1        72
    1850      691        1        79
    1850      691        1        87
    1848      661        1        91
    1848      661        1        93
    1848      661        1        93
    1848      661        1        92
    1829      635        1        92
    1829      635        1        92
    1829      635        1        91
    1797      624        1        90
    1797      624        1        89
    1765      626        1        88
    1765      626        1        88
    1723      641        1        87
    1723      641        1        86
    1684      662        1        85
                                   1,1           Top
```

Figure 4.1: Tablet PC Data Sample

unicodes. The conversion between Unipen and ORCCA digital ink was conducted via sequential runs of custom-built parser, syntax checker and Unipen segment parser. The resulting Unipen data has the following structure:

```
.VERSION 1.0
.INCLUDE aga/data/annzchr1.dat
.SEGMENT CHARACTER 77 ? "1"
.SEGMENT CHARACTER 363-364 ? "4"
.SEGMENT CHARACTER 368 ? "6"
.SEGMENT CHARACTER 456 ? "2"
.SEGMENT CHARACTER 458-459 ? "4"
.SEGMENT CHARACTER 460-461 ? "5"
.SEGMENT CHARACTER 463 ? "7"
.SEGMENT CHARACTER 464 ? "8"
```

There are two possible options for the *.INCLUDE* statement, one of them pointing to a file with *.doc* extension which specifies *.ALPHABET*, *HIERARCHY* and other header information. The other possibility is to include a file with a *.dat* extension containing the real stroke information. Sometimes, the *.dat* file contains an *.INCLUDE* statement, which in turn specifies the *.doc* file. Given a Unipen data file, our hierarchy parser can work through the *.include* hierarchy and find the data of pen trajectory. The second argument of *.SEGMENT* uses the expression [A[:M]] - B[:N]],[C] to refer to the actual data segment. Our Unipen segment parser processes this format and extracts the corresponding digital ink.

Converting from ORCCA tablet digital ink format to Unipen format is easier, as it involves extracting x, y and pressure data from ORCCA ink format and converting them into Unipen formats.

Chapter 5

Preprocessing

The input for most of the handwriting recognizers is “clean” data. This requires pre-processing of the raw data collected from tablet devices. Preprocessing is needed for the following reasons as well:

- Raw data contains noise from the electronic device used for data collection. For example, the uneven surface or the improper configuration may generate noise.
- Individual writers would introduce variations in the handwriting.
- The sampling rate is not sufficiently high for certain tablet devices, thereby requiring the insertion of additional points.
- Symbol size should not affect the recognition results, which is the reason for the scaling of symbols.
- Strokes can be written in different directions and orders. For example, one may write a horizontal or vertical line first when recording the character “+”.

In the following sections, we describe the preprocessing operations in detail. The preprocessing operations include: dehooking, smoothing, re-sampling, stroke re-direction, stroke re-ordering and size normalization.

5.1 Dehooking

Hooks at the ends of the stroke are very common. In our symbol database, certain individuals’ handwriting always begins with hooks (see the hooks in the Figure 5.1), as hooks are associated with writing styles. In fast writing, hooks may be artifacts of the device. The pen-down/pen-up event cannot be captured at the same time as the stylus touches/lifts off the tablets.

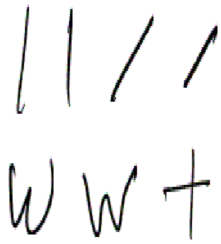


Figure 5.1: Examples of Hooks

Since hooks are positioned at the ends of a stroke accompanied by a sharp turning point in the stroke, we detect them by checking the changes of the turning angles as well as the location of the hooks. Turning angle is formed by the consecutive line $\overline{p_{i-1}p_i}$ and $\overline{p_ip_{i+1}}$ (see Figure 5.2). Mathematically, the following conditions are used to detect hooks:

$$\theta_i > \theta \tag{5.1}$$

where θ is a given threshold.

$$\sum_{k=1}^{i-1} \mathrm{arcLength}(\mathrm{p_{k+1}}, \mathrm{p_k}) < \alpha \mathrm{L} \tag{5.2}$$

$$\sum_{k=i}^{n-1} \mathrm{arcLength}(\mathrm{p_{k+1}}, \mathrm{p_k}) < \alpha \mathrm{L} \tag{5.3}$$

where α is a real number between 0 and 1 and L is the stroke's length. In the present work use $\theta = 90$ degrees and $\alpha = 0.12$. In comparison, Tapia [61] used a threshold value of $\theta = 85$ degrees. Figures 5.3, 5.4, 5.5 and 5.6 show examples of characters before and after dehooking.

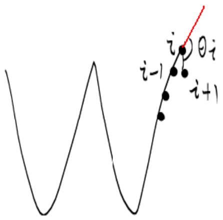
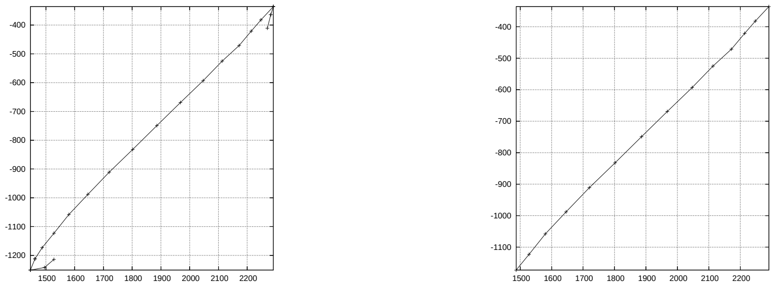


Figure 5.2: Definition of theta in the case of hooks



(a) **(b)**

Figure 5.3: (a) before dehooking (b) after dehooking

(a) **(b)**

Figure 5.4: (a) the symbol "slash" before dehooking (b) after dehooking

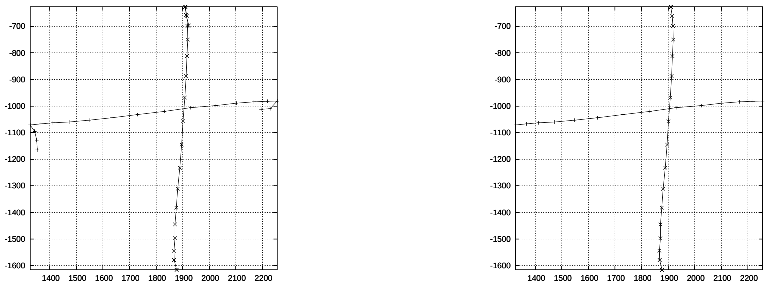
5.2 Smoothing

Smoothing operations are widely used as a filter in noise removal. For example, sharp angle changes at the ends of strokes may generate digital noise. Smoothing is done by substitution of a point with the weighted average of its neighboring points.

$$x_i^{'} = \sum_{k=-m}^{m} \alpha_k x_{i+k} \tag{5.4}$$

$$y_i^{'} = \sum_{k=-m}^{m} \alpha_k y_{i+k} \tag{5.5}$$

where α_k is the weight on point (x_{i+k}, y_{i+k}), $2m$ is the number of neighboring points. We set $m = 3$, $\alpha_k = 1/6$ in our smoothing operation. Other researchers have used a discrete Gaussian distribution to determine the coefficients. For example, the coefficients' values in Tapia's work [61] have been set to be $\alpha_{-1} = 1/4$, $\alpha_0 = 1/2$ and $\alpha_1 = 1/4$.



(a) **(b)**

Figure 5.5: (a) before dehooking (b) after dehooking

(a) **(b)**

Figure 5.6: (a) the symbol “plus” before dehooking (b) after dehooking

Figure 5.7 and Figure 5.8 show the results of different smoothing operations.

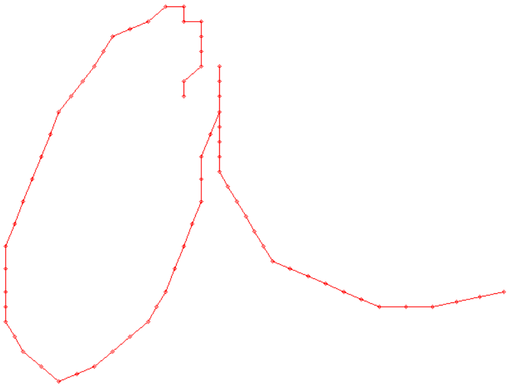


Figure 5.7: (a) before smoothing

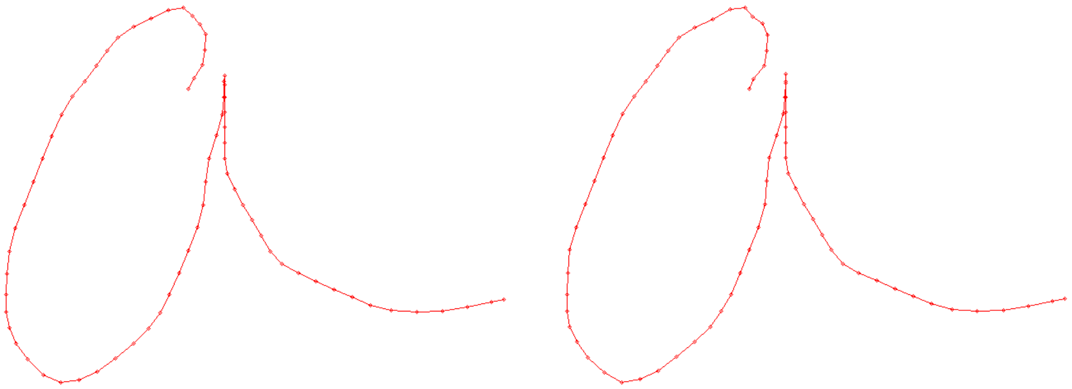


Figure 5.8: (b) after average smoothing (c) after Gaussian smoothing

5.3 Re-sampling

Depending on writing speed, the points in the handwritten strokes may be separated by large distance and in general are unevenly distributed. In the case of fast writing, only a few points are recorded while at slow speeds strokes are recorded with high point density. Re-sampling is applied to remove the speed factor by interpolating new points or removing points depending on distance so that the final separation between consecutive points is equal.

Re-sampling can also reduce computation time by removing points which are close. The re-sampling procedure starts at the first point p_1, checks the distance to the next point, $\mathrm{arcLength(p_1, p_2)}$ and if $\mathrm{arcLength(p_1, p_2) > \alpha L}$, the insertion process is performed as follows: m points are inserted between p_1 and p_2,
$m = \lfloor \mathrm{arcLength(p_1, p_2)/(\alpha L)} \rfloor$.

$$x_i^{'} = x_1 - k_i \times (x_1 - x_2)/(\alpha L + 1) \tag{5.6}$$

$$y_i^{'} = y_1 - k_i \times (y_1 - y_2)/(\alpha L + 1) \tag{5.7}$$

where i from 2 to $m + 1$ in x_i, y_i and k_i.

After inserting m points, we check the distance between $p_m^{'}$ and its next point, if the distance is greater than αL, the insertion process is repeated. Otherwise the following removal process is applied:

```
if arcLength($p_1, p_2$) < $\alpha$L then
    Remove $p_2$
    if arcLength($p_1$, p3) < $\alpha$L then
        Remove $p_3$
    else
        Insert between $p_1$ and $p_3$
    endif
else
    Insert between $p_1$ and $p_2$
endif
```

Algorithm 5.1: Resampling

5.4 Stroke Re-direction

The existence of variants in the direction of handwritten strokes is another important reason for the difference between on-line and off-line handwriting recognition. One individual may record a given symbol leaning in various directions, as well as left- and right-handed individuals may have preferences in systematically writing characters slanted left or right thereby affecting the accuracy of an on-line recognition process. By looking at the handwriting images, the difference in the stroke directions cannot be detected, therefore stroke direction is not considered in off-line handwriting recognition.

Most intuitive way to handle the stroke direction is to build different models for different directions, although this would be computationally expensive since the number of models would double for a single stroke symbol. For a two-stroke symbol, there would be four models even if we didn't consider the stroke order.

Nicholas [44] created a canonical direction method for each stroke, which we have used in stroke redirection. Strokes are classified into four types: horizontal, vertical, diagonal and closed. Two weighted ratios are defined in strokes classification:

$$R_x = \frac{|x_1 - x_n|}{D} \tag{5.8}$$

$$R_y = \frac{|y_1 - y_n|}{D} \tag{5.9}$$

where x_1, x_n are the first and last point in the stroke respectively. D is the length of the diagonal of the bounding box of the stroke.

We have used a threshold $\delta \in [0, 1]$ together with the following conditions to detect the stroke type:

$$\textit{Horizontal:} \quad R_x \geq \delta \textit{ and } R_y < \delta$$

$$\textit{Vertical:} \quad R_x < \delta \textit{ and } R_y \geq \delta$$

$$\textit{Diagonal:} \quad R_x \geq \delta \textit{ and } R_y \geq \delta$$

$$\textit{Closed:} \quad R_x < \delta \textit{ and } R_y < \delta$$

Alteration in horizontal direction of the stroke is performed if $x_1 < x_n$, while vertical and diagonal alterations in stroke direction are carried out if $y_1 > y_n$. After the stroke direction has been normalized, we build a model for the character. To recognize a symbol, stroke normalization is performed on the unknown symbol as well.

It is important to select a proper value of δ to avoid errors. This value need to satisfy “horizontal”,“vertical”,“diagonal” and “closed” categories. For example, in the case of two symbols having the same stroke, we may only reverse the stroke in one of the symbols. Nicholas [44] used $\delta = 0.37$, we have used this result in our implementation.

5.5 Stroke Re-ordering

Multiple-stroke character, such as “+”, may be written by first recording either the horizontal line or the vertical line. Similarly to variance in stroke direction, this would affect on-line handwriting recognition. Building multiple models for the same character is to be avoided for the sake of efficiency. Normalizing the order of the strokes is an option to reduce the stroke order variance. Nicholas [44] defined a canonical stroke order. It is necessary to measure the angle between the upper edge of the symbol's bounding box and the line segment defined by the upper left corner of the bounding box and the last point in the stroke. Figure 5.9 shows the angle in symbol π. Each stroke is assigned a value according to the angle, followed by the strokes reordered in the ascending order based on the angle value.

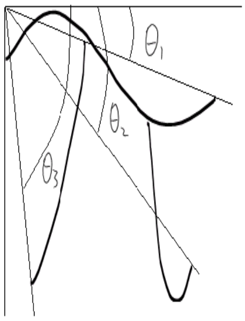


Figure 5.9: Stroke Re-ordering, from Nicholas [44]

5.6 Size Normalization

Size normalization is necessary for model based recognizers since we need to compare the unknown symbol with the model and size should not affect the recognition result. In our size normalization, we move the upper left corner of the bounding box to $(0, 0)$. Given a new width $w^{'}$, we calculate the new height to keep the height-width ratio. The new center of the bounding box is:

$$h^{'} = w^{'} \times h/w,\ c_x^{'} = w^{'}/2,\ c_y^{'} = h^{'}/2$$

where $h^{'}, w^{'}$ is the new height and the new width respectively.

We select $w^{'} = 50$ pixels. While the selected value is unimportant by itself, it needs to be consistently applied to all symbols.

We use the following formulas to calculate the new x, y, *i.e.* $x^{'}, y^{'}$:

$$x^{'} = \frac{(x - c_x) \times w^{'}}{w} + c_x^{'} \qquad (5.10)$$

$$y^{'} = \frac{(y - c_y) \times h^{'}}{h} + c_y^{'} \tag{5.11}$$

In this chapter, we discussed why it is necessary to do pre-processing operations. We described how we preprocessed our data by the following operations: dehooking, smoothing, re-sampling, stroke re-direction, stroke re-ordering and size normalization. These operations removed digital noise and eliminated stroke order, stroke direction and size impact on recognition.

Chapter 6

Mathematical Handwriting Variant Analysis

6.1 Introduction

Character variant analysis is important for math handwriting recognition. For this, we need to look at actual data to see what variants arise in practice. Studies in this field have been performed by Ward and Kuklinski [69], particularly on the components of handwriting style variability. Kuklinski has proposed improvements of recognition through shape correlation among characters. Srihari, *et al.* [60] used legibility, contour features, *etc* to identify writers.

We have proposed an allomorph set for each studied symbol based on its geometry structure and individuals' writing habits. An allomorph is a representative example of a handwriting symbol. We have collected handwriting samples from about 50 individuals using a Tablet PC and an IBM CrossPad. The participants in the study were chosen to have different scientific backgrounds, such as mathematics, physics, chemical, electronic and mechanical engineering. Also, they were selected to represent various national backgrounds, such as Canada, Russia, Japan, China, Iran, France, USA and Romania. The inclusion of a variety of backgrounds in the study is a promise for the achievement of higher precision in handwriting analysis, since nationality and mathematical background are important factors, as we shall see later.

6.2 Variant Analysis

In this section, the factors that cause variants in handwriting are presented. These factors are the geometric structure of the symbol, writing habits, visual perception as well as the academic and national background of the individual.

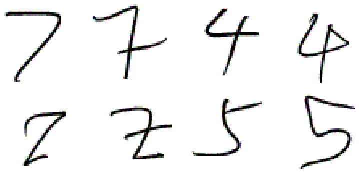


Figure 6.1: Different Number of Strokes

6.2.1 Number of Strokes

Number of strokes is one obvious and important factor contributing to variance. Because of cursive writing style and national writing preference, the same character can have a different number of strokes in different written forms (Figure 6.1). For example, Ward and Kuklinski [69] maintained that the symbol displayed on the top left in Figure 6.1 is common for North Americans, while the symbol on its right is widely accepted in Europe. While it is true that nationality affects writing style, the above conclusion is certainly not a hard and fast rule.

Tapia [61] reduced the stroke variance by enforcing fixed number of strokes per character, e.g. if the number of strokes M exceeds the given number N, the $N-1$st to Mth strokes are concatenated. In our HMM recognizer, the number of strokes is not reduced, but reduction has been considered by building a hierarchy hidden Markov models for each character, which have different numbers of strokes.

6.2.2 Beginning and Ending Hooks

Sometimes individuals would write symbols that begin and end with a hook. Hooks may be extended into lines, curves or even loops, resulting in different shapes depending on hook lengths. In our studies, length of hooks won't be considered a factor if the overall symbol structure is not changed by the hooks. Again, this is consistent with the fact that we conduct shape analysis from a comprehensive perspective. Figure 6.2 shows examples of hooks.

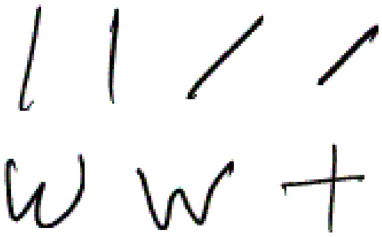


Figure 6.2: Hooks

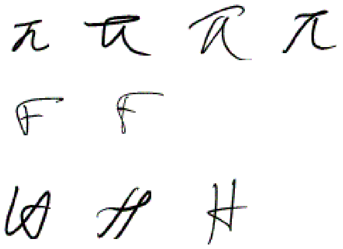


Figure 6.3: Connected Strokes

6.2.3 Cursively Connecting Segments

If one would segment the cursive and non-cursive handwriting of the same symbol at its turning points, in most cases the cursive one would turn out to have more segments. This is due to the connecting segments, which make handwriting complicated and cause variant. Figure 6.3 shows some examples. The first "H" in Figure 6.3 can be segmented to five segments while the second "H" has three segments. Cursive writing generated different number of strokes and shapes. The different ways of segment connections will result in different representations.

6.2.4 Cusps Changing to Small Loops

In the examples shown in Figure 6.4, one can see that how cusps could be written as small loops. Most of these changes are caused by retracing, as seen in the cases of symbols "m","n". Some of the turning angles change to loops, as in the case of symbol "a". The loop in the second last symbol "r" in Figure 6.4 is hidden. Hidden loops can still be considered as cusps, while a visible loop makes a different allomorph.

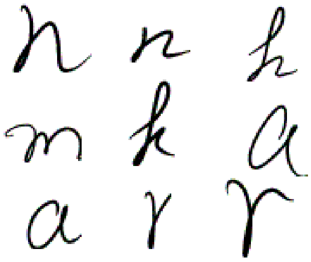


Figure 6.4: Cusps becoming Loops

6.2.5 Angles Changing to Loops

A loop could appear due to the style of handwriting. Examples are shown in Figure 6.5. This can also happen when an ending stroke crosses another stroke and forms a loop (see the symbols “j” and “g” in Figure 6.5).

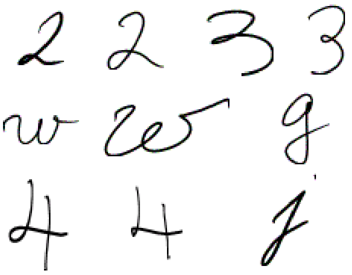


Figure 6.5: Angles becoming Loops

6.2.6 Omitting Character Tails

Symbol tails can be important in the recognition process. For example, the characters “9” and “*g*” are distinguished through the handwriting of the tail. As it is possible that an individual would omit writing the tails, this would cause ambiguities as seen in the second symbol in Figure 6.6, which can be considered to represent one of the four different symbols: “*I*”,“*L*”,“*l*”,“1”.

Figure 6.6: Omitting Little Tails

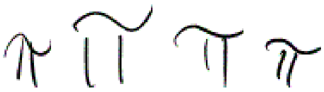


Figure 6.7: Straight Line Variance

6.2.7 Wavy Lines

In the case when straight lines are not properly written, a smoothing operation is needed to remove a wavy line. If the wiggles are large enough, they cannot be removed by smoothing and become a variant-producing factor as shown in Figure 6.7. The large wiggle in π creates an allomorph, which is included in the recognition models for π.

6.2.8 Quick and Cursive Handwriting

Such handwriting results in a reduced number of strokes as well as in an altered shape of a character, which is another variant-producing factor. Examples are shown in Figure 6.8. We have to admit that we cannot recognize all of the simplified handwriting. Some of them are not suitable for machine-enabled recognition.

Figure 6.8: Simplified Symbols

Figure 6.9: Loop to Cusp Example

6.3 Factors Which Are Not Considered

There are certain factors we do not consider for a number of reasons. For example, Ward and Kuklinski argued that loops could collapse to cusps (Figure 6.9). This may be useful in English, but is not a useful strategy when math symbols are used.

6.4 Allomorph Variation Among Characters

Based on the above variance considerations, we have built a database of allomorphs for each character. From the set of allomorphs, we have reached the conclusion that handwritten alphabetic characters have larger variability than Greek letters, followed by numerals and mathematical operators. When Srihari and his colleagues investigated the individuality of handwritten characters, they reached a similar conclusion [60]. The degree of variability also depends on the complexity of the handwritten character.

6.5 Using of Variance Analysis in Handwriting Recognition

The question of how to use the variance to improve handwriting recognition depends on the method used for recognition.

For model comparison based recognition, we can create difficult models from the variants. Ward and Kuklinski [69] have built symbol models by producing combinatorial variations of strokes. For upper case “A”, their method was set up to estimate 15,552 symbol models, which required significant resources. In our recognizer we have built multiple hidden Markov models for each symbol according to variants. These models can be classified by the writing style. For example, if one writes “h” as in Figure 6.10 (a), his or her “k” will quite likely be the “k” in Figure 6.10 (b). If one writes “h” as the “h” in Figure 6.10 (d), then his or her “k” will quite likely be as the “k” in Figure 6.10 (c). This seems like user identification, but it is much simpler and easier to compute.

In this chapter, we described mathematical handwriting variant analysis. We observed that the following factors contributed to variant: number of strokes, hooks, cursive writing, loops coming from cusps and angles, tails and wavy lines. Allomorphs for each symbol are represented. We also described how to use variant analysis in recognition.

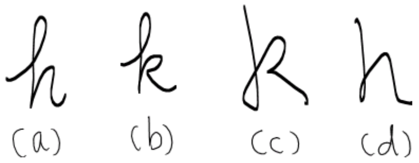


Figure 6.10: Writing Style Examples

Chapter 7

Feature Extraction

7.1 Introduction

It is well accepted that feature extraction is important in handwriting recognition [65]. Despite that recently there has been little research development in this area, most of the existing recognizers use this technique to certain degree [40].

At present, there is not an agreed upon set of features that is considered sufficiently exhaustive to be used in a universal recognizer. In handwriting recognition, especially for large sets of symbols, extraction of proper features is challenging due to handwriting variances. For this reason, we have performed a feature study based on the variance analysis outlined in the previous chapter, specifically attempting to define features with high distinguishing effect. In this chapter, we describe the set of features we have adopted (based on analysis of empirical data), and provide algorithms for extracting these features. These features are then used in the elastic recognizer and hidden Markov model recognizers.

7.2 Published Studies on Features of Written Symbols

There are three main categories of features used in handwriting recognition, namely direction features, shape features and ink-related features. Generally speaking, these features are used for two different purposes, either for recognizing or for grouping handwritten symbols. We will give a brief review of what they are and how they are used.

- Direction Features: Rigoll and Kosmala [50] used the angle between two consecutive points as one of the major features in their HMM-based recognizer. Winkler [72] used angle and angle difference in his HMM-based recognizer. Okamoto and Yamamoto [47] used features with clockwise/counterclockwise change of direction for Japanese character recognition.

- Shape Features: Okamoto and Yamamoto [47] used circle as their shape feature. Chan and Yeung [8] used lines, counter-clockwise/clockwise curves and loops in their mathematical handwriting recognition.
- Ink-related Features: Winker [72] used pen-down/pen-up and Kurtzberg [35] used number of points in each stroke and number of strokes per symbol as ink-related features.

The studies listed above used features for recognizing symbols. Kurtzberg [35] also used features for grouping in order to reduce computation. After grouping, comparisons in recognition are based on groups of models instead of the whole set of models. Kurtzbeg's features can be listed as: number of strokes, number of points per stroke, number of points in the symbol, height of the lowest point, height of the highest point, height of the lowest point per stroke and height of the highest point per stroke. Some of these features, *e.g.* number of points per stroke, are device dependent. Other features such as height are assumed to depend on certain data-collection conditions, *e.g.* guidelines.

In our work, we studied several published features and decided to give preference to device-independent and context-independent features. Some features have been tested on small set of symbols elsewhere. Here we tested our features on a larger set of mathematical symbols. We have selected a subset of these features for the purpose of pre-classification in an elastic matching based recognizer. We describe more details in the next chapter. We also used some of these features in our hidden Markov model based recognizer, as presented in Chapter 10 and Chapter 11.

7.3 Feature Families

We have organized symbol features into different categories. When handwritten symbols are recognized, geometric and other features, such as loops, play an important role. If the characters are too cursive to recognize individually, one must rely on context, loops and intersections to distinguish characters. All these factors can then be taken into account.

7.3.1 Geometric Features

Number of Cusps: A cusp is defined as a sharp turning point in a stroke, formed locally by three points, *e.g.* p1, p2, p3 (Figure 7.1 (a)). If the angle of p1, p2 and p3 is sufficiently small, a cusp could exist within the given symbol. In order to determine which cusps are well-defined, two more neighboring points are checked: p0 and p4. p0, p1 and p2 should be on a relatively straight line, likewise for p2, p3, p4. As our straightness threshold, we have selected the value of 170 degrees.

Number of Loops: This includes closed and open loops. In order to find loops, we define the so-called "minimum distance pair".

Minimum Distance Pair: A pair of points that has the minimum non-local distance in a given area. To ensure non-locality (*e.g.* sequential points in a trace), the time interval between the pair of points must exceed a certain threshold. There is only one minimum distance pair in a given neighborhood.

Approximate loops may be found using minimum distance pairs. Our algorithm starts with finding a "minimum distance pair". Given time threshold and distance threshold, we have a sequence of pairs:

$$(p_i, p_{i+m_{i_1}}), (p_i, p_{i+m_{i_2}}), ..., (p_i, p_{i+m_{i_k}}), ..., (p_j, p_{j+m_{j_1}}), (p_j, p_{j+m_{j_2}}), ..., (p_j, p_{j+m_{j_k}}), ...$$

those satisfy the following conditions:

$$m_{i_q}, m_{j_q} > n \times \delta_t, \mathrm{distance(p_i, p_{i+m_{i_q}}), distance(p_j, p_{j+m_{j_q}}) > L \times \delta_d}$$

where $q \in 1..k$, n is the total number of points, δ_t is the time threshold, L is total length of the stroke, δ_d is the distance threshold. Next, for the pair that has the same beginning index, we find the minimum distance one: $(p_i, p_k), (p_j, p_l), ...$ followed by the application of certain filters on these pairs.

The “v” filter starts a search with a pair of points separated by minimum distance points, known as “begin” and “end” points. The algorithm then computes a straight line between them, followed by building a parallel line from a neighboring point of the “begin” point, “nextp” in the Figure 7.2. This parallel line intersects the stroke at “interp” point. The distance between the point “nextp” and the point “interp” is called “paraDistance”. The distance between the “begin” point and “end” point is called “origDistance”. We filter out this minimum distance pair, with distance expension threshold, ρ_d, if $paraDistance < origDistance \times \rho_d$.

An “area filter” is used to remove tiny loops by computation of the polygon area formed by the points between “begin” and “end” points. If $polygonArea < boundingboxArea \times \rho_a$, the loop is filtered out (ρ_a is area threshold).

The “index filter” checks the distance between beginning indexes. If two beginning indices i, j are too close, *i.e.* $|i - j| < n \times \rho_i$, one pair is filtered out. n is the total number of points. ρ_i is index threshold.

Several thresholds are being used in the finding loop algorithm. Since setting proper threshold values improves the work of the algorithm, we have conducted a number of experiments and have adopted the following empirical values:
$\delta_t = 1/5,\ \delta_d = 1/8,\ \rho_d = 1.125,\ \rho_a = 0.05,\ \rho_i = 1/12.$
The above values are valid when character size is normalized with 50 pixels width, while keep width-height ratio. Refer to section 5.6 for details.

Chan and Yeung pointed out that detection of loops is not always trivial [8]. They used chain code sequences to detect loops. Their chain codes depend on the starting stroke position. In comparison, our algorithm does not have this limitation. It is able to detect the loop in Figure 7.2(b) while Chan and Yeung’s method fails to do so.

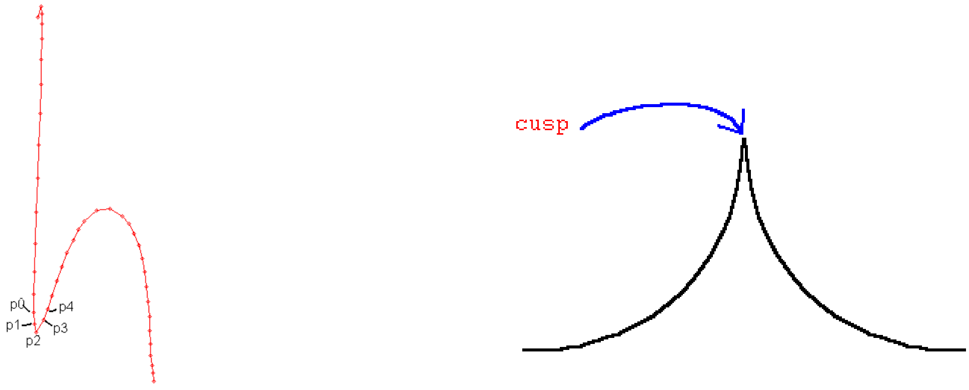
Number of Self- and Intra-stroke Intersections: To calculate the number of intersections, we have implemented the modified Bently-Ottman [14] sweepline algorithm [12] [18]. Every time an intersection is found, the two line segments associated with the intersection are deleted, followed by the insertion of two new lines into the set of line segments. These two lines begin from the intersection point, and end with their old ending points.

This method may be described in terms of a "sweep line" which can be treated as an imaginary vertical line. If it is made to pass through the given set of line segments from left to right, the line segments that intersect a vertical sweep line are ordered according to the y-coordinates of the intersection points.

Let us consider two segments a and b. We shall say that these segments are comparable at x if the vertical sweep line with x-coordinate x intersects both of them. We shall say that a is above b at x, written as $a >_x b$, if a and b are comparable at x and the intersection of a with the sweep line at x being higher than the intersection of b with the same sweep line. For example, in Figure 7.3, we have $a >_r b$, $c >_s b_2 >_s a_2$.

We introduce a sweep line status T, which is used to describe the relationships among the line segments. The event point schedule E is another quantity being used to store a sequence of event points, ordered from left to right. To get a set P of intersections, the following operations are required:

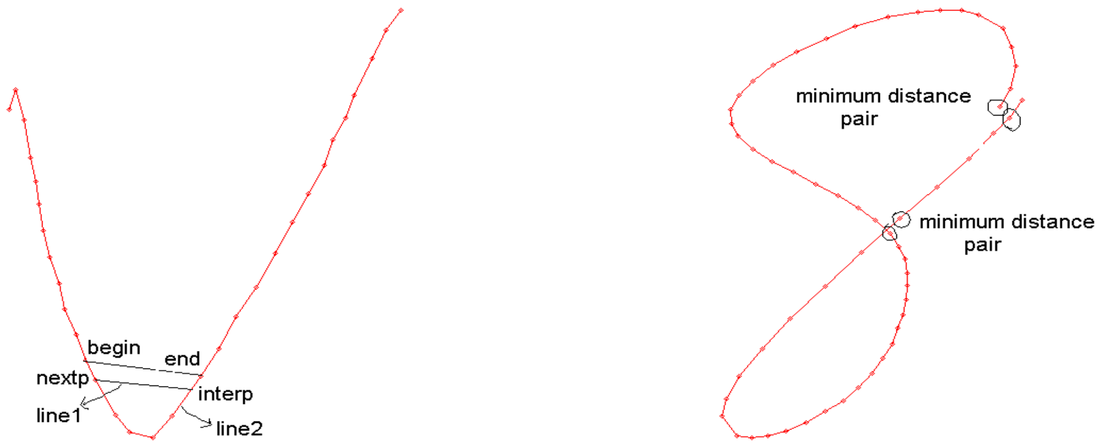
- INSERT(T, s): insert segment s into T
- DELETE(T, s): delete segment s from T
- ABOVE(T, s): return the segment immediately above segment s in T
- BELOW(T, s): return the segment immediately below segment s in T
- INSERT(P,p): insert point p into P
- INSERT(E,e): insert point e into E
- DELETE(E,e): delete point e from E



(a) **(b)**

Figure 7.1: (a) Cusps in "h" (b) Cusps

Algorithm 7.3.1 takes a set of line segments S as input and finds a set P of intersections:



(a) **(b)**

Figure 7.2: (a) Example 1 for Loop Detection (b) Example 2 for Loop Detection

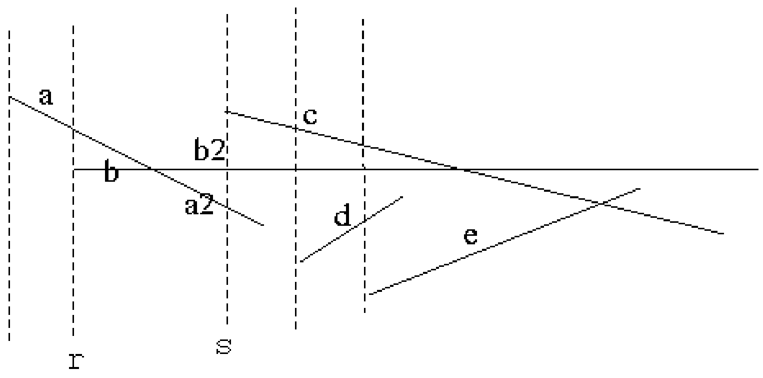


Figure 7.3: Modified Sweepline Algorithm

```
Data: $P \leftarrow \emptyset$
Data: $T \leftarrow \emptyset$
Data: sort the endpoints of the line segments in **S** from left to right
Data: breaking ties by putting points with lower $y$ coordinates first
Data: $E \leftarrow$ sorted endpoints
foreach point p in E do
    if p is the left endpoint of a segment s then
        INSERT(T,s), if ABOVE(T,s) exists and intersects s at point p then
            INSERT(P,p), INSERT( E,p)
        endif
        if BELOW(T,s) exists and intersects s at point p then
            INSERT(P,p), INSERT(E,p)
        endif
    else
        if p is the right endpoint of a segment s then
            if ABOVE(T,s) and BELOW(T,s) exist and ABOVE(T,s) intersects
            BELOW(T,s) then
                INSERT(P,p), INSERT(E,p), DELETE(T,s)
            endif
        endif
        if p is the intersection point of segment m and segment n then
            build two new line segments $m2$ and $n2$ which begin with $p$,
            end with their old ending points, $m_{endp}$, $n_{endp}$,
            INSERT(E,p), DELETE(E,$m_{beginp}$),
            DELETE(E,$n_{beginp}$), DELETE(T,m),
            DELETE(T,n), INSERT(T,m2), INSERT(T,n2)
        endif
    endif
endforeach
return [P]
```

Algorithm 7.1: Intersection Detection

7.3.2 Ink-Related Features

Ink-related features capture ink distributions within the symbol. These features can be used to pre-classify symbols. For example, a, b, and p belong to three groups according to their spatial point density. The features include number of strokes, number of points, point density and pen-down/pen-up.

Number of Strokes: This information is contained in the data structures returned by the ink collection.

Point Density: We have adopted the letters "o," "p" or "b" as a measure of spatial point density.

- In the case of o-density, ink is to be evenly distributed in the whole stroke.
- In the case of p-density, ink is to be distributed in the upper part more than that in the lower part.
- In the case of b-density, ink is to be distributed in the lower part more than that in the upper part.

To compute these, the ink bounding box is divided vertically into three parts: the upper 40%, the middle 20% and the lower box of remaining 40%. The ink bounding box is divided into three boxes instead of two due to the variance in handwriting. For example, the lower part of letter "b"may occupy more than 50% of the symbol height. By measuring the ink in different boxes, we calculate the point density. Figure 7.4 shows the point density of symbol "b".

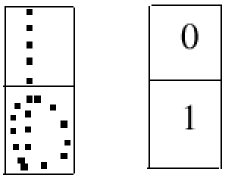


Figure 7.4: Point Density

Pen-up and Pen-down: Traditionally, only pen-down strokes have been used in handwriting recognition, while only recently the detection of pen-up strokes has become possible by using tablet devices - an advantage which is now being used by many researchers. In our work, we have chosen 1 to represent pen-down, and 0 to represent pen-up state.

7.3.3 Directional Features

Initial Direction: the direction from the beginning point toward the stroke.

End Direction: the direction from the end point toward the stroke.

Initial-End Direction: the direction from the initial point to the end point.

The value of the direction is one of the 8 chain codes. See Figure 7.5. The initial direction is calculated from the first point and the third point, while the end direction is calculated from the last point and the third last point. The following formula is valid for the calculation of direction:

$$direction = \lfloor((((angle \times 16)/(2 \times \pi)) + 1)\%16)/2\rfloor$$

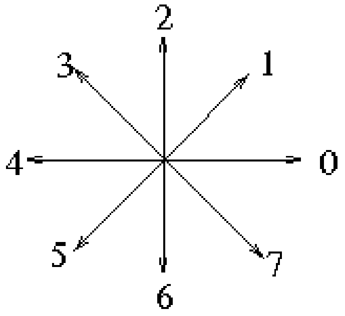


Figure 7.5: Chain Code

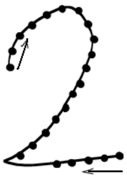


Figure 7.6: Initial-End Direction

Writing Angle: Every point i is considered to have a writing angle, defined as the angle between line segment $l(i, i-2)$ and line segment $s(i, i+2)$. See Figure 7.7.

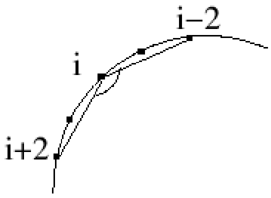


Figure 7.7: Writing Angle

7.3.4 Global Features

Initial and End Position: If the ink bounding box is divided into four quadrants: NE, NW, SE, SW, the initial and end points of a handwritten symbol will be positioned within one or two of these boxes, which is considered as a separated, distinctive feature of the given symbol.

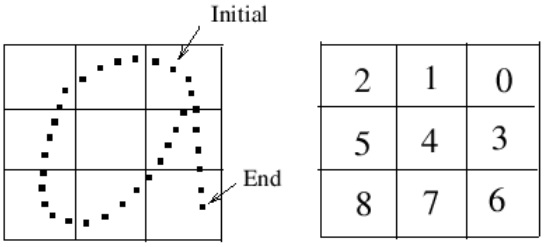


Figure 7.8: Initial-End Position

Width to Height Ratio: Depending on width and height, the symbols are classified as having a width to height ratio of 0 (slim symbol), 1 (wide symbol) or 2 (regular symbol).

In this chapter, we reviewed the features widely used in handwriting recognition. Moreover, we described the features used in our recognizer. Furthermore, algorithms for extracting these features are also presented. Our algorithm for loop extraction performs better than other algorithms. We improved an existing algorithm for finding intersections as well.

Chapter 8

Prototype Pruning in Elastic Recognition

In the previous chapter, we have described ink features and we have chosen algorithms for extracting these features. In this chapter we describe how to use these for pruning prototypes. The performance of the recognizer after applying them is analyzed as well.

8.1 Prototype Pruning by Feature Extraction

Achieving a faster recognition process without sacrificing accuracy is one of the main objectives in handwriting recognition research. In the present work, a pre-classification strategy in combination with elastic matching has been used to improve recognition speed. Elastic matching [26] [66] is a model-based method that involves computation proportional to the set of candidate models. Increasing the number of models normally improves recognition accuracy, but also reduces recognition speed. To address this problem, we propose to prune the character prototypes by examining certain character features. Suitable definitions arising from the analysis of different features have enabled us to adapt them in an elastic recognition system, achieving higher recognition speed while maintaining high recognition accuracy.

Prototype pruning is an intuitive way to break a large vocabulary into several smaller groups. Our method locates the group to which the unknown symbol belongs and attempts to find the symbol in the group. The computational effort in locating the group is assumed to be low enough to ensure better recognition performance. If there is prior knowledge of the chances for input of a particular symbol, the search can be narrowed [54]. But such information is not available in most cases.

8.2 Elastic Matching

Elastic matching achieves recognition by finding the minimum distance between the unknown symbol and a member of a set of models. Figure 8.1 (from [66]) shows an example.

The distance between two continuous curves is approximated by summing the distances between corresponding points. The mapping between the two sequences of points allows for both one-to-one and many-to-one correspondences between points using dynamic programming [66]. Every point in the unknown symbol is matched to one or many points in the model, or none at all. There is no need to have exactly the same number of points in the model and the unknown symbol since redundant points will be removed by the matching algorithm.

The total distance between the ith point of the unknown character and the jth point of the model, $D(i,j)$, is calculated as:

$$D(i,j) = \delta(i,j) + \begin{cases} \sum_{k=0}^{j-1} \delta(0,k) & \text{if } i = 0 \\ \sum_{k=0}^{i-1} \delta(k,0) & \text{if } j = 0 \\ \min \begin{cases} D(i-1,j) \\ D(i-1,j-1) \end{cases} & \text{if } i > 0, j = 1 \\ \min \begin{cases} D(i-1,j) \\ D(i-1,j-1) \\ D(i-1,j-2) \end{cases} & \text{if } i > 0, j > 1 \end{cases} \tag{8.1}$$

$$\delta(i,j) = (x_i - x_j)^2 + (y_i - y_j)^2 + C\,|\phi_i - \phi_j|$$

where ϕ represents the orientation and the curvature. Constant C is empirically determined.

Let ϕ be the angle of elevation of the tangent to the point, calculated as follows:

$$\begin{cases} \phi_i = \arccos(\frac{x_{i+1} - x_i}{r}) & y_{i+1} < y_i \\ \phi_i = \arccos(\frac{x_{i+1} - x_i}{r}) + \pi & y_{i+1} < y_i \\ \phi_i = \phi_{i-1} & i = n \end{cases}$$

where,

$$r = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$$

In the general case, $i > 0, j > 1$, at every step in the calculation of the distance there is a choice whether to match the next point in the unknown to the same point in the model $D(i-1, j)$, to the next point in the model $D(i-1, j-1)$ or skip a point in the model $D(i-1, j-2)$. The choice that gives a minimum distance is selected.

The distance function, δ, was adopted for the calculation of the sum of the Euclidean distance and the weighted difference of curvature. To save computation time, we use Euclidean distance's square, $(x_i - x_j)^2 + (y_i - y_j)^2$ for δ.

The distance $D(n, m)$ between the last points is divided by the number of points in the model as seen in the Equation 8.2. This facilitates comparison between different models of different lengths, so that the one with the smallest average distance is chosen.

$$\tilde{D}(n, m) = \frac{D(n, m)}{m} \tag{8.2}$$

The calculation of the inter-point distance (Equation 8.1) was a main factor in slowing recognition. This is due to the fact that the calculation contains many overlapping sub-problems that require multiple re-calculations of the same distances. We avoid this by using dynamic programming to implement elastic matching in C++. For the details of the program for elastic matching, please refer to Appendix B: Dynamic Programming for Elastic Matching. The calculation starts at $D(0, 0)$, building up a matrix of distances until the top $D(n, m)$ is reached. For example, $D(0, 0)$ is the distance between first point in the unknown character and first point in the model. After we filled $D(0, 0)$ to the matrix, we calculated $D(0, 1)$, which is the summation of $D(0, 0)$ and the distance between the first point in unknown character and the second point in the model. Similarly, $D(0, 2)$ is the summation of $D(0, 1)$ and the distance between the first point in unknown character and the third point in the model. The total distance between the unknown character and the model is $D(n, m)$.

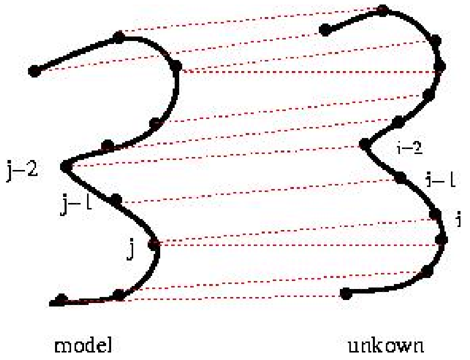


Figure 8.1: Elastic Matching (from [66])

8.3 Experimental Results

Our initial objective was to reduce the computation time, using the features in our elastic matching-based recognizer. We have therefore based our selection of features not just on their effectiveness at dividing the symbol set, but also on their computational cost. The features we selected for our experiments were: number of strokes, initial position, width to height ratio, end direction and initial to end direction. The number of strokes and the width to height ratio were required to match exactly. The value assigned to initial position as described in section 7.3.4 could differ by one. The same rule applies to initial to end direction, described in section 7.3.3. Additionally, the value assigned to end direction was allowed to vary by up to two.

Our test data set contained 227 symbols shown in Figure 8.2, including digits, Latin letters, some Greek letters and mathematical operators. In the tests individuals were requested to write eight sets of the symbols. The first four sets were used as prototype sets, and the rest of the sets were used as testing sets. The results are shown in Tables 8.1, 8.2 and 8.3. P denotes prototype sets. For example, $P1$ indicates the data from the first prototype set. T stands for test sets. For example, $T1$ denotes the data from the first testing set.

Table 8.1 shows experimental results when features are not used. The first column indicates the prototype and testing sets. The second column contains the number of prototypes used for recognizing a symbol. For example, if the prototype set is $P1$, for each symbol in $T1$, we need to perform 227 symbol comparisons. When another prototype set is added, the number of prototypes is 227×2 for each symbol in the test sets.

Table 8.2 presents the experimental results when features were used. The first column is the same as in Table 8.1. The second column is the number of prototypes in their respective sets. The prototypes are updated by training on the second, third and fourth sets. The third column displays the number of prototypes used in recognition. For example, if the prototype set is $P1$, for a symbol in $T1$, we need to perform 26 comparisons. The fourth column is the percentage of prototypes pruned.

Tables 8.1 and 8.2 show a final recognition rate of 94.8% without using any features. With features the recognition rate is 91.9%, but the prototypes are pruned by 89.6%, reducing computation proportionately. Note that the percentage of prototypes pruned was relatively constant for each prototype set.

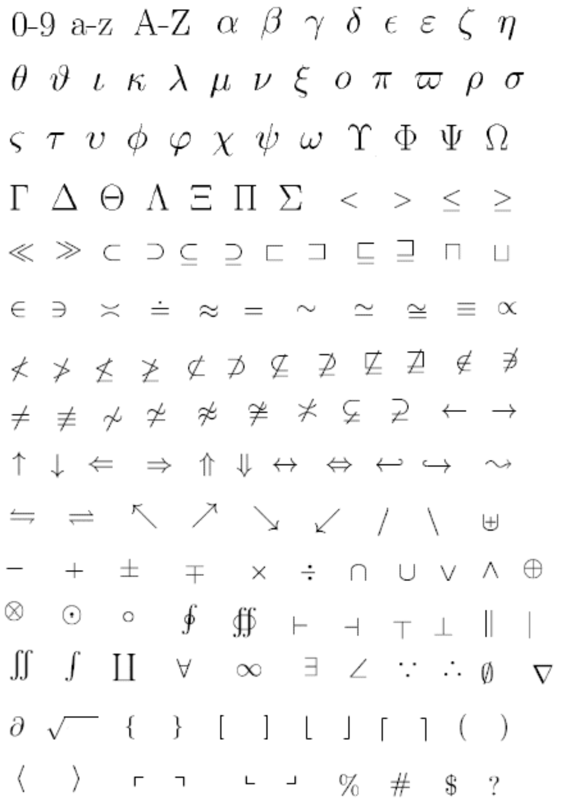


Figure 8.2: Symbols for Experiments

Table 8.3 presents the test results for Kurtzberg's set [35], containing 72 symbols, including 10 digits, 52 upper and lower case Latin letters and 10 punctuation symbols. We acknowledge that it is difficult to compare different recognition systems when the test data sets are not exactly same. If there are large differences in the results, however, the comparison can still be useful.

According to Table 8.3, the fraction of prototypes pruned has improved significantly over Kurtzberg's result, from 61.5% to 85.8%. The recognition rate has been reduced by about 1% but still remains high. As the features used for prototype pruning were easy to compute, the computation was insignificant compared to the elastic matching. In contrast with Kurtzberg's features, the ones used in the present work are device independent and somewhat more general. The last row in Table 8.3 shows the experimental results without using features. Since Kurtzberg used a number of parameters, the number of prototypes in his work has been reduced.

Although we have not used all the features we have studied in the previous chapter (such as loops, intersections *etc.*) for prototype pruning, they are still useful for other purposes.

We have compared our results with Henning's model of word recognition [2]. For this purpose we implemented some of the characteristic features of the model, such as ascender, descender, length of character, and number of center horizontal line crossings. Section 3 of Henning's paper discusses dictionary reduction, which is comparable to prototype pruning used in the present work. To the best of our knowledge, there is no similar recent work for on-line handwritten symbol recognition.

We have applied the features from Henning at the character level and examined the results. The capacity of the reduced (pruned) lexicon to include the correct match is defined as "coverage", which is adopted from Koerich [31]. The results are analyzed in terms of coverage. We tested our features and the ones from Henning's paper (applied to symbols instead of words) on 227 symbols of 2 different writers. All the features, except for length, require exact matches, and length must match within a certain tolerance. Our experiments showed that both Henning's and our features give 100% coverage.

Experiment	# Prototypes	Recog. Rate(%)
P1:T1,2,3,4	227	81.8
P1,2:T1,2,3,4	454	90.1
P1,2,3:T1,2,3,4	681	93.9
P1,2,3,4:T1,2,3,4	908	94.8

Table 8.1: Results in the case when features are not used

Experiment	Number of Prototypes	Candidate Prototypes	Percentage Pruned	Recognition Rate(%)
P1:T1,2,3,4	227	26	88.5	76.0
P1,2:T1,2,3,4	366	38	89.6	85.5
P1,2,3:T1,2,3,4	495	52	89.5	90.0
P1,2,3,4:T1,2,3,4	575	60	89.6	91.9

Table 8.2: Results in the case when features are used

Experiment	Number Of Prototypes		Candidate Prototypes		Percentage Pruned		Recognition Rate(%)	
	J.K's	Ours	J.K's	Ours	J.K's	Ours	J.K's	Ours
P1,2,3,4: T1,2,3,4	121	169	47	24	61.5	85.8	99.0	97.6
P1,2,3,4: T1,2,3,4	122	288	92	288	N/A	N/A	99.0	99.7

Table 8.3: Comparison with Kurtzberg's Results

8.4 Conclusions

Most recognizers focus on a limited range of symbols such as postal codes or Latin letters. Even recognizers for Asian languages deal with a limited vocabulary of strokes which must be given in particular orders. Recognition for large set of symbols is still a challenging research problem.

The present work uses feature sets used in pruning the number of candidates considered in a character match. This is a central feature of our symbol recognition framework, and influences our overall mathematical handwriting project. Our recognizer is able to identify digits, English and Greek letters as well as the common mathematical operators and notations. We have selected our features based on their effectiveness and computational cost. In the optimization of our algorithm, we have identified the symbols features empirically by analyzing a database of 10,000 mathematical handwriting samples. The use of features in pruning was found to be effective, greatly reducing computation time at only a modest decrease in recognition rate.

Chapter 9

Hidden Markov Model Based Recognition

Hidden Markov models (HMM) provide a promising approach to handwriting recognition because they can model temporal patterns. In this chapter we summarize some relevant work in handwriting recognition using HMM. We also provide for completeness an overview of the theory of HMM. A more detailed introduction to HMM is provided by Rabiner's tutorial [49].

9.1 Introduction

Hidden Markov models (HMM) are widely used in handwriting recognition at least in part due to their success in speech recognition. The basic theory of hidden Markov model was published in a series of classic papers by Baum and his colleagues at the Institute for Defense Analyses in the late 1960s [7]. Baker at Carnegie-Mellon University(CMU) implemented the theory in speech processing application in early 1970s [6]. Jelinek and his colleagues at IBM have also applied hidden Markov models to speech recognition in the 1970s [5]. Widespread application and understanding of the theory,however, was not reached until the 1980s. The speech recognition systems from AT&T, BBN and CMU were based on HMM, achieving superior results, as HMMs could effectively model time and space variance in speech signals. The success of HMM in speech recognition has generated strong interest towards its application in handwriting recognition [32] [64] [67]. For example:

- Bellegarda *et al.* [46] at IBM included local position, curvature and global information bearing into a feature set for the purpose of recognizing characters on a 81 character data corpus.

- Starner *et al.* [42] at BBN implemented angle, delta angle, delta x, delta y, pen lifts and sgn(x − max(x)) features in the recognition of handwritten words.
- Shu [51] at MIT used a vertical height, space and sub-stroke features in addition to Starner's features, thereby achieving reliable recognition.
- Winkler [72] used both on-line and off-line features in a HMM-based symbol recognizer. The on-line features included in the model were local position, sine and cosine value of the writing angle, and information on whether a point belongs to a pen-down stroke or pen-up stroke. Off-line features were extracted from images of handwritten symbols.
- Guillevic and Suen [21] used slope and position features extracted from images in their handwritten word recognition system.
- Lee *et al.* [37] designed a data-driven HMM topology for on-line handwriting recognition, using direction features in their model.
- Kosmala *et al.* [34] included analysis of sine and cosine of writing and differential angles, pen pressure and certain off-line features in their HMM-based mathematical expression recognition system.

Together with the positive results achieved with HMM, the above studies have shown that certain problems need to be addressed in order to achieve improved recognition. For instance, it is important to know which features should be used to represent a handwritten symbol and what kind of HMM topology benefits the recognition most.

In speech recognition, features representing the speech signal are now well understood. At present Mel-frequency cepstral coefficients and delta Mel-frequency cepstral coefficients are features used in HMM-based speech recognition [13]. It is still not clear however which features provide the best results in handwriting recognition.

In this chapter we summarize the theory of HMM as it can be applied to handwriting recognition.

9.2 Elements of a Hidden Markov Model

9.2.1 Discrete Markov Process

A Markov process is a stochastic process that satisfies a particular condition: the process's future behavior does not depend on its previous states but only on its current state. Let a discrete Markov process have a set of N states, $S_1, S_2, \ldots, S_N$. We denote the actual state at time t as q_t. The probability of the process being in state S_i at time t satisfies the following Markov condition:

$$P(q_t = S_i | q_{t-1} = S_j, q_{t-2} = S_k, \ldots) = P(q_t = S_i | q_{t-1} = S_j) \tag{9.1}$$

Figure 9.1 displays a Markov chain with four states.

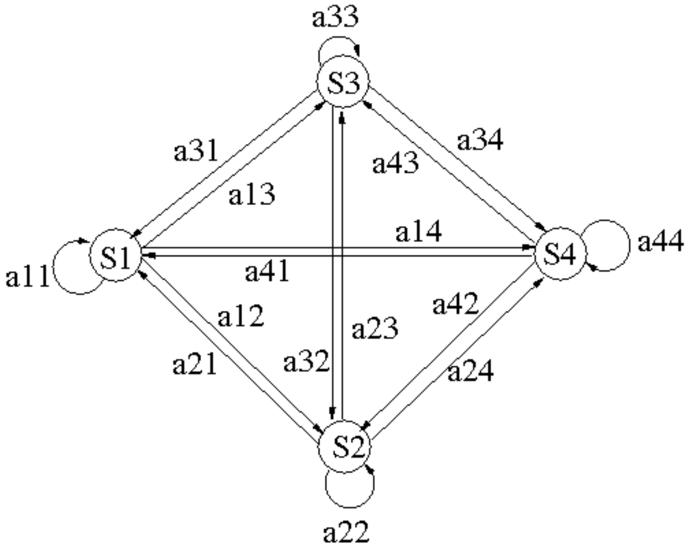


Figure 9.1: Markov Chain with Four States

The state transition probabilities a_{ij} are denoted as:

$$a_{ij} = P(q_t = S_i | q_{t-1} = S_j), 1 \leq i, j \leq N$$

and

$$\sum_{j=1}^{N} a_{ij} = 1$$

We denote the initial state probabilities as $\pi_i = P(q_1 = S_i)$.

The state transition probabilities and initial state probabilities can completely describe a discrete Markov process. The matrices A and $\prod$ shown below describe the discrete Markov chain shown in Figure 9.1.

$$A = a_{ij} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

$\prod = \pi_k = [\pi_1 \ \pi_2 \ \pi_3]$

If O denotes an observation sequence, then $O = S_i, S_j, S_k, S_l, \ldots$, where each state corresponds to an observable event.

9.2.2 Hidden Markov Models

In a discrete Markov process each observation symbol is associated with a state. This model is too restrictive and is not applicable to a range of problems. In a hidden Markov model, the observation is a probabilistic function of the state. The state sequence is not observable, but hidden. To illustrate the hidden Markov model, let us consider a urn and ball example [49]: There are 3 urns in a room. Each urn has many colored balls. The color of the balls are red, green, yellow and blue. A ball is randomly chosen from an urn, and its color is recorded as an observation. The ball is then replaced in the urn where it came from. A new urn is then selected, and the ball selection process is repeated. The process generates an observation sequence of colors without information of urns. Figure 9.2 illustrates the model. An HMM is characterized by the following parameters:

1. N, the number of states in the model. We denote the states as $S = S_1, S_2, \ldots, S_N$ and the state at time t as q_t.
2. M, the number of distinct observation symbols. The individual symbols are denoted $V = v_1, v_2, \ldots, v_M$.
3. the state transition probability distribution $A = a_{ij}$, and $a_{ij} = P(q_{t+1} = S_j | q_t = S_i), 1 \leq i, j \leq N$.
4. the observation symbol probability distribution in state i, $B = b_i(k)$, and

$b_i(k) = P(v_k \ at \ t|q_t = S_i), 1 \leq i \leq N, 1 \leq k \leq M$

5. the initial state distribution $\prod = \pi_i$, and

   $\pi_i = P(q_1 = S_i), 1 \leq i \leq N$

$\lambda = (A, B, \prod)$ denotes a HMM.

9.2.3 The Three Basic Problems of HMM

Given the idea of HMM reviewed in previous section, there are three basic problems to solve if the model is to be used in handwriting recognition.

Problem 1. Given an observation sequence $O = O_1O_2 \ldots O_T$ and a model $\lambda = (A, B, \prod)$, how is the probability $P(O|\lambda)$ of the observation sequence efficiently computed?

Problem 2. Given the observation sequence $O = O_1O_2 \ldots O_T$ and the model λ, how is one to choose a corresponding state sequence $Q = q_1q_2 \ldots q_T$ which is optimal in some meaningful sense(*i.e.*, that best "explains" the observation)?

Problem 3. How do we adjust the model parameters $\lambda = (A, B, \prod)$ to maximize $P(O|\lambda)$?

We will describe how to solve the above problems in detail in the next session.

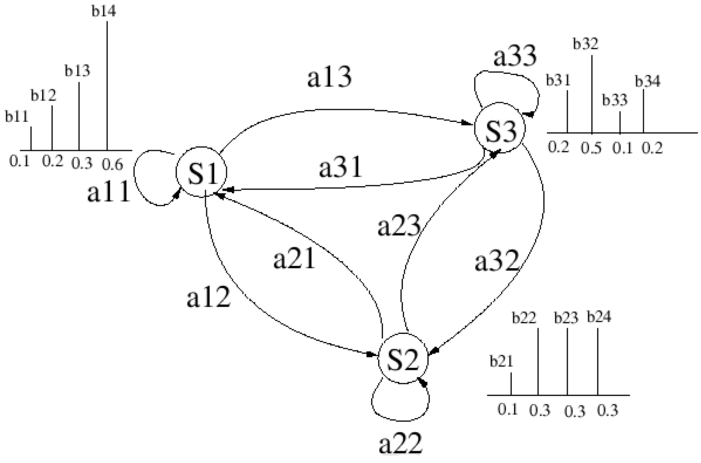


Figure 9.2: Three State HMM

9.2.4 Solutions to the Three Basic Problems of HMM

Calculating Probability of Observation Sequence

The first problem is an evaluation problem, *i.e.*, given a model and a sequence of observations, how do we compute the probability of the observation sequence produced by the model? This problem allows us to choose the model which best matches the observations.

The most straightforward way to calculate $P(O|\lambda)$ is enumerating all possible state sequences.

$$\begin{aligned} P(O|\lambda) &= \sum_{allQ} P(O,Q|\lambda) && (9.2) \\ &= \sum_{allQ} P(O|Q,\lambda)P(Q|\lambda) && (9.3) \end{aligned}$$

According to Equation 9.1, we can write,

$$P(Q|\lambda) = \pi_{q_1} a_{q_1 q_2} a_{q_2 q_3} \dots a_{q_{T-1} q_T} \tag{9.4}$$

The observation independence,

$$P(O|Q,\lambda) = \prod_{t=1}^{T} P(O_t|q_t,\lambda) \tag{9.5}$$

This gives,

$$P(O|Q,\lambda) = b_{q_1}(O_1)b_{q_2}(O_2)...b_{q_T}(O_T) \tag{9.6}$$

Equations 9.3, 9.4 and 9.6 then give,

$$P(O|\lambda) = \sum_{q_1,q_2,\dots,q_T} \pi_{q_1} b_{q_1}(O_1) a_{q_1 q_2} b_{q_2}(O_2) \dots a_{q_{T-1} q_T} b_{q_T}(O_T) \tag{9.7}$$

Equation 9.7 involves computational operations on the order of $2TN^T$. The computation becomes infeasible when the number of states, N, or the length of the observation sequence T increases.

Fortunately, an efficient algorithm exists in the so-called "forward-backward" procedure. Only the forward part is used in the solution. The forward variable $\alpha_t(i)$ is defined as,

$$\alpha_t(i) = P(O_1, O_2, O_3 \dots O_t, q_t = S_i|\lambda)$$

$\alpha_t(i)$ denotes the probability of the partial observation sequence, $O_1, O_2, \dots, O_t$ and state S_i at time t, given the model λ. We can solve $\alpha_t(i)$ inductively as follows [49]:

1) Initialization:

$$a_1(i) = \pi_i b_i(O_1), 1 \leq i \leq N \tag{9.8}$$

2) Induction:

$$a_{t+1}(j) = [\sum_{i=1}^{N} \alpha_t(i) a_{ij}] b_j(O_{t+1}), 1 \leq t \leq T-1, 1 \leq j \leq N \tag{9.9}$$

3) Termination:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{9.10}$$

Equations 9.8, 9.9 and 9.10 show how the probability is calculated. First, the forward probabilities are set to be equal to the joint probability of state S_i and initial observation O_1 (equation 9.8). Then in induction step, $\alpha_t(i)$ is calculated from 1 to T. Finally, the probability is calculated by summing all the forward variables at time T. The number of calculations involved is in the order of TN^2. Therefore, this algorithm is more efficient than the direct calculation of equation 9.7.

Finding a Best State Sequence

The second problem is the decoding problem to uncover the hidden part of the model by finding a best state sequence for an observation sequence. Theoretically, there is no exact solution. We cannot find a single "correct" state sequence. Instead we have attempted to find a near optimal state sequence for practical situations. The optimality criterion is to find a state sequence $Q = Q_1 Q_2 \dots Q_T$, that maximizes $P(Q, O|\lambda)$, the joint probability of the state sequence Q and the observation sequence O, given the model λ.

The Viterbi algorithm [49] is widely used to find the solution to Problem 2. The best score is defined along a single path, at time t, as:

$\delta_t(i) = \max_{q_1,q_2,\ldots,q_{t-1}} P[q_1, q_2 \ldots q_t = i, O_1, O_2 \ldots O_t|\lambda]$.

The quantity $\delta_t(i)$ can be computed recursively.

$$\delta_{t+1}(j) = \max_i(\delta_t(i)a_{ij})b_j(O_{t+1}) \tag{9.11}$$

The following procedure is used to get best state sequence.

1) Initialization:

$$\delta_1(i) = \pi_i b_i(O_1), 1 \leq i \leq N$$

$$\psi_1(i) = 0$$

2) Recursion:

$$\delta_t(j) = \max_{1\leq i\leq N}[\delta_{t-1}(i)a_{ij}]b_j(O_t),\ 2 \leq t \leq T, 1 \leq j \leq N$$

$$\psi_t(j) = \mathrm{argmax}_{1\leq i\leq N}[\delta_{t-1}(i)a_{ij}]$$

3) Termination:

$$\hat{P} = \max_{1\leq i\leq N}[\delta_T(i)]$$

$$\hat{q_T} = \mathrm{argmax}_{1\leq i\leq N}[\delta_T(i)]$$

4) Path(state sequence) backtracking:

$$\hat{q_T} = \psi_{t+1}(\hat{q_{t+1}}),\ t = T-1, T-2, \ldots, 1$$

To retrieve the state sequence, we need to keep track of the arguments that have maximized equation 9.11 for each t and j. The quantity $\psi_t(j)$ gives the state.

Adjusting Model Parameter

Problem 3 is a training problem. The model parameter λ is optimized to best describe the observation sequence, $O = O_1O_2 \ldots O_T$. The observation sequences used to adjust the model parameters are called training sequences. The training problem is a crucial one for most applications of HMMs since it allows the creation of the best model for our application. However it is also a difficult problem since there is no known analytical solution for this optimization. The Baum-Welch algorithm [49] is a well-known iterative procedure that gives a locally optimal solution to the training problem.

First, we define a backward variable $\beta_t(i)$ as:

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, O_{t+3} \ldots O_T, q_t = S_i|\lambda) \tag{9.12}$$

This denotes the probability of the partial observation sequence, $O_{t+1}O_{t+2}\ldots O_T$, given the state S_i at time t and the model λ. Like the forward variable $\alpha_t(i)$, it can also be calculated recursively. Next, we define the probability of being in state S_i at time t, and state S_j at time $t+1$, given the model and the observation sequence $\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j|O, \lambda)$.

The probability of being in state S_i at time t, given the model and the observation sequence, is denoted as $\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j)$.

From the definition of the forward and backward variables, we can write,

$$\begin{aligned} \xi_t(i,j) &= \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{P(O|\lambda)} && (9.13) \\ &= \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)} && (9.14) \end{aligned}$$

The summation of $\xi_t(i,j)$ over t ($t = 1, \ldots, T-1$) equals the expected number of transitions from state S_i to state S_j. The summation of $\gamma_t(i)$ over t is the expected number of transitions from state S_i. Using the above concepts and formulas, we can write the equations for the re-estimation of the HMM parameters as follows:

$$\overline{\pi_i} = \gamma_1(i)$$

$$\overline{a_{ij}} = \frac{\sum_{t=1}^{T-1}\xi_t(i,j)}{\sum_{t=1}^{T-1}\gamma_t(i)}$$

$$\overline{b_i(k)} = \frac{\sum_{t=1, O_t=v_k}^{T}\gamma_t(i)}{\sum_{t=1}^{T}\gamma_t(i)}$$

In this chapter, we have summarized the work in handwriting recognition using hidden Markov models. We have also reviewed the theory of HMM which includes Markov processes, the three basic problems and their solutions. We give a detailed description of how to use HMM in handwriting recognition in the next chapter.

Chapter 10

A HMM for Mathematical Characters

In this chapter we describe how we map handwritten mathematical characters to discrete observation symbols for an HMM. We also describe how we select and adjust the model parameters such as number of states and initial observation distributions.

10.1 The Architecture of Our Hidden Markov Model Recognizer

In the following sections, we describe our hidden Markov model-based recognizer in detail. Here we present an introductory outline of the model.

Figure 10.1 displays a diagram illustrating the recognition process. Handwritten symbols are input into a pre-processing module, which performs de-hooking, smoothing, re-sampling, stroke re-direction, stroke re-ordering and size normalization. (See Chapter 5 for descriptions of these operations.) The pre-processed symbols are then sent to the feature extractor module. We extract two sets of features: a positional feature set and a chain code based feature set. At the end of this step the handwritten symbols are represented in feature space as feature vectors. Since we use discrete hidden Markov model, we quantize the feature vectors, then calculate the sequences of observation symbols, which are to represent handwritten symbols. Each feature vector is mapped to a *codevector*. The set of all *codevector* is called the *codebook*. In the training, many data samples are used to generate the hidden Markov models. According to the codebook created in the vector quantization module, multiple observation sequences are generated from the training data. We use the Baum-Welch algorithm [49] for multi-observation sequences to re-estimate the model. We build a model for each symbol by training the model with approximately 1000 samples.

To recognize a given unknown symbol, we compute its observation symbols first, then test the sequence of observation symbols with each of the hidden Markov models. The probabilities for each model are then calculated and the model with highest probability is taken as the recognition result.

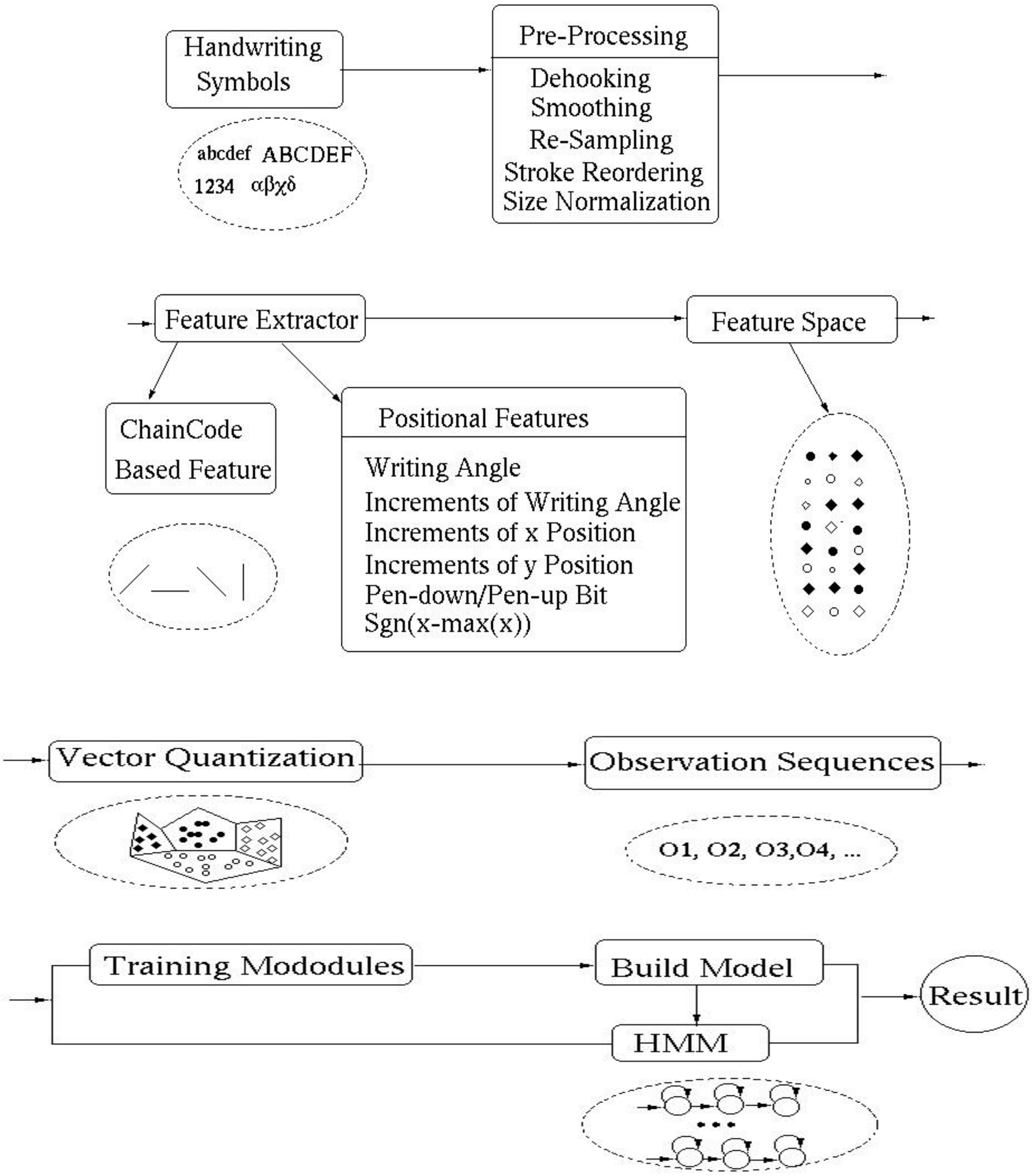


Figure 10.1: HMM Recognition Diagram

Symbol decomposition is introduced in the next section followed by description of how to represent the symbols using features. Then we introduce the observation symbol calculation and vector quantization. The section "HMM Topology" 10.4 describes the design of HMM.

10.2 Modeling of Handwritten Mathematical Symbols

This section describes how handwritten mathematical symbols have been modeled in this work. Two aspects have been considered: symbol decomposition and feature representation.

10.2.1 Decomposing Mathematical Symbols

The idea of decomposition was inspired by the approach used to analyze Chinese characters. A Chinese character is typically composed of several strokes, with the most complicated characters containing over 30 strokes. Still, every Chinese character can be decomposed into a set of basic strokes. There are only eighteen basic strokes in total, shown in Figure 10.2.

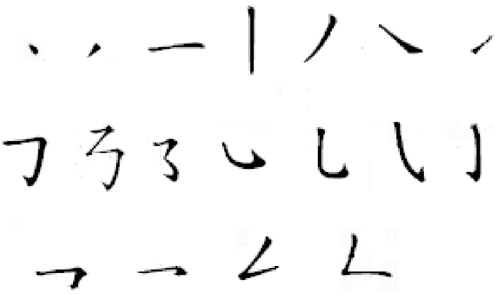


Figure 10.2: Chinese Strokes (from [77])

Similar to this approach, we have attempted to identify basic stroke elements for handwritten mathematical symbols and decompose symbols according to their basic elements. In our HMM design, we used the idea of decomposition. It has been pointed out by Lee [37] that the design of a model should be based on the modeled function with a number of parameters sufficient to handle function's complexity. In practice, the number of the model parameters cannot be increased arbitrarily. In our design of HMM for mathematical symbols, the number of states is based on the number of basic elements.

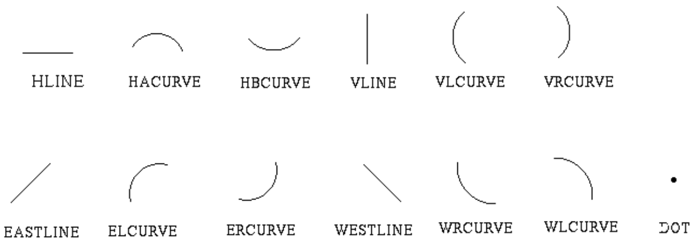


Figure 10.3: Curve-based Elements

Curvature-Based Decomposition

We use the 13 basic elements in Figure 10.3. This approach is extended from Zhao's method [74]. Zhao and his colleagues used 10 basic elements. After analyzing around 1000 samples of each symbol, it showed that our decomposition is suitable for mathematical symbols.

The basic elements are extracted by traversing the points along the symbol via the algorithm 10.2.1.

The SubElement(p_{i-2},p_i, p_{i+2}) function detects the element type formed by three points. The value $\cos(\theta)$ is used to distinguish lines from curves, where θ is the angle formed by the three points. The algorithm 10.2.1 is used to detect the element type.

The function call "findDirection(p_{i-2}, p_{i+2})" calculates a chain code between 0 and 7.

Figures 10.4 and 10.5 show the basic elements for an entire symbol after curved-based decomposition.

The decomposition is used in a HMM. The number of states of the character hidden Markov model will be the number of basic elements. Some researchers [34] [67] have adopted 5 or 7 as the number of states of their HMMs, which helps to achieve best results in their experiments. The disadvantage of using a constant number of states is the lack of flexibility when modeling the character. In our work, we have allowed the number of states is flexible since it is based on the symbol to be modeled. The results are shown in tables 11.2 and 11.3 and are explained in the Chapter "Experiments and Results", section 11.3.

The main disadvantage of this curve-based decomposition is its inability to precisely reconstruct the symbol without recording indexes. The curve length information is not included in the basic element definition. We record the beginning and ending indices of each basic element in the symbol, and then we retrace the strokes backwards to find the corresponding segments. Equal length-based decomposition overcomes this disadvantage.

```
if number of points $< 5$ then
    element type is DOT
else
    start from the third point
    foreach point $p_i$ in the symbol do
        element1=SubElement($p_{i-2}$,$p_i$, $p_{i+2}$)
        element2=SubElement($p_{i-1}$,$p_{i+1}$,$p_{i+3}$)
        if element1=element2 then
            initialElement=element1
            continue
        else
            found an element type: element1
            continue with point pi+1
        endif
    endforeach
endif
```

Algorithm 10.1: Basic Element Extraction

```
if $\cos(\theta) >= -1$ and $\cos(\theta) < PI_{THRESHOLD}$ then
    if $|(x0 - x2)| < LINE_{THRESHOLD}$ then
    Element Type = VLINE;
    if $|(y0 - y2)| < LINE_{THRESHOLD}$ then
    Element Type = HLINE;
    if $(x0 - x2) \times (y0 - y2) > 0$ then
    Element Type = WESTLINE;
    if $(x0 - x2) \times (y0 - y2) < 0$ then
    Element Type = EASTLINE;
    else Element Type = UNKNOWN;

else
    findDirection($p_{i-2}, p_{i+2}$);
    if direction is horizontal and is convex curve then
    Element Type is HBCURVE
    if direction is horizontal and is concave curve then
    Element Type is HACURVE
    if direction is vertical and $X_{p_i} > X_{p_{i-2}}$ then
    Element Type is VRCURVE
    if direction is vertical and $X_{p_i} > X_{p_{i-2}}$ then
    Element Type is VLCURVE
    if direction is east and is convex curve then
    Element Type is ERCURVE
    if direction is East and is concave curve then
    Element Type is ELCURVE
    if direction is west and is convex curve then
    Element Type is WLCURVE
    if direction is west and is concave curve then
    Element Type is WRCURVE

endif
```

Algorithm 10.2: Element Type Detection

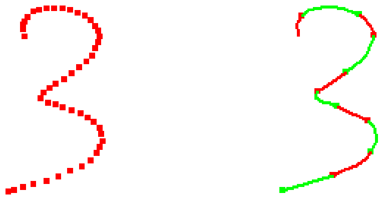


Figure 10.4: Curve-based Decomposed “3”

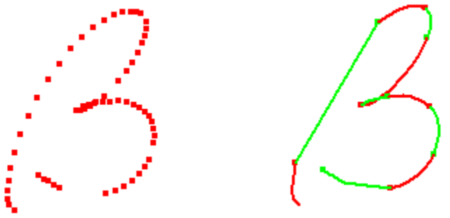


Figure 10.5: Curve-based Decomposed “β”

Equal Length-Based Decomposition

In equal length-based decomposition, all basic elements have equal arc lengths. There are 25 basic elements as shown in the following Figure 10.6.

We divide the stroke into *NSeg* segments, the number of segments desired. For each segment, we use θ to denote the angle between x axis and the line formed by the two end points. Then we execute the algorithm 10.2.1.

The threshold parameter “distance threshold” is used to detect a horizontal line. Positive and negative y may exist on one segment if the number of segments is small. In an attempt to avoid such situations, we performed a number of experiments to determine the proper number of segments.

The method of choosing basic elements should enable us to easily reconstruct any symbol when needed. In equal length-based decomposition, 24 basic elements are selected to represent direction information, and the length of each element is derived from the total length of a symbol and the number of segments. Based on the direction, length and shape (line, curve and dot) information, we are able to regenerate the symbol. The regenerated symbol defines the "standard" format of the handwritten symbol which can be used to obtain variance of handwriting symbols by applying certain deformation operations.

If a HMM model is built from an equal length-based decomposition, then the number of states will be the number of segments, which comes from the desired "standard" format.

Figures 10.7 and 10.8 show some examples of equal length-based decompositions.

The number of segments in these figures is 10. We tried different numbers, *e.g.*, 10, 5, 7, 4, 6, 20, *etc.* The regenerated symbols from small number of segments are far away from the original symbols. On the other hand, the bigger the number of segments, the more computation involved. We find 10 is suitable for our decomposition.

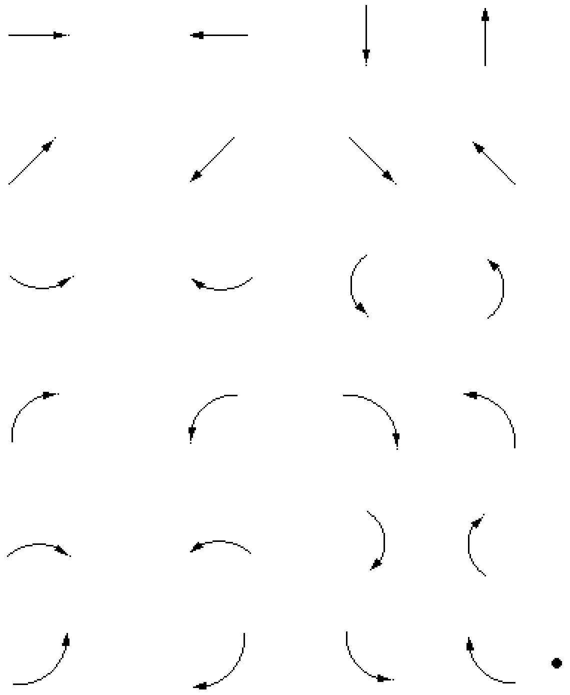


Figure 10.6: Encoding Elements

```
foreach divided segment ($p_b$ to $p_e$) do
    foreach point p, between begin index and end index do
        rotate the x axis to the line formed by two end points
        calculate the y coordinate of the point $p$
        $y = -(x_p - x_{p_b}) \times \sin(\theta) + (y_p - y_{p_b}) \times \cos(\theta)$
        check if $y > 0$ or not
        find the y which has maximum absolute value, $y_{max}$
        normalize the $y_{max}$, $normal_y = y_{max}/seg_{length}$
        find the direction between $p_b$ and $p_e$
        decide the segment type according to direction and sign of $y_{max}$
        e.g., if direction is 0, $normal_y < distance\,threshold$, and $y_{max} > 0$
        then the segment type is 8.
    endforeach
endforeach
```

Algorithm 10.3: Equal Length Decomposition

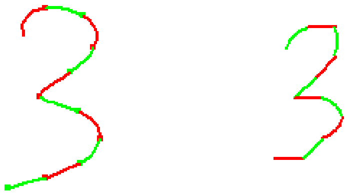


Figure 10.7: Equal Length Decoding of the Symbol “3”

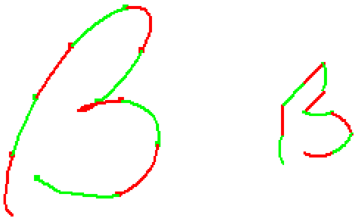


Figure 10.8: Equal Length Decoding of the Symbol “β”

10.2.2 Representing Handwritten Symbols by Using Features

Traditionally, direction and position features have been used within HMM-based handwriting recognition because HMM has the nature of modeling time sequential signals. With directional features, such as chain codes, we can rebuild the symbol. Similarly to directional features, positional features (such as x, y coordinates) are a time-dependent characteristic of a handwriting signal. With the modern hardware, the above information can be determined during handwriting, offering the promise of an improved recognition process. In addition to directional and positional features, we have also recorded and analyzed pen-down and space-related information as it is considered beneficial.

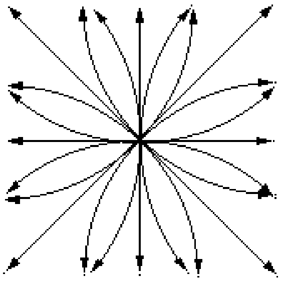


Figure 10.9: Pendown Features

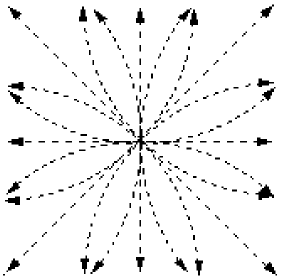


Figure 10.10: Penup Features

Feature Sets

We have used two sets of features within our recognizer. The first set contains direction and curve features, based on our equal length decoding. There are 24 features for pen-up strokes and as many features for pen-down strokes. We shall refer to the first set of features as *chaincode-based features*. The features are shown in Figure 10.9 and Figure 10.10. The second set of features include writing angle, the increments (delta) of the writing angle, the increments (delta) of the x and y position, the pen-up/pen-down bit, and weather x is greater than all of its previous points or not. The last two of these features have binary values, while the rest are real numbers. We shall call the second set of features *positional features*. The writing angle was defined in Chapter 6. The delta writing angle is equal to the difference between the writing angles of the current and the previous data points. The delta writing angle of the first data point is set to 0.0. By subtracting the x position of point $i-2$ from the x position of point $i+2$, we calculate the delta x position of point i. For the first two points, the point $i-2$ does not exist. Their own x positions are used instead. Likewise, for the last two points, the point $i+2$ is not defined, their x positions are used as the x position of point $i+2$. The procedure for obtaining the delta y position is identical.

$$\delta(x_i) = \begin{cases} x_{i+2} - x_i, & i = 0, 1 \\ x_{i+2} - x_{i-2}, & 2 \leq i \leq N-2 \\ x_i - x_{i-2}, & i = N-1, N \end{cases}$$

The next property is penup versus pendown. We assign 0 for the points on pen up stroke, 1 for the points on pen down stroke. The pen up stroke is invisible, but it captures the pen movement. It is as important as the pen down stroke.

The sgn(x − max(x)) represents the spatial relation in horizontal direction. It indicates whether the x position of the current point is greater or less than the x positions of all the preceding points. To compute the sgn(x − max(x)) feature, the maximum x position for all proceeding points need to be calculated. If $x < \max(x)$, feature value of the current point is set to be 0, or 1 when the opposite is true.

Figure 10.11 shows examples of this spatial feature. The dashed lines in the symbol "h" and "β" are smaller, while the solid lines are greater than the ones connecting its previous points.

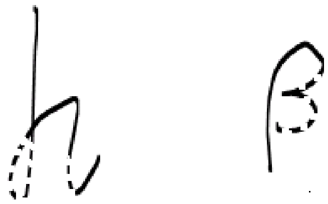


Figure 10.11: Spatial Relation

Simple spatial features, sgn(x − max(x)), is suitable for mathematical symbol recognition. While some researchers investigated spatial features in handwriting recognition by using somewhat elaborated spatial features. For example, Marukatat and Artieres [43] compared two types of spatial features: absolute and relative features. In Marukatat and Artieres's Korean character recognition system, they embedded spatial information within a function for the segmentation of the handwriting signal. $P_{spatial}(X|Q,\lambda) = P_{spatial}(seg_1,\ldots,seg_k|\lambda)$. In the case of absolute spatial features, the bounding box of the character was computed first. The center of each segment is calculated as $P_{spatial}(X|Q,\lambda) = \prod_{i=1}^{k} P_{Absolute}(seg_i/S_i)$. The relative spatial features are designed to describe the spatial information among strokes. For example, vertical position, horizontal position and connection are used to describe the strokes relation. The vertical position has the value of "above", "aligned" and "below". The horizontal position has the value of "left", "aligned" and "right". The connection has the value of "touching" and "not touching". This relation is expressed as $esr(seg_i, seg_j)$, and for simplicity, only the relation between any segment and the preceding segment is considered. While such approach can improve recognition rate, it is more suitable for recognizing multiple strokes symbols, such as Korean and Chinese characters. In mathematics, most of the symbols have only one stroke. Therefore we have adopted simple feature description, *i.e.* sgn(x − max(x)) to represent the spatial information.

Integrating Features into HMM

From the first set of features (*chaincode-based features*) we can rebuild the symbol. The second set of features (*positional features*) can also be used to reconstruct the symbol. HMM based on these two sets of features have different information about the pen trajectory. We use these two sets of features independently in two separate HMM, and then combine the results to achieve the final recognition result.

To describe how to integrate the features into HMM, let us assume the handwriting symbol X has N points, $X = (p_1, p_2, \ldots, p_N)$. The symbol X is decomposed into k segments, $X = (seg_1, seg_2, \ldots, seg_k)$. We use left-right HMMs. The probability of the N-lengthed handwriting signal X in the left-right HMM, λ, is computed by:

$$P(X|\lambda) = \sum_{Q} P(X|Q, \lambda) \times P(Q|\lambda) \tag{10.1}$$

For the chaincode-based features, the probability $P(X|Q, \lambda)$ is:

$$P(X|Q, \lambda) = P_{chaincode-based\ features}(X|Q, \lambda) \tag{10.2}$$

Using the independence assumption of the features in the above equation, we get:

$$P_{chaincode-based\ features}(X|Q, \lambda) = \prod_{i=1}^{k} P_{chaincode-based\ features}(seg_i|S_i) \tag{10.3}$$

where S_i is the ith state.

The initial distributions of the observation sequences' probability are obtained from the mapping relation between states and the decoded segments, as shown in Figure 10.12. The distribution of the line segments is accumulated from the training sample, then they are normalized and used as the initial parameters of the corresponding state. In Figure 10.12, symbol "α" has 9 segments. Its model also has 9 states. For each segment, the probability in the corresponding state is assumed to be a Gaussian distribution.

For the positional features, we compute the feature vectors for each point, defined as follows: A handwritten symbol is represented by a sequence of feature vectors, $(f_1, f_2, \ldots, f_N)$. Each feature vector f_i contains the six features. For each segment, let b_i, e_i represent the beginning point and end point respectively. The probability $P(X|Q, \lambda)$ is computed by:

$$\begin{aligned} P(X|Q,\lambda) &= \prod_{i=1}^{k} P(f_{b_i}^{e_i}|S_i) & (10.4) \\ &= \prod_{i=1}^{k} P(f_{b_i}, f_{b_{i+1}}, \dots, f_{e_i}|S_i) & (10.5) \\ &= \prod_{i=1}^{k} \prod_{t=b_i}^{e_i} P(f_t|S_i) & (10.6) \end{aligned}$$

The above equations are based on the assumption that segments are independent. In the same way as for the chaincode-based features, the distributions of the observation sequences' probabilities are obtained from the training samples.

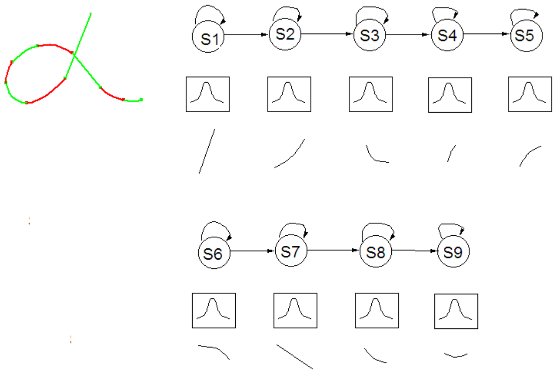


Figure 10.12: HMM and Initial Distribution for Symbol "α"

Combining Chaincode-Based Feature Set and Positional Feature Set

Each feature set we described above can be used to model a handwritten symbol. As we pointed out, the information is redundant, but if they work together, better recognition results can be achieved. Table 11.4 shows that combining features can improve the recognition rate by 6.5%.

We examined two ways to combine the features. First, we combined the two HMMs by computing the square root of the probabilities. Second, we combined the features directly and used the combined features to build one single HMM. These two approaches are illustrated in figures 10.13 and 10.14.

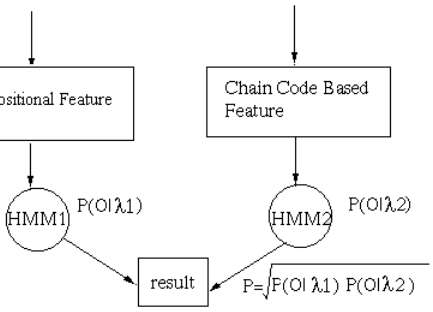


Figure 10.13: Combined HMM

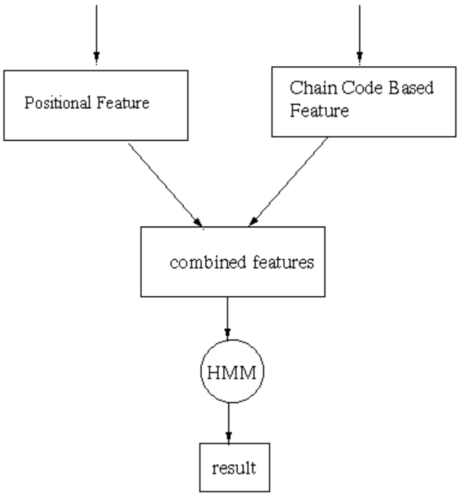


Figure 10.14: Combined Features

In the first approach, we build two HMMs. For each symbol X_1 of our alphabet $\{X_1, X_2, \ldots, X_n\}$, we generate two HMMs $\lambda_{X_1}^{f_{chaincode-based}}$, $\lambda_{X_1}^{f_{positionalfeature}}$ from the chaincode-based feature and positional feature respectively. Each HMM has its own parameters, such as the number of states, state transition probabilities and observation probabilities. The probabilities $P(X|\lambda_X^{f_{chaincode-based}})$, $P(X|\lambda_X^{f_{positionalfeature}})$ of symbol X for the two HMMs are calculated by the forward-backward algorithm. The final result is obtained from the combination of the two HMMs. The final probability of the handwriting signal on the combined HMM is calculated as:

$$P(X|\lambda_X) = \sqrt{P(X|\lambda_X^{f_{chaincode-based}}) \times P(X|\lambda_X^{f_{positionalfeature}})} \tag{10.7}$$

Figure 10.14 shows the direct combination of *chaincode-based features* and *positional features*. In the second combination method, we calculated the positional features and chain code based features individually. The two feature sets were combined to produce a combined feature set. This combined feature set was used to generate a HMM, which produces the final result. The probability of the handwriting signal is calculated as:

$$P(X|\lambda_X) = P(X|\lambda_X^{f_{chaincode-based}+f_{postionalfeature}}) \tag{10.8}$$

The experiments we performed on the two combination methods showed that the second combination method can improve the recognition rate, while the first one did not provide satisfactory results. We explain the experiments and the results for the second combination method in the section “Experiments and Results”, section 11.3.

10.3 Calculating Observation Symbols

We used discrete-density HMMs, requiring discrete observation symbols as input instead of feature vectors. To do this, the feature vectors need to be converted into observation symbols from a set of M discrete values, where M is determined by experiment.

For the chaincode-based HMM, we map the features to observation symbols directly. Recall that in our chaincode-based HMM, there are 48 features, 24 for pen-up and 24 for pen-down. We label these 48 curves as $(O_1, O_2, \ldots, O_{48})$, and use them for the observation sequences.

In the positional feature HMM, each point has a feature vector that includes six features. How do we convert this set of multi-dimensional feature vectors into M observation symbols?

We use a data compression technique, called *vector quantization* [3] to convert a multi-dimensional vector into a discrete symbol. We can view VQ as an approximator.

In one dimension, given a set of numbers, we would do the following: given the size of the observation sequence, M, find M numbers that can divide the original numbers evenly. Each original number is approximated by one of the new M numbers. The same rules apply in multiple dimensions. Figure 10.15 shows an example in 2D. Small dots are the training vectors, big circles are the codevectors, that represent the centroid of each domain. We use the algorithm created by Linde, Buzo and Gray [39] to compute the codevectors. The algorithm uses iterative domain splitting. An initial codevector is obtained by averaging the entire training sequences, then the initial codevector is split in two. An iterative algorithm is used on the two codevectors to calculate the distortion error, defined below. Splitting and iteration are repeated until there are the desired number of codevectors and distortion error.

We describe the algorithm here for completeness. The notation and algorithm are from a data compression tutorial [1]. Let $\Gamma = \{x_1, x_2, \ldots, x_N\}$ be the training sequence. Each vector x_i is k dimensional, *i.e.* $x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,k})$, $i = 1, 2, \ldots, N$. In our training data, $k = 6$. Let M be the number of codevectors, and denote the codebook as $C = (c_1, c_2, \ldots, c_M)$. Each codevector is also k dimensional, *i.e.* $c_i = (c_{i,1}, c_{i,2}, \ldots, c_{i,k})$, $i = 1, 2, \ldots, M$. Let S_n be the encoding region associated with codevector c_n and let $P = (S_1, S_2, \ldots, S_M)$ denote the partition of the space. If the vector x_m is in the encoding region S_n, then its approximation, denoted by $Q(x_m)$, is c_n: $Q(x_m) = c_n, \text{if } \mathrm{x_m \in S_n}$. The average distortion is given by $D_{ave} = \frac{1}{Nk}\sum_{m=1}^{N} \|x_m - Q(x_m)\|^2$. The algorithm is summarized below, from [1].

1. Given Γ, set distortion error ε to a very small number: 0.001.
2. Initialization: Let $M = 1$, and

$$c_1^* = \frac{1}{N}\sum_{m=1}^{N} x_m \tag{10.9}$$

Calculate

$$D_{ave}^* = \frac{1}{N}\sum_{m=1}^{N} \|x_m - c_1^*\|^2 \tag{10.10}$$

3. Splitting: For $i = 1, 2, \ldots, M$, set

$$c_i^{(0)} = (1+\varepsilon)c_i^* \tag{10.11}$$

$$c_{M+i}^{(0)} = (1-\varepsilon)c_i^* \tag{10.12}$$

   set $M = 2M$.

4. Iteration: let $D_{ave}^{(0)} = D_{ave}^*$. Set the iteration index $i = 0$.

   i For $m = 1, 2, \ldots, N$, find the minimum value of $\|x_m - c_n^{(i)}\|^2$, over all $n = 1, 2, \ldots, M$. Let n^* be the index which achieves the minimum. Set

$$Q(x_m) = c_{n^*}^{(i)} \tag{10.13}$$

   ii For $n = 1, 2, \ldots, M$, update the codevector

$$c_n^{(i+1)} = \frac{\sum_{Q(x_m)=c_n^{(i)}} x_m}{\sum_{Q(x_m)=c_n^{(i)}} 1} \tag{10.14}$$

   iii Set $i = i + 1$.

   iv Calculate

$$D_{ave}^{(i)} = \frac{1}{Nk} \sum_{m=1}^{N} \|x_m - Q(x_m)\|^2 \tag{10.15}$$

   v If $(D_{ave}^{(i-1)} - D_{ave}^{(i)})/D_{ave}^{(i-1)} > \varepsilon$, go back to step(i).

   vi Set $D_{ave}^* = D_{ave}^{(i)}$. For $n = 1, 2, \ldots, M$, set $c_n^* = c_n^{(i)}$ as the final codevectors.

5. Repeat steps 3 and 4 until the desired number of codevectors is obtained.

Let us use one dimensional vectors as an example. We have $x1 = 1$, $x2 = 2$, $x3 = 3$, $x4 = 4$ as our input sequences, and we want to find two codevectors for this sequence. First, $c_1^* = \frac{1}{4}\sum_{m=1}^{4} x_m = 2.5$, then we do the splitting: $c_1^{(0)} = (1+\varepsilon)c_1^* = 2.5025$, $c_2^{(0)} = (1-\varepsilon)(c_1^* = 2.4975$, next is the iteration: we calculate $Q(x_1) = c_2^{(0)}$, $Q(x_2) = c_2^{(0)}$, $Q(x_3) = c_1^{(0)}$, $Q(x_4) = c_1^{(0)}$, now we update the codevector: $c_1 = (x3 + x4)/2 = 3.5$, $c_2 = (x1 + x2)/2 = 1.5$. Our input sequences are quantized by two codevectors: c_1 and c_2.

We have calculated the codebook for $M = 64$. This value of M was chosen experimentally to give the best results. It needs to be calculated once and can then be used for all observation symbols. For an unknown symbol, the feature vectors are extracted and compared with the codevectors. The index of the closest codevector is selected as the observation symbol.

The experiments on different codebook sizes are explained in Section 11.3 "Experiments and Results".

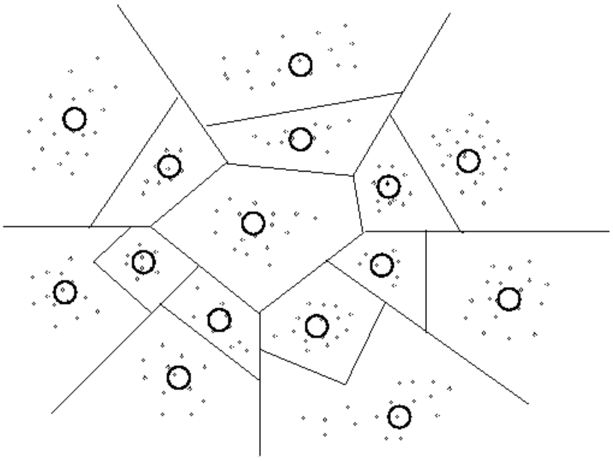


Figure 10.15: 2D Vector Quantization

10.4 HMM Topology

The usual approach to determine and adjust the HMM topology parameters has been empirical [17] [38]. An alternative approach was taken by Lee *et al.* who introduced a data driven design of HMM topology [37] for Hangul recognition. In data driven design, the HMM topology comes from the data to be modeled. We have taken a data-driven approach in our mathematical symbols recognition.

10.4.1 Number of States

As discussed above, the number of states is determined by the number of segments, which we get from the decomposition. From our design, we can see that the HMM is based on the handwritten symbol it modeled. Our results show that this design achieves better recognition result than traditional fixed number of states. This can be found in Section 11.3 “Experiments and Results”. Figure 10.16 shows the HMMs and their corresponding segments. The HMM is a left-right model. The number of states is determined by mapping each segment to a single HMM state. Then each state of the HMM corresponds to a segment of handwriting in time-sequential order.

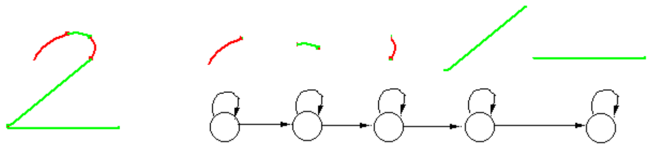


Figure 10.16: HMM States and Segments

10.4.2 Initial Observation Distribution

Choosing an initial distribution of observation sequence probabilities is important in the design of a HMM. If supplied with a proper initial distribution, the Baum-Welch algorithm [49] will converge quickly, resulting in fast recognition. In random and uniform initial distributions, the observation symbols are not associated with the states. In the Baum-Welch algorithm, the model parameters for the computation of probability are adjusted at each iteration. The process is repeated until the probability difference between two iterations reaches a given threshold. Using random and uniform initial distribution takes a large number of iterations to get the desired probability difference between two iterations. We have used a Gaussian distribution as the initial distribution for the Baum-Welch algorithm. For each state, the maximum distribution is assigned to the observation symbol that is associated with the state. Recall that we decomposed the symbol and associated each segment with the state. The observation symbols within each segment are associated with the corresponding state, thus building a map between the observation symbols and the states. From the uniform distribution experiments, we know that a flexible number of states achieves better recognition results than a fixed number of states. We have therefore used flexible number of states within the Gaussian distribution. In the implementation, for each symbol, the training samples have different number of decomposed segments. We take the largest as the number of states to use, then we use algorithm 10.4.2 to assign the probability of observation symbols in each state.

```
Data: N = max(number of observation symbols for all samples)
foreach sample in the training database do
    foreach segment in the sample do
        foreach state in the model do
            if the segment is in the state then
                set the frequency of the segment in the state
            else
                the frequency of the segment in the state is zero
            endif
        endforeach
    endforeach
endforeach
foreach state in the model do
    generate N Gaussian probability
    assign N Gaussian probability to N observation symbols according to their frequency in the state
endforeach
```

Algorithm 10.4: Initial Observation Distribution

We find the Gaussian distribution gives better recognition results than either the random or the uniform distributions. The details are shown in Section 11.3 “Experiments and Results”.

The choice between multiple or a single HMM to model a handwritten symbol presents an important design decision. Lee [37] pointed out that one model for a class has many benefits, for instance achieving a modular design of the recognizer. If we assign one model per symbol, the model can be easily replaced with another one. Using one model also makes post-processing much easier.

For these reasons, we tried to build one model for each handwritten symbol. Different models are created for each symbol variant. To combine these models, we created a dummy start and final states. From the starting state, we reach the first state of each model and go through each one until the final state is reached. Figure 10.17 shows a combined HMM model.

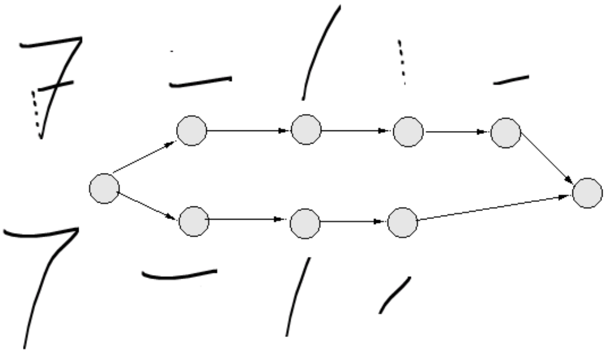


Figure 10.17: Combined HMM

In this chapter, we have described how we model handwritten mathematical symbols for a hidden Markov model. In particular, we have described the hidden Markov model topology together with the calculation of observation symbols which may occur in each model state.

Chapter 11

Implementation, Experiments and Results

In the last chapter we described HMM topology. In this chapter, we introduce our implementation of a HMM based recognizer. The mathematical symbols databased used in present work is also described. In the last section, we present the experiments with our proposed recognizer and the results is also given.

11.1 Implementation of the HMM Based Recognizer

We now discuss the software architecture of our implementation. Figure 11.1 and Figure 11.2 show the major blocks in its organization.

11.1.1 Ink Processing

In off-line recognition, ink is stored in a data file, and read to build an *Ink* object. For on-line handwriting, recognition is performed upon termination of writing. The ink information is accessed directly from memory to build an *Ink* object. An *Ink* object has one or multiple *Stroke* objects. A *Stroke* contains many *Points*. After we build the *Ink* object, we apply the preprocessing operations, such as re-sampling, smoothing, re-ordering *etc*, to the ink object which in turn calls the related operations on its strokes. The *Point*, *Stroke* and *Ink* classes are shared between the elastic and HMM based recognizers.

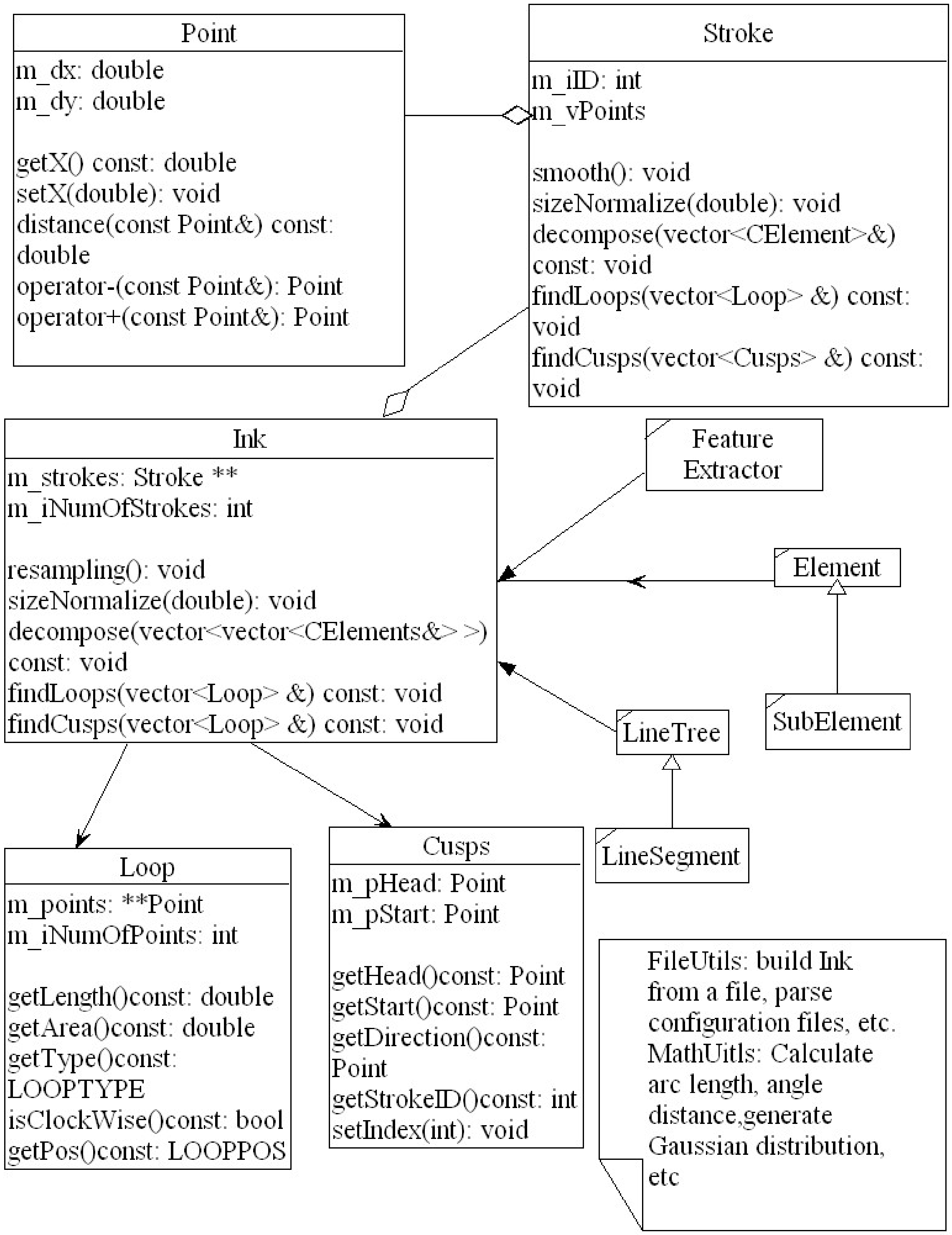

Point
m_dx: double
m_dy: double
getX() const: double
setX(double): void
distance(const Point&) const: double
operator-(const Point&): Point
operator+(const Point&): Point
Stroke
m_iID: int
m_vPoints
smooth(): void
sizeNormalize(double): void
decompose(vector<CElement>&) const: void
findLoops(vector<Loop> &) const: void
findCusps(vector<Cusps> &) const: void
Ink
m_strokes: Stroke **
m_iNumOfStrokes: int
resampling(): void
sizeNormalize(double): void
decompose(vector<vector<CElements&> >) const: void
findLoops(vector<Loop> &) const: void
findCusps(vector<Loop> &) const: void
Feature Extractor
Element
SubElement
LineTree
LineSegment
Loop
m_points: **Point
m_iNumOfPoints: int
getLength()const: double
getArea()const: double
getType()const: LOOPTYPE
isClockWise()const: bool
getPos()const: LOOPPOS
Cusps
m_pHead: Point
m_pStart: Point
getHead()const: Point
getStart()const: Point
getDirection()const: Point
getStrokeID()const: int
setIndex(int): void
FileUtils: build Ink from a file, parse configuration files, etc.
MathUitls: Calculate arc length, angle distance,generate Gaussian distribution, etc

Figure 11.1: Diagram of the Major Classes of the Recognizer (1)

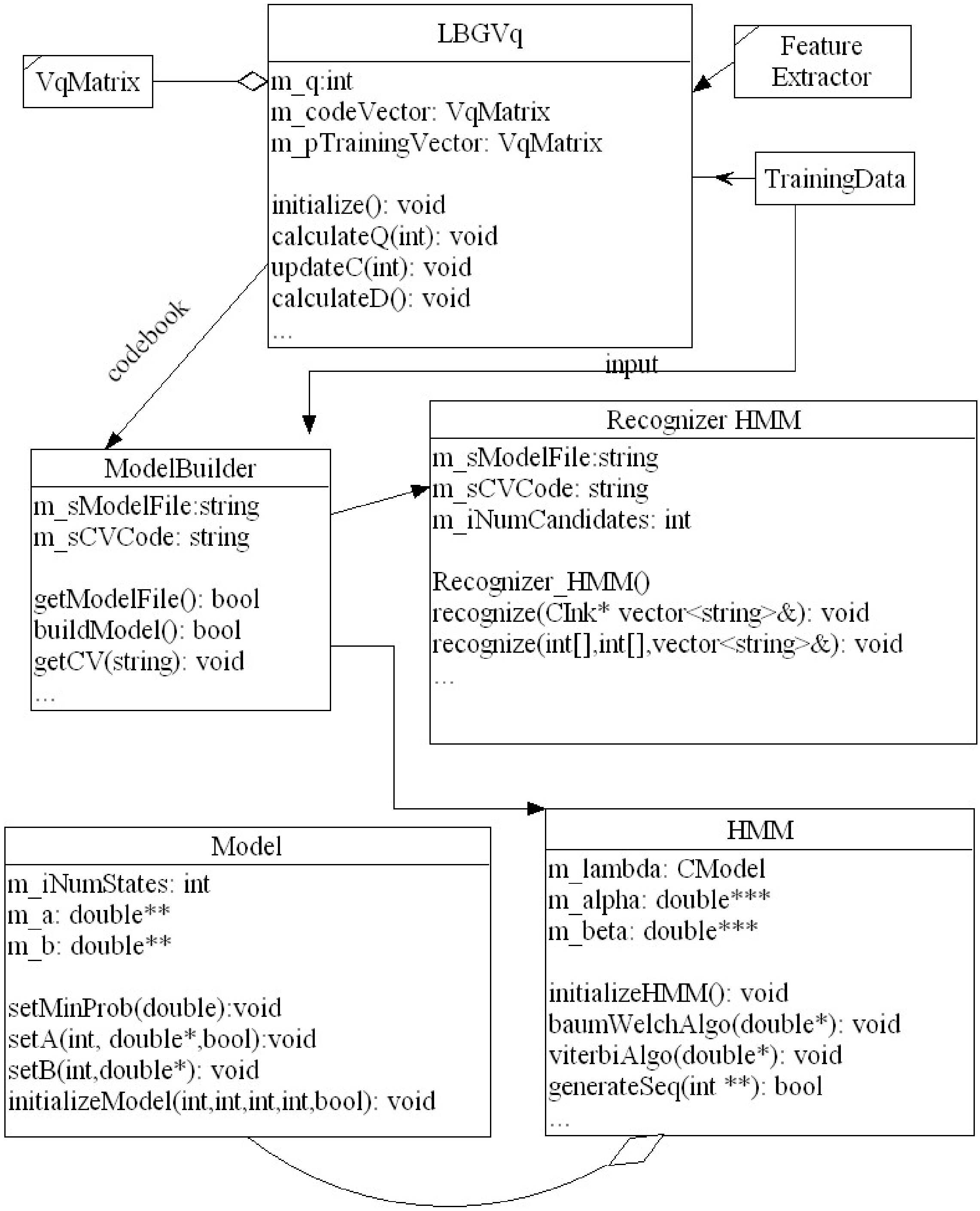


Figure 11.2: Diagram of the Major Classes of the Recognizer (2)

11.1.2 Ink Representation

Many feature sets, such as loops, cusps, intersections and so on are defined in the *FeatureExtractor* module. For the HMM based recognizer, we use two sets of features, defined in the *FeatureExtractor* class, namely the positional features and the decomposed features. The classes *Element* and *SubElement* are used for decomposition. The feature size is 7 (positional features plus decomposed features). We create a feature matrix for each symbol in the training database, *i.e.* data from Unipen and half of our collection. If the number of points for an ink sample is n, the feature matrix for this symbol will be of size $n \times 7$. These feature matrices build our feature space. In the *LBGvq* module, we generate a codebook from the feature space. For an unknown symbol, its feature matrix is quantified to discrete symbols according to the codebook. The discrete symbols are the observation symbols of the hidden Markov model. The sequence of these observation symbols describes the ink trace.

11.1.3 Model Parameter Selection and Implementation

We use a flexible number of states for the model. For example, to build a model for the symbol "α", all of the sample "α" symbols in the training database are decomposed and then the largest number of segments, N, is selected from the decomposition results. N is assigned as the number of states of model "α". We can choose from uniform, random or Gaussian distribution to build the observation symbol probability distribution of the states. The uniform and random distributions are simple: each $b_i(O(t))$ needs to be assigned the value of $1/N$ for uniform distribution, while in random distribution, we generate $M \times N$ random positive numbers which are less than 1 and assign these values to each $b_i(O(t))$. For a Gaussian distribution, we go through all of the sample "α" symbols and count the frequency of its observation symbols in each state. The total frequencies distribution in each state, $f_i(O(t))$, where i is from 1 to N, are calculated. For each observation symbol $O(t)$, we generate N probabilities which are Gaussian distribution, $p_i(O(t))$, where i ranges from 1 to N. We map state i with the largest frequency for $O(t)$, $f_i(O(t))$, to the maximum Gaussian distribution. In other words, probability of $O(t)$ in state i has the value of our largest Gaussian distribution value. If state j has the second largest frequency for $O(t)$, then probability of $O(t)$ in state j is the second largest Gaussian distribution value. This assignment is applied repeatedly to all probabilities and frequencies that follow. At the end, the resulting probability distribution for all of the observation symbols is Gaussian.

In the implementation of a HMM, we need to consider scaling, threshold selection, type of HMM, training *etc.* In the forward-backward algorithm, to compute $\alpha_t(i)$ we need two coefficients a_{ij} and $b_j(O_t)$, where each of these coefficients is significantly less than 1. As t increases, each term of $\alpha_t(i)$ tends toward zero and may exceed the precision range of a machine. To avoid this, we performed scaling on α and β (refer to the *Scaling* section 11.1.4). We set the minimum probability threshold to 0.01 with a maximum iteration count of 200 so that the iterations in Baum-welch training algorithm will stop if the probability difference reaches 0.01 or the number of iterations reaches 200. The type of HMM we used is left-right model with multiple observation sequences for training purpose.

11.1.4 Scaling

Therefore, in order to calculate $\alpha_t(i)$ correctly without exceeding the precision range of a machine. We need to scale α and β.

The scaling procedure multiplies $\alpha_t(i)$ and $\beta_t(i)$ by a scaling coefficient that is independent of i. The scaled forward variable is :

$$\begin{aligned}\hat{\alpha}_t(i) &= \frac{\alpha_t(i)}{\sum_{j=1}^{N}\alpha_t(j)} && (11.1)\\ &= C_t\alpha_t(i) && (11.2)\end{aligned}$$

To recursively compute $\hat{\alpha}_t(i)$, we have the following:

$$\begin{aligned}\bar{\alpha}_1(i) &= \alpha_1(i) && (11.3)\\ \bar{\alpha}_{t+1}(j) &= \sum_{i=1}^{N}\hat{\alpha}_t(i)a_{ij}b_j(O_{t+1}) && (11.4)\\ c_{t+1} &= \frac{1}{\sum_i \bar{\alpha}_{t+1}(i)} && (11.5)\\ \hat{\alpha}_{t+1}(i) &= c_{t+1}\bar{\alpha}_{t+1}(i) && (11.6)\end{aligned}$$

The same scaling procedure applies to backward variable $\beta_t(i)$.

11.1.5 Training

The training procedure was designed to maximize the probability of the observation sequence by adjusting the parameter $\lambda = (A, B, \prod)$. Our database includes Unipen data as well as data collected at the ORCCA lab. Half of the data set, which includes ORCCA and Unipen data, was used for training. We use the Baum-Welch algorithm to adjust the model parameters.

Since we use multiple observation sequences to train HMMs, the re-estimating procedure in Baum-Welch algorithm needs to be modified. We denote the set of K observation sequences as: $O = [O^{(1)}, O^{(2)}, \ldots, O^{(k)}]$, where $O^{(k)} = [O_1^{(k)}, O_2^{(k)}, \ldots, O_{T_k}^{(k)}]$ is the kth observation sequence and $1, 2, \ldots, T_k$ are kth sequence indexes. Our goal is to adjust the parameters of the model λ to maximize:

$$\begin{aligned} P(O|\lambda) &= \prod_{k=1}^{K} P(O^{(k)}|\lambda) && (11.7) \\ &= \prod_{k=1}^{K} P_k && (11.8) \end{aligned}$$

We assume that the observation sequences are independent of each other. By adding together the individual frequencies of occurrence for each observation sequence, we calculate the modified re-estimating formulas for $\bar{a}_{ij}$ and $\bar{b_j}(l)$ as:

$$\bar{a_{ij}} = \frac{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \hat{\alpha}_t^k(i) a_{ij} b_j(O_{t+1}^{(k)}) \hat{\beta}_{t+1}^k(j)}{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \hat{\alpha}_t^k(i) \hat{\beta}_t^k(i)} \tag{11.9}$$

$$\bar{b_j}(l) = \frac{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1, s.t. O_t = v_l}^{T_k-1} \hat{\alpha}_t^k(i) \hat{\beta}_t^k(i)}{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \hat{\alpha}_t^k(i) \hat{\beta}_t^k(i)} \tag{11.10}$$

11.1.6 Recognition

The goal of recognition is to find the HMM model that results in a maximum probability for correctly matching the unknown symbol. The recognition process uses extensively the forward-backward algorithm (defined in section 9.2.3 *The Three Basic Problems of HMM*).

For each symbol in the training database, we build a HMM. The training procedure forces the HMM to generate the maximum probability for the symbol it models. For an unknown symbol, we extract the feature vectors and apply the vector quantization procedure. The output of VQ is a sequence of observation symbols. This observation sequence represents the unknown symbol and becomes the input of each HMM. For each HMM, we compute the probability of the observation sequence. The maximum probability is calculated and the label of the corresponding HMM is the recognized symbol.

11.2 The Mathematical Symbol Database Used in the Present Work

As mentioned in Chapter 4, we have built the ORCCA mathematical handwriting database since such data was not available from any other source. Initially, we collected data with an IBM Crosspad. Due to the limitations of the device, such as no pen down information, we used a Tablet PCs to collect further data. Our database has 301 mathematical symbols and 68 formulas, including matrix notation. It was designed to cover most of the mathematical symbols used in the fields of applied mathematics and engineering. The database also contains different styles of mathematical symbols. For example, we have collected script letters as well as open face letters. Table 11.1 shows an excerpt of the data.

The data includes information about x, y coordinates, pressure, stylus status and timing. The stylus status is 1 or 0, indicating whether the stylus is on or off the screen respectively. Pressure varies from 0 to 255. A time stamp is recorded at the beginning and end of each stroke.

The questionnaire takes about 20 minutes to complete. Each questionnaire provides to approximately 1MB of data. We collected 70 samples, which were insufficient for HMM training and thus we use a subset of Unipen Train-R01/V07 benchmarks 1a, 1b and 1c [23] which contain digits, upper case letters and lower case letters in addition to our database. Half of our data and of the Unipen subset were used as training data, while the other half was used as testing data.

Alphanumeric	0-9, a-z, A-Z
Greek Lowercase Letters	α, β, γ, δ, ϵ, ε, ζ, η, θ, ϑ, ι, κ, λ, μ, ν, ξ, o, π, ϖ, ρ, σ, ς, τ, υ, ϕ, φ, χ, ψ, ω
Greek Uppercase Letters	Γ, Δ, Θ, Λ, Ξ, Π, Σ, Υ, Φ, Ψ, Ω
Calligraphic Letters	$\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$, $\mathcal{E}$, $\mathcal{F}$, $\mathcal{G}$, $\mathcal{H}$, $\mathcal{I}$, $\mathcal{J}$, $\mathcal{K}$, $\mathcal{L}$, $\mathcal{M}$, $\mathcal{N}$, $\mathcal{O}$, $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{R}$, $\mathcal{S}$, $\mathcal{T}$, $\mathcal{U}$, $\mathcal{V}$, $\mathcal{W}$, $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{Z}$
Script Uppercase Letters	$\mathscr{A}$, $\mathscr{B}$, $\mathscr{C}$, $\mathscr{D}$, $\mathscr{E}$, $\mathscr{F}$, $\mathscr{G}$, $\mathscr{H}$, $\mathscr{I}$, $\mathscr{J}$, $\mathscr{K}$, $\mathscr{L}$, $\mathscr{M}$, $\mathscr{N}$, $\mathscr{O}$, $\mathscr{P}$, $\mathscr{Q}$, $\mathscr{R}$, $\mathscr{S}$, $\mathscr{T}$, $\mathscr{U}$, $\mathscr{V}$, $\mathscr{W}$, $\mathscr{X}$, $\mathscr{Y}$, $\mathscr{Z}$
Open Face Letters	$\mathbb{A}$, $\mathbb{B}$, $\mathbb{C}$, $\mathbb{D}$, $\mathbb{E}$, $\mathbb{F}$, $\mathbb{G}$, $\mathbb{H}$, $\mathbb{I}$, $\mathbb{J}$, $\mathbb{K}$, $\mathbb{L}$, $\mathbb{M}$, $\mathbb{N}$, $\mathbb{O}$, $\mathbb{P}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{S}$, $\mathbb{T}$, $\mathbb{U}$, $\mathbb{V}$, $\mathbb{W}$, $\mathbb{X}$, $\mathbb{Y}$, $\mathbb{Z}$
Relations and Their Negations	$<$, $>$, $\leq$, $\geq$, $\ll$, $\gg$, $\subset$, $\supset$, $\subseteq$, $\supseteq$, $\sqsubset$, $\sqsupset$, $\sqsubseteq$, $\sqsupseteq$, $\sqcap$, $\sqcup$, $\vdash$, $\dashv$, $\top$, $\perp$, $\Vert$, $\vert$, $\in$, $\ni$, $\doteq$, $\sim$, $\approx$, $\simeq$, $\cong$, $\equiv$, $\propto$, $\not<$, $\not>$, $\not\leq$, $\not\geq$, $\not\subset$
Arrows and Pointers	$\leftarrow$, $\rightarrow$, $\uparrow$, $\downarrow$, $\Leftarrow$, $\Rightarrow$, $\Uparrow$, $\Downarrow$, $\leftrightarrow$, $\Leftrightarrow$, $\hookleftarrow$, $\hookrightarrow$, $\rightleftharpoons$, $\leadsto$, $\nearrow$, $\searrow$, $\swarrow$, $\nwarrow$
Mathematical Accents	a', a'', $\dot{a}$, $\ddot{a}$, $\hat{a}$, $\bar{a}$, $\tilde{a}$, $a^\dagger$, $\vec{a}$, $\mathring{a}$
Other Binary Operators	$\pm$, $\mp$, $\times$, $\div$, $\cap$, $\cup$, $\vee$, $\wedge$, $\uplus$, $\oplus$, $\otimes$, $\odot$, $\circ$, $\bigcirc$, $\cdot$, $*$
Various Other Symbols	$\aleph$, $\hbar$, $\wp$, $\mho$, $\Re$, $\Im$, ∂, ∇, $\sum$, $\int$, $\prod$, $\coprod$, $\surd$, $\emptyset$, $\forall$, $\exists$

Table 11.1: ORCCA Mathematical Symbol Data Set

11.3 Experiments and Results

Table 11.2 and Table 11.3 show an improved recognition rate when using dynamic number of states as compared to a constant number of states. In Table 11.2, we use positional features (e.g. writing angle, delta writing angle, delta of x and y position, pen-up/pen-down bit and sgn(x − max(x))) described in Chapter 7. For the same number of observation symbols, *i.e.* codebook size, we compare the recognition rate of a HMM using dynamic number of states with a HMM using 7 states and the same scenario when 5 and 3 states are used. The results show that for 16 codebook size and dynamic number of states, the recognition rate is improved 6.37% over 16 codebook size and 3 states. The recognition rate of 64 codebook size and dynamic number of states is improved 2.53% over 64 codebook size and 7 states. We performed the same experiments using chain code based feature. The results are shown in Table 11.3. From the table we can see that the dynamic number of states gives better performance than constant number of states. For 64 codebook size, the recognition rate is improved 2.23% over 7 states, 1.28% over 5 states, 2.03% over 3 states. For 32 codebook size, the recognition rate is improved 2.59% over 7 states, 4.06% over 5 states, 3.66% over 3 states. For 16 codebook size, the recognition rate is improved 5.24% over 7 states, 5.99% over 5 states, 4.14% over 3 states.

Recall that we used two combination methods on two feature sets. In the first combination, we built HMM for each feature set. Then a combined HMM is derived from the two individual HMMs. The final probability is the square root of the two probabilities. In the second combination, we combine the two feature sets directly to form a combined feature set. A single HMM is built based on the combined feature set. The probability is calculated from the combined feature set. We implemented the two combination methods and performed experiments on our database. The result for the first combination is not satisfactory. In the first combination, the recognition rate is slightly lower than the one achieved by positional features only and higher compared to the case when only the chain code based feature was utilized. Therefore we discarded this combination method. The second combination achieved better results. It can improve the recognition rate for both positional feature and chain code based feature. From Table 11.4 we can see that the recognition rate is improved 6.47% over positional feature. We also compared the result of the second combination with the first combination. The recognition improvement is shown in the second row of Table 11.4. The recognition rate of the second combination is improved 7.07% over the first combination.

We performed vector quantization to form observation symbols for discrete HMMs. It is important to note that the size of codebook is an important parameter, which can affect recognition results. We performed experiments on different codebook sizes. The results are shown in Table 11.5 and Table 11.6 for positional features and chain code-based features respectively. The first column gives the number of states. The second column demonstrates the recognition improvement of a 64 codebook size over a 32 codebook size. The third column shows the recognition improvement of 32 codebook size over 16 codebook size. For example, if we use 7 states and the positional feature set, the recognition rate of a 64 codebook size is improved 5.5% over a 32 codebook size. From the tables we see that a 64 codebook size achieves the best recognition rate among 64, 32 and 16 codebook sizes. This conclusion holds for both positional features and chain code-based features.

Table 11.7 shows a comparison of using a Gaussian distribution with using random distribution of the observation symbols. The experiments are performed on chain code-based features for different codebook sizes.

The second column shows an improvement in recognition rate. For a codebook of size 16, Gaussian distribution performs better by 2.4% compared to a random distribution. For a codebook of size 32, the improvement is 3.8%, while for a codebook of size 64, it is 2.9%.

Tables 11.2 and 11.3 show that a flexible number of states achieves better recognition rate than a constant number of states. We also performed experiments on different constant numbers of states. Table 11.8 shows the comparison among 7, 5 and 3 states for positional feature-based models. The second column is the recognition improvement of 7 states over 5 states. The third column is the recognition improvement of 5 states over 3 states. For the case in which the size of codebook is 16, the sub-case of 7 states performs by 2.2% better than the sub-case of 5 states, while the latter one outperforms the sub-case of 3 states by 0.9%. For the case in which the size of codebook is 32, the sub-case of 7 states achieves best results with 0.8% improvement over the 5 states sub-case, and 5 states improves by 5.4% over the 3 states sub-case. For the case in which the size of codebook is 64, the trend is similar with 7 states sub-case showing 2.9% improvement than 5 states sub-case, itself improving the performance of the 3 states sub-case by 3.4%.

Size of Code Book	Dynamic States vs. 7 States	Dynamic States vs. 5 States	Dynamic States vs. 3 States
16	+3.3%	+5.5%	+6.4%
32	+2.4%	+3.2%	+8.6%
64	+2.5%	+5.4%	+8.8%

Table 11.2: Dynamic States vs. Constant States: The Case of Positional Feature

Size of Code Book	Dynamic States vs. 7 States	Dynamic States vs. 5 States	Dynamic States vs. 3 States
16	+5.2%	+6.0%	+4.1%
32	+2.6%	+4.1%	+3.7%
64	+2.2%	+1.9%	+2.0%

Table 11.3: Dynamic States vs. Constant States: The Case of Chain-Code Based Features

Number of States	7
Size of Code Book	64
Second Combination vs. Positional Features	+6.5%
Second Combination vs. First Combination	+7.1%

Table 11.4: Combining Feature Sets

Number of States	64 vs. 32	32 vs. 16
3	+5.4%	+6.9%
5	+3.4%	+11.4%
7	+5.5%	+10.0%
Dynamic	+5.6%	+9.1%

Table 11.5: Different Code Book Sizes: The Case of positional features

Number of States	64 vs. 32	32 vs. 16
3	+1.6%	+9.4%
5	+2.8%	+10.9%
7	+0.4%	+11.6%
Dynamic	+1.0%	+9.0%

Table 11.6: Different Code Book Sizes: The Case of Chain Code Based Feature

Size of Code Book	Gaussian vs. Random
16	+2.4%
32	+3.8%
64	+2.9%

Table 11.7: Gaussian Distribution vs. Random Distribution: The Case of Chain Code Based Feature

Size of Code Book	7 States vs. 5 States	5 States vs. 3 States
16	+2.2%	+0.9%
32	+0.8%	+5.4%
64	+2.8%	+3.4%

Table 11.8: Number of States: The Case of 7 States, 5 States and 3 States

Chapter 12

Other Recognition Methodologies

12.1 Introduction

There are a number of other methodologies for pattern recognition that might be applied to handwriting. We have explored one of them to see if it was suitable for recognition in large sets of mathematical symbols. We studied the subspace classification method and implemented a prototype recognizer based on this method.

Initially, the subspace method was used for data compression and reconstruction [36]. In the case of multidimensional data, one extracts only the principle components, which form a subspace within the feature space. The resulting subspace is used to represent the multidimensional data. The subspace method is now used in pattern classification field [15]. First, a subspace for each class is built from all of the training data by extracting predefined principle components. The subspace is the image of a linear transformation of the feature space, which is formed by the pattern and its variations. After this, the feature vector of an unknown class is extracted. The feature vector contains features representing the pattern. For example, the feature vector of a handwritten symbol can be a vector of x and y coordinates. The distances from the feature vector to the subspace of each class are calculated. Finally, the minimum distance choice is selected as the classification result.

12.2 Subspace Classification

For handwriting symbols, we build the feature vector and use x and y coordinates as our features. We use symbol α as an example to describe the process of subspace recognition. If we extract x, y coordinates from symbol α, the feature vector for α is:

$$u_\alpha = [x_0, x_1, \ldots, x_n, y_0, y_1, \ldots, y_n] \tag{12.1}$$

Assuming the number of training data samples is M, the M training vectors is $\{u_0, u_1, \ldots, u_M\}$. The correlation matrix U is defined as :

$$U = \frac{1}{M}\sum_{i=1}^{M} u_i u_i^{'} \tag{12.2}$$

The principal components are calculated by finding the eigenvectors of U:

$$Uv_j = \lambda_j v_j \tag{12.3}$$

We can find up to M eigenvectors for each class i, $v_j^i, j = 1, \ldots, M$.
Subspace $\mathcal{L}$ is defined on these eigenvectors as:

$$\begin{aligned} \mathcal{L} &= \mathcal{L}_U \\ &= \{x|x = \sum_{i=1}^{M} c_i v_i, c_i \in \Re\} \end{aligned}$$

This gives the subspace on the whole space $\Re$ for M vectors $v_i, i = 1, \ldots, M$.
For an unknown symbol a, the feature vector is x_a. The distance from the vector x_a to a class i is the orthogonal distance $D_i^\perp$ from x_a to the subspace of class i:

$$D_i^{\perp^2} = \|x_a\|^2 - \|\hat{x}_a^i\|^2 \tag{12.4}$$

where $\hat{x}_a^i$ is the projection of x_a into the subspace of class i. The projection into the subspace is the sum of the projections along the eigenvectors as the eigenvectors are orthogonal. Therefore, using a dot product, we compute the projection as:

$$\hat{x}_a^i = \sum_{M}^{j=1} x_a^{'} v_j^i \tag{12.5}$$

Figure 12.1 shows the projection of vector x into subspace $\mathcal{L}$.

To recognize an unknown symbol a, we find the minimum distance D, and assign the label of the class as the final recognition result. In other words, we have the following:

$$C = \mathrm{argmin_j}(\|\mathrm{x_a}\|^2 - \|\hat{\mathrm{x}}_\mathrm{a}^\mathrm{i}\|^2) \tag{12.6}$$

Equation 12.6 is equivalent to Equation 12.7:

$$C = \mathrm{argmax_j} \|\hat{\mathrm{x}}_{\mathrm{a}}^{\mathrm{i}}\|^2 \tag{12.7}$$

The meaning of Equation 12.7 is that we can find the recognition result by finding the largest projections along the eigenvectors.

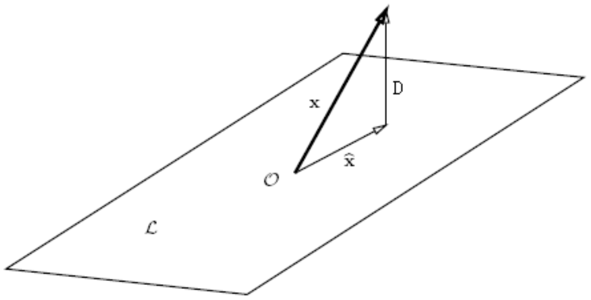


Figure 12.1: Projection into Subspace

12.3 Implementation of a Subspace Recognizer

From the algorithm described in the above section, our subspace prototype recognizer contains two steps: building the subspace and recognition.

12.3.1 How to Build the Subspace

Similar to the cases of HMM and elastic matching recognizers, we preprocessed the handwriting symbols. The preprocessed symbols were sent to the feature extractor module. The extracted feature vectors are used to build the correlation matrix. The correlation matrix was sent to the PCA (Principal Component Analysis) module, where we calculated eigenvectors. Then the subspace for each class in the training database was built from the eigenvectors. The diagram is shown in Figure 12.2.

12.3.2 Recognition

In the recognition step, the unknown symbol goes through the preprocessing and feature extractor modules. Next the feature vector is projected on each subspace calculated from the *Build Subspace* step. After each projection is calculated, we find the maximum projection and label the corresponding class as the recognition result. This is shown in Figure 12.3.

We used x, y coordinates as the features in the recognizer and selected 20 as the number of principle components.

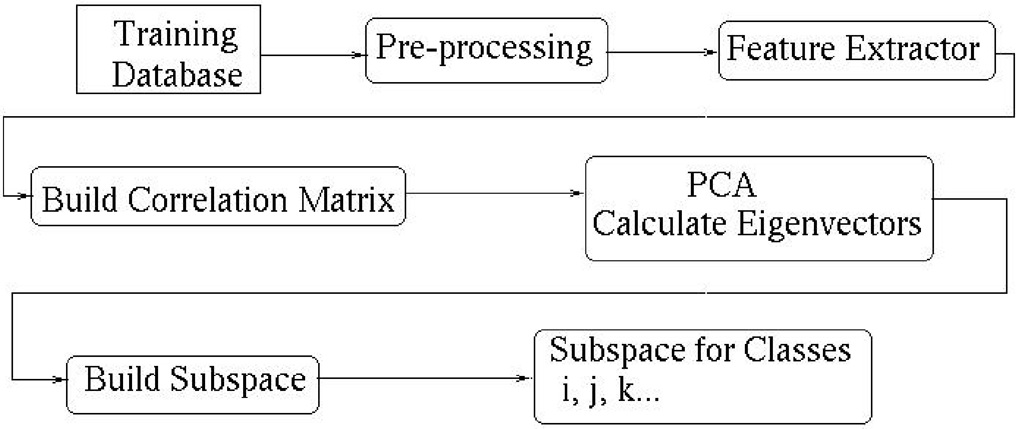


Figure 12.2: Build Subspace

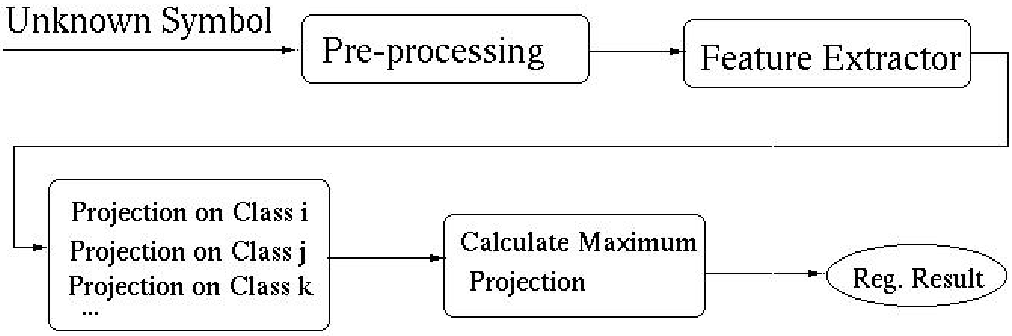


Figure 12.3: Subspace Recognition

12.4 Discussion

We tested the prototype recognizer on the ORCCA database. The overall recognition rate is about 60% over all of the symbols. We believe this is due to the following:

The subspace classification method models a class using a subspace. The linear combination of the basis vectors represents the variation of the class. This limits the variations in the pattern. Given the fact that handwritten mathematical symbols have substantial variations, we need to work on the limitations in order to use subspace method.

To further investigate the subspace method, we may be able to overcome its limitations by:

- finding a proper subspace for each class, *e.g.* increasing the dimension of the space.
- finding other distance measure rules, since the orthogonal distance measure does not consider any distortion within the subspace [15].
- investigating other subspace classification methods.

However, our first results indicate that the non-subspace methods will initially be more useful for mathematical symbol recognition.

Chapter 13

Combining Recognizers and Applying Context Rules

The software we have developed for character recognition has been used in a number of experiments in the ORCCA laboratory. One of these is an experiment in using context information to guide recognition and another is in the combination recognizers. Smirnova and Watt [54] have explored these ideas and have shown how they may be used to improve recognition results. We summarize that work here to show an application of our recognizer.

13.1 Combination of Recognizers

The combination of multiple classifiers has become a subject of attention as recent results show that it can improve recognition [30]. However the combination of different recognizers requires more computation. We have chosen not to combine our elastic matching recognizer and HMM recognizer for the following reasons:

- In a combined recognizer, the recognition process needs to be performed by two recognizers, increasing the recognition time. In addition to this, computations are required for the combination scheme.
- The HMM and elastic matching recognizers already have reasonable performance.

13.2 Individual Classifier Generation

Recently, a number of procedures called *ensemble methods* were proposed in the field of handwriting recognition [22]. Ensemble methods have become popular and are often embedded in the algorithms of individual handwriting recognizers [30]. Given a base recognizer, an ensemble of different recognizers can be generated by changing the training set, the input features or the parameters and architecture of the base recognizer.

The classic ensemble methods include the Bagging, AdaBoost, random subspace and architecture variation methods, as described in the paper by Bunke [22], which we summarize here.

Bagging means bootstrapping and aggregating. Given a training set S of size n, bagging generates m new training sets $S_1, \ldots, S_m$, each of size n, by randomly drawing elements of the original training set. Each of the new sets S_i is used to train exactly one recognizer. In this way, a set of m individual recognizers are assembled from a trading set S.

AdaBoost was developed in 1995 and is one of the most prominent algorithms [22]. Similarly to Bagging, the original training set is also modified for the creation of the ensemble in AdaBoost algorithm. A selection probability is assigned to each element in the training set. AdaBoost creates a new training set by randomly selecting elements from the original training set but taking the selection probabilities into account. Like Bagging, the new training set is used to train one recognizer. The new recognizer is tested on the original training set. The selection probabilities will be modified if the new recognizer gives different recognition results from the original recognizer. One need to repeat the process of creating new training sets and modifying selection probability until the desired number of recognizers are created. Unlike Bagging, where the recognizers are created independently, the recognizers generated by AdaBoost are created dependent on the selection probabilities, which means it depends on the recognition results of previous recognizers.

In the random subspace method, a subset of all features is selected for training and recognition. Given the size of subset, the features are randomly chosen from the set of all features. The subset of features is used to create a new recognizer. Each new recognizer is trained on the entire training set. If the feature set is small, modifying the algorithm by setting equal selection probabilities can improve the performance [22].

As the name indicates, in the architecture variation ensemble method, the structure or the parameters of the architecture of the recognizer is varied. For example, in the HMM-based recognizers, the following parameters can be modified to generate new recognizers:

1. the number of states
2. the type of HMM
3. the number of training iterations

Other ensemble methods exist presently in the literature. Some of them are derived from the classic ones. For example, Günter used simple probabilistic boosting method, effort based boosting method, *etc* [22].

13.3 Combination Schemes

A main issue in the combination of recognizers is the choice of combination schemes. The selection of schemes depends on the output of the recognizers. For example, if the outputs of the recognizers are ranked lists of the classes, then the Borda count may be applied. If the outputs are scores of each class, then we can apply certain operations on the scores such as summation, multiplication, maximum, *etc.* We describe some of the well-known combination schemes in the following paragraphs. The notations and formulars are adopted from [22].

Bayesian Combination Rule: This method requires the calculation of probability of a pattern belonging to a class i when the recognizer outputs class j. Thus, probabilities for all pairs of classes and all recognizers need to be calculated, which is not computationally feasible in the cases of large vocabularies.

Score Combination Scheme: requires the calculation of the maximum, minimum, average or the median of the scores for each class over all recognizers. Since the scores for all classes need to be computed, this scheme is not applicable for the ranked list output. However the maximum score combination scheme is an exception. The class that has the maximum score for each recognizer is the first one in the rank list. Therefore maximum score is equivalent to (from [22]):

$$\mathrm{maxscore}(\mathrm{C_1, C_2, \ldots, C_n})(\mathrm{x}) = \mathrm{class}(\mathrm{arg}(\max_{\mathrm{c_i} \in \{\mathrm{c_1, \ldots, c_n}\}}(\mathrm{score}(\mathrm{C_i}, 1)(\mathrm{x}))), 1)$$

where $\mathrm{score(C, i)(x)}$ denotes the score of the ith best class output by the recognizer C for the pattern x. And $\mathrm{class(C, i)(x)}$ is the ith best class output by the recognizer C for the pattern x. These notations apply to the voting schemes as well.

Voting: Only the best class output by each recognizer is considered. The class that appears most often in the outputs of the recognizers is the output of the combined recognizer. The voting method can be expressed as (from [22]):

$$\mathrm{vote(C_1, C_2, \ldots, C_n)(x) = arg(\max_{c_i \in \{c_1, \ldots, c_m\}} (C_j | class(C_j, 1)(x) = c_i))}$$

arg in the above equation means the variables in left hand side comes from the related arguments of the right hand side.

To break ties, the following approaches may be used:

- All patterns where a tie occurs are rejected by the combined classifier.
- One of the tied classes is randomly selected as output of the combined classifier.
- Apply another combination scheme on the tied classes.

The voting scheme is applicable for the ranked list output. It is easy to compute.

Weighted Voting: A weight w_i is assigned to each recognizer C_i. The class that has the highest sum of weights is the output of the combined recognizer. The following equation shows the definition of weighted voting (from [22]):

$$\mathrm{wvote(C_1, C_2, \ldots, C_n)(x) = arg(\max_{c_i \in \{c_1, \ldots, c_m\}} (\sum_{\{j | class(C_j, 1)(x) = c_i\}} w_j))}$$

The weight for each recognizer can be set up according to the performance of the recognizer or to certain optimization rules. These two weight selection approaches are called performance weights and optimized weights respectively. In the performance weights, the recognition rate of each recognizer is used as the weight: the better the recognition performance, the higher the weight. In the case of optimized weights, they are assigned to each recognizer so that the optimal performance of the combined recognizer is achieved. Genetic algorithm is used to optimize the weights.

Borda Count: The k-best list of classes are considered in the Borda count method. The output of the combined recognizer is computed according to the ranked list of each individual recognizer. The Borda count is defined by the following equation (from [22]):

$$\mathrm{bcount}(\mathrm{C_1}, \mathrm{C_2}, \ldots, \mathrm{C_n})(\mathrm{x}) = \arg(\max_{\mathrm{c_i} \in \{\mathrm{c_1}, \ldots, \mathrm{c_m}\}} \sum_{\mathrm{j}=1}^{\mathrm{n}} (\mathrm{k} - \mathrm{rank}(\mathrm{C_j}, \mathrm{c_i})(\mathrm{x}) + 1))$$

Borda count does not consider the recognition performance of each recognizer. If we take the recognition ability of each recognizer into account, a more complex combination is defined as (from [22]):

$$\mathrm{rcombi}(\mathrm{C_1}, \mathrm{C_2}, \ldots, \mathrm{C_n}) = \arg(\max_{\mathrm{c_i} \in \{\mathrm{c_1}, \ldots, \mathrm{c_m}\}} \sum_{\mathrm{j}=1}^{\mathrm{n}} (\mathrm{a_j} + \mathrm{w_j}(\mathrm{rank}(\mathrm{C_j}, \mathrm{c_i})(\mathrm{x}))))$$

A weight function $w_j(rank)$ is assigned to each recognizer C_j, in addition, a weight a_j is assigned to each recognizer. The values for $w_j(rank)$ and a_j may be set by optimization algorithm or manually.

13.4 Combining Dictionary-Based Prediction with Recognition

Due to similarity and variability of handwriting characters, ambiguity always exists and impedes handwriting recognition. This is illustrated in Figures 13.1 and 13.2 (from [70]). Notice that the circled symbols are exactly the same. We can apply context information to resolve ambiguity and improve the recognition rate.

Dictionary-based methods are widely used in natural language handwriting recognition. Figure 13.3 shows an example where context information applies in natural language recognition. Even though the first and last strokes are exactly same, it is recognized as *cloud* instead of *doud* based on the dictionary. Smirnova and Watt extended the techniques to mathematical character recognition [54]. They built a "mathematical vocabulary" from expressions in a large collection of research articles and used this to compute sequence frequencies. Based on their context database, the equation in the Figure 13.1 is recognized as "$\dot{z} + z = \sin \omega t$", while for the Figure 13.2, the recognition result would be "$\sum_i i^2$". We present a brief description of their work here. More detailed information can be found in reference [54].

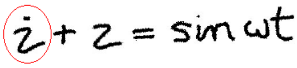


Figure 13.1: Context - Ambiguity in Mathematics(1)

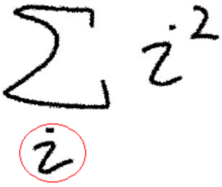


Figure 13.2: Context - Ambiguity in Mathematics(2)

13.4.1 Building Database of Mathematical Context

So and Watt [59] analyzed and categorized 20,000 mathematical documents published between 2000 and 2004 by the mathematical arXiv service. Based on that analysis, Smirnova and Watt [54] studied mathematical sub-expressions and identified the "most common mathematical patterns" after extracting all possible symbol sequences of lengths 3, 4 and 5. These expressions were recorded in presentation MathML, which contains spatial information. For example, $(a + b)^2$ can be expressed in MathML as: `<msup> <mfenced> <mi> a </mi> <mo> + </mo> <mi> b </mi> </mfenced> <mn> 2 </mn> </msup>`. A "trie" data structure was used to store these subsequence. A node of the trie stores the total frequency of the symbol it represents. Figure 13.4 shows a partial trie for symbol "i". The size of the trie can be up to several gigabytes. By examining the tries for sub-expressions of length 3, 4 and 5, it was found that the distribution for the number of expressions at each frequency appeared to be reciprocal to the frequency of an expression: there was a significant increase in the number of expressions with low frequencies and *vice-versa*. For example, there are about 2000 out of $456,751$ subexpressions for which the frequency in the mathematical articles is 91, while only 10 out of $456,751$ subexpressions with frequency 1302 appear in those articles. Based on these findings, the trie was cut off by storing only M expressions with frequencies more than F. Three different sizes of database were created: 1. a database with all sequences that were encountered more than 400 times; 2. a database containing all expressions that occurred more than 100 times; 3. a database including all expressions with frequencies more than 1000 times. The second one continued to be used for their research and experiments.

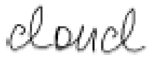


Figure 13.3: Context - Ambiguity in Natural Language, from [54]

13.4.2 Predicting Characters from the Mathematical Context

Based on the mathematical context, one can provide the user with the next possible symbol that fits into the formula that was written so far. The Markov chain property applies here, *i.e.* we only consider the current state to predict the next symbol. The state consists of the sequence of previous $n-1$ symbols. For example, after a user writes $\sum$ in Figure 13.2, if we detect that the pen is in the subscript area of $\sum$, we can offer the prediction "i". The probabilities of subscript of $\sum$ are shown in the Table 13.1. The Figure 13.4 and Table 13.1 are from [54]. The prediction is done by searching the database of mathematical context and normalizing the probabilities of the possible symbols. Three alternatives are used to normalize the probability of the possible symbol, for example, for symbol "x": 1) frequency of "x"/ frequency of tree root; 2) frequency of "x"/ frequency of immediate root; 3) frequency of "x"/max frequency of siblings of "x".

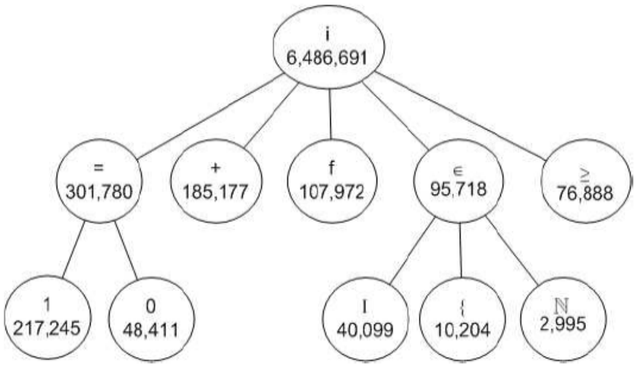


Figure 13.4: Trie for Symbol "i", from [54]

Symbol	Probability	Symbol	Probability	Symbol	Probability
i	1	j	0.59426	k	0.502789
n	0.59426	m	0.063205	l	0.110686
r	0.082793	p	0.079962	α	0.079631

Table 13.1: Subscripts of $\sum$, from [54]

13.4.3 Combining Prediction and Recognition

Smirnova and Watt examined the following three combination functions:

$$\begin{aligned}
\mathcal{C}_1(\alpha,\beta) &= \alpha\mathcal{R}+\beta \\
\mathcal{C}_2(\alpha,\beta) &= \mathcal{R}^{\alpha}\mathcal{P}^{\beta} \\
\mathcal{C}_3(\alpha,\beta) &= \alpha\times exp(\mathcal{R})+\beta\times exp(\mathcal{P})
\end{aligned}$$

Experiments were done on a set of 17 formulas written by 10 different writers.

By a proper choice of α and β, recognition rate can be increased. For example, for $\mathcal{C}_3$, given $\alpha = 6, \beta = 1$ the recognition rate can be increased by up to 7.5%.

In this chapter, we gave an overview of combining multiple recognizers. Various ensemble scheme and combination methods are studied. We also summarized the work of Smirnova and Watt on applying context rules for handwritten mathematical recognition.

Chapter 14

Conclusion and Future Work

14.1 Conclusion

We have studied different aspects of recognition of handwritten mathematical symbols and have presented new methods to model them. We have also implemented these models in a handwriting recognition prototype with two recognizers: an elastic matching-based recognizer and a hidden Markov model-based recognizer. We have used C++ on Linux to implement the recognizers. We have found these methods to be effective in recognizing handwritten mathematical symbols.

At a more detailed level, we have done the following:

- A recognition prototype was built to include data preprocessing, analysis, symbol representation and recognition. Handwriting recognition relies upon a large number of samples for each symbol, involving the assembly of raw database which usually contains corrupt data (for instance, missing stroke). We built tools for examination and verification of raw data. Since our database comprises data from a variety of resources, we produced converters to transform the data among different formats. After the initial examination and conversions are completed, we have started from preprocessing, followed by variance analysis, feature extraction, and build training and recognition modules.
- We performed handwriting variance analysis and identified the factors contributed to the variants and could be used as the most-distinguishably features.
- Based on the variance analysis, we prepared modules for all allomorphs of the mathematical symbols to be used in elastic matching. The hidden Markov model also used the allomorphs for training purposes.

- We examined different possible features of handwritten mathematical symbols. We identified new features to represent the handwritten mathematical symbols. Feature analysis and extraction are used in both the elastic recognizer and the HMM recognizer.
- By grouping the large set of symbols according to different features, we pruned the prototypes for elastic matching and the recognition speed was improved.
- Two new encoding/decomposing schemes for handwritten symbols were developed. The first encoding method is curved-based, where the number of encoded segments is flexible and depends on the symbol under consideration. The second encoding method has fixed number of segments of equal length. These encoding were used in the design of HMM.
- We designed a multi-path, multi-model HMM topology. The hidden Markov model for each character had multiple paths according to its variants. Except for using information based on time series, we integrated inter-stroke information into the HMM. We built individual models based on different features. These models were combined to form a final model with an improved recognition rate compared the traditional, flat HMM.
- We associated the decomposed segments with their corresponding states. From the mapping, we chose the observation distribution to be of a Gaussian type which has enabled us to achieve promising recognition results.

14.2 Future Work

Ambiguity is always present in handwriting recognition due to similarities and variability of handwriting characters. This can be alleviated by the application of context information. Others have examined frequencies of the symbols appearing in mathematical expressions and built Markov chains for prediction. Integration of this work with ours gives the promise of an improved recognition rate.

The combination of multiple recognizers is another research direction to be explored further. The following directions appear particularly interesting:

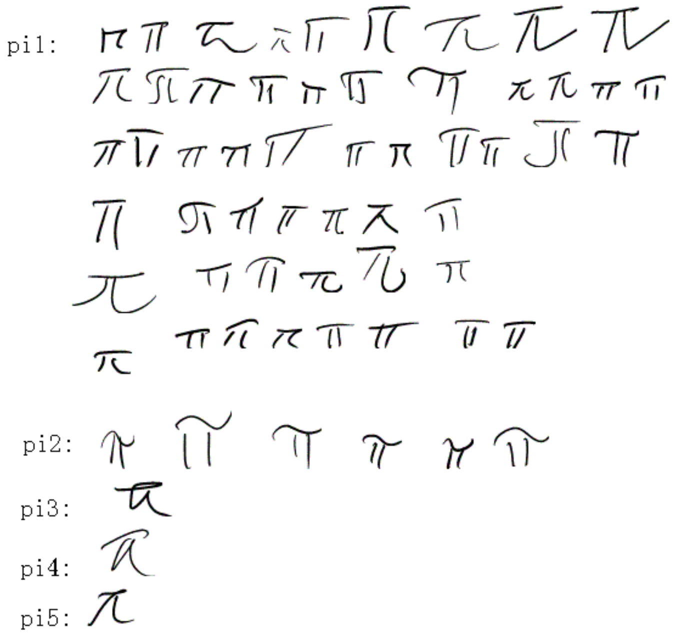
- Generation of an writer-specific Recognizer. The write-specific recognizer should be suitable for combination and should be easily generated.
- Combination Schemes. While a number of combination schemes appear in the literature, more effective combination schemes need to be developed.

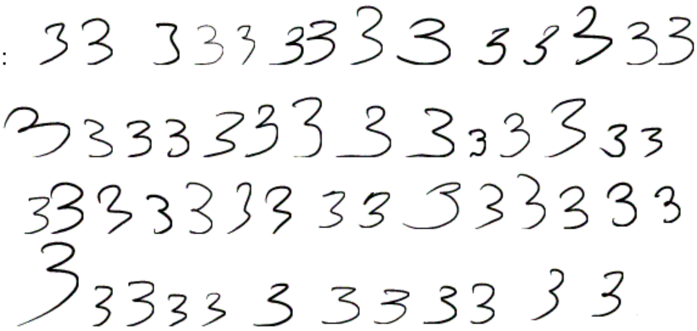
Our handwritten mathematical symbol recognizers have been embedded in a handwritten mathematical expression recognition system. The present work can be further extended to editing and recognition of handwritten mathematical expression.

Appendix A

Samples of Allomorph of Handwritten Symbols

In Chapter 6, we introduced mathematical handwriting variant analysis. Based on the variant analysis, we built a database of allomorphs for each character. Samples of allomorphs are shown in this appendix. These allomorphs were used in building hidden Markov model.



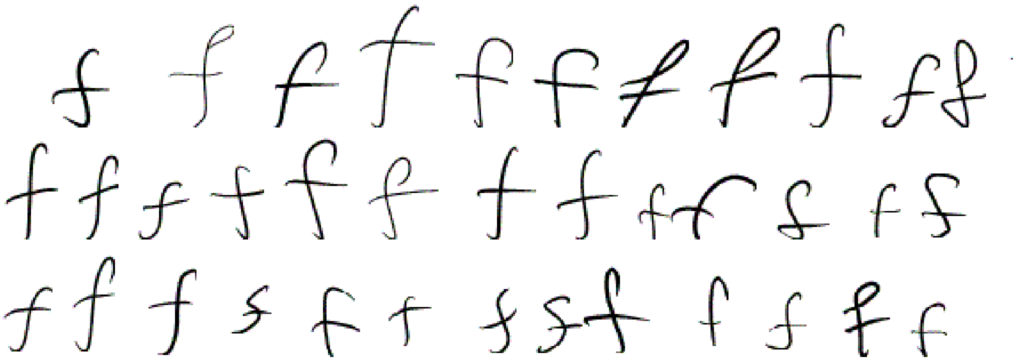
three1: 

three2:

three3:

three4:

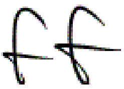
three5: f1:



f2:

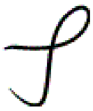
f3:

f4:

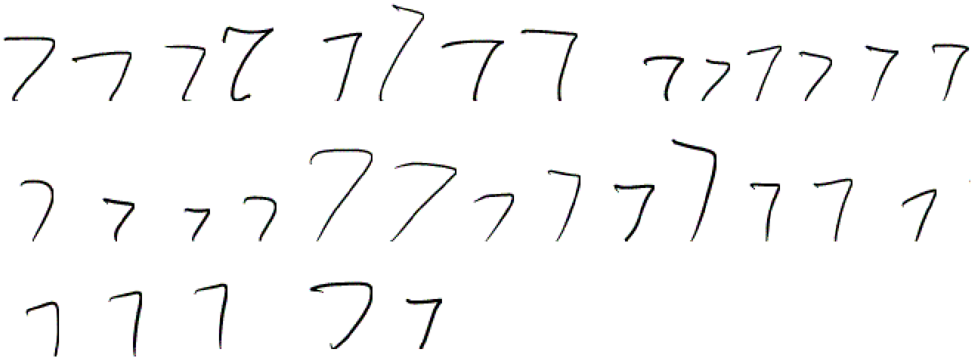


f5:

f6:

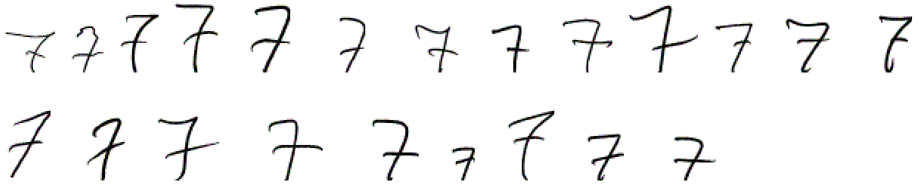


seven1:

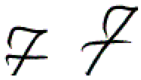


seven2:

seven3:

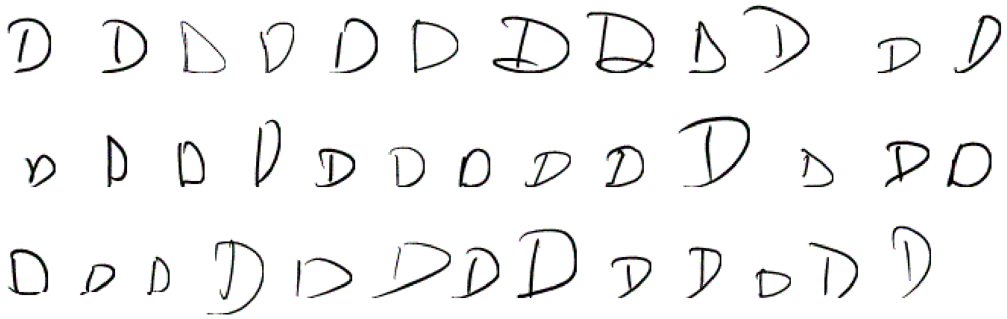


seven4:

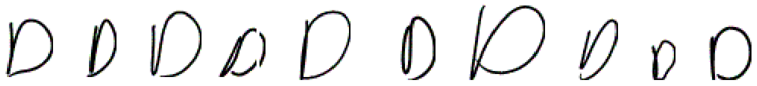


seven5:

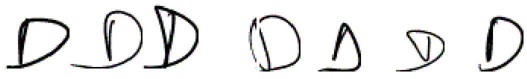
D1:



D2:

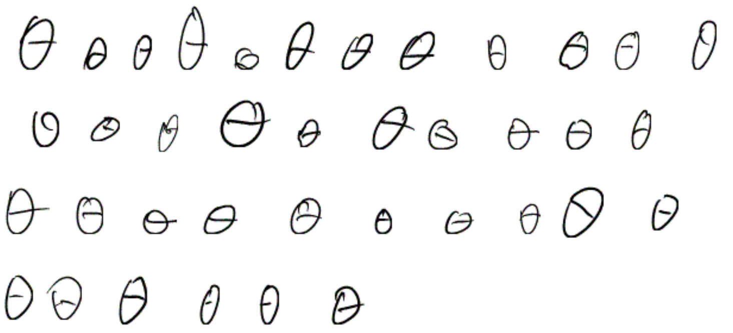


D3:

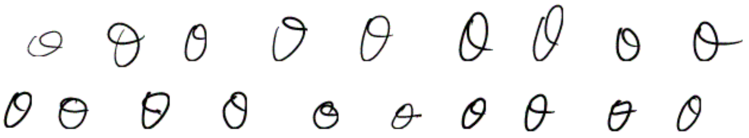


D4:

theta1:



theta2:



Appendix B

Dynamic Programming for Elastic Matching

```
// Below is the code for Elastic Matching Dynamic programming.
// Refer to Equation 8.1 in Chapter 8.
// Input parameter distanceMatrix’s index is 1 based.
void CRecognizer_Elastic::iniDisMatrix(CMatrix &distanceMatrix,
    vector<CPoint> &unknownpoints, vector<CPoint> &modelpoints,
    vector<double> &unknownangles, vector<double> &modelangles)
{
  int nrow=distanceMatrix.NRow();
  int ncol=distanceMatrix.NCol();
  //initilize the matrix
  for(int i=0; i<nrow; i++)
  {
    for(int j=0;j<ncol; j++)
    {
      distanceMatrix.SetAt(i+1,j+1,-1.0);
    }
  }
  double inidis=Util::pointDistance(unknownpoints, modelpoints,0,0,
        unknownangles,modelangles);
  distanceMatrix.SetAt(1,1,inidis);

  //if "j = 0" in Equation 8.1 in Chapter 8
```

```
for(int j=1; j<ncol; j++)
{
  double predis=distanceMatrix.GetAt(1,j);
  double pointdis=Util::pointDistance(unknownpoints, modelpoints,0,
          j,unknownangles, modelangles);
  distanceMatrix.SetAt(1,j+1,predis+pointdis);
}

//if "i = 0" in Equation 8.2 in Chapter 8
for(int i=1;i<nrow; i++)
{
  double predis=distanceMatrix.GetAt(i,1);
  double pointdis=Util::pointDistance(unknownpoints, modelpoints,i,
          0,unknownangles, modelangles);
  distanceMatrix.SetAt(i+1,1,predis+pointdis);
}

// if "i >0, j = 1" and if "i > 0, j > 1" in Equation 8.1
for(int i=2;i<=nrow;i++)
{
  for(int j=2;j<=ncol;j++)
  {
    //if "i > 0, j =1" in Equation 8.1 in Chapter 8
    if(j==2)
    {
      double min=(distanceMatrix.GetAt(i-1,j)  <
         distanceMatrix.GetAt(i-1,j-1) ? distanceMatrix.GetAt(i-1,j):
         distanceMatrix.GetAt(i-1,j-1));
      double pointdis=Util::pointDistance(unknownpoints, modelpoints,
         i-1,j-1,unknownangles,modelangles);
    distanceMatrix.SetAt(i,j,pointdis+min);
   }
   else //if "i > 0, j > 1" in Equation 8.1 in Chapter 8
   {
    double totaldis1=distanceMatrix.GetAt(i-1,j);
    double totaldis2=distanceMatrix.GetAt(i-1,j-1);
    double totaldis3=distanceMatrix.GetAt(i-1,j-2);

    if( totaldis1 == -1 || totaldis2 == -1 ||totaldis3 == -1)
    {
     throw Exception("totalDistance, uninitialized value!");
    }

    double min1= (totaldis1 <=totaldis2 ? totaldis1 : totaldis2);
    double min =(min1 < totaldis3 ? min1 : totaldis3);

    double pointdis=Util::pointDistance(unknownpoints,
     modelpoints,i-1,j-1,unknownangles,modelangles);
    distanceMatrix.SetAt(i,j,pointdis+min);
   }
  } //end of for j = 2 loop
 }//end of for i = 2 loop
}

//related utilities functions
namespace Util
{
  double pointDistance(std::vector<CPoint> &unknownpoints,
   std::vector<CPoint> &modelpoints, int i, int j,
   std::vector<double> &unknownangles,
      std::vector<double> &modelangles)
    {
        double pointdis=unknownpoints[i].distance(modelpoints[j]);
        double angledis=angleDistance(unknownangles[i],modelangles[j]);
        return (pointdis+angledis);
    } //end of pointDistance

    double angleDistance(double angle1, double angle2)
    {
        double delta=fabs(angle1-angle2);
        delta=(delta <= 2*M_PI-delta ? delta : 2*M_PI-delta);
        return ELASTIC_ANGLE_CONST*delta;
    }
}
```

REFERENCES

[1] "Vector quantization," http://www.data-compression.com/vq.shtmla, Valid on: Nov 9, 2007.

[2] H. A and S. N., "Cursive script recognition using wildcards and multiple experts," in *Pattern Analysis and Applications*, vol. 4, no. 1. Springer, 2001, pp. 51–60.

[3] H. Abut, *Vector Quantization.* Institute of Electrical & Electronics Engineer Press, 1990.

[4] R. Anderson, "Syntax-directed recognition of hand-printed two-dimensional mathematics," in *Interactive Systems for Experimental Applied Mathematics*, 1968, pp. 436–459.

[5] L. Bahl and F. Jelinek, "Decoding for channels with insertions, deletions, and substitutions with applications to speech recognition," in *IEEE Transactions on Information Theory*, vol. 21. Institute of Electrical and Electronics Engineers, 1975, pp. 404–411.

[6] J. Baker, "The dragon system - an overview," in *IEEE Transactions on Acoustics Speech Signal Processing*, vol. 23, no. 1. Institute of Electrical and Electronics Engineers, 1975, pp. 24–29.

[7] L. Baum and T. Petrie, "Statistical inference for probabilistic functions of finite state Markov chains," in *Ann. Math. Stat.*, vol. 37, 1966, pp. 1554–1563.

[8] K. Chan and D. Yeung, "Elastic structural matching for on-line handwritten alphanumeric character recognition," Department of Computer Science, Hong Kong University of Science and Technology, Tech. Rep. 98-07, March 1998.

[9] K. Chan and D. Yeung, "Recognizing on-line handwritten alphanumeric characters through flexible structural matching," in *Pattern Recognition*, vol. 32. Elsevier Science, 1999, pp. 1099–1114.

[10] K. Chan and D. Yeung, "Mathematical expression recognition: A survey," in *International Journal on Document Analysis and Recognition*, vol. 3, no. 1. Springer, Cagliaris, Italy, August 2000, pp. 3–15.

[11] B. W. Char and S. M. Watt, “Representing and characterizing handwritten mathematical symbols through succinct functional approximation,” in *Internatiional Conference on Document Analysis and Recognition.* IEEE Computer Society, Sept 2007, pp. 1198–1202.

[12] B. Chazelle and H. Edelsbrunner, “An optimal algorithm for intersecting line segments in the plane,” in *ACM*, vol. 39, no. 1, Jan 1992, pp. 1–54.

[13] E. Choi, “On compensating the mel-frequency cepstral coefficients for noisy speech recognition,” in *Proceedings of Twenty-Ninth Australasian Computer Science Conference.* Australasian Computer Science Communications, 2006, pp. 49–54.

[14] T. H. Corman, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms.* MIT Press and McGraw-Hill, 1989.

[15] V. Deepu, S. Madhvanath, and A. Ramakrishnan, “Principal component analysis for online handwritten character recognition,” in *Proceedings of the 17th International Conference on Pattern Recognition.* IEEE Computer Society, 2004, pp. 327–320.

[16] K. Durdle, “Supporting mathematical handwriting recognition through an extended digital ink framwork,” Master’s thesis, Department of Computer Science, the University of Western Ontario, 2004.

[17] S. Eickeler, A. Kosmala, and G. Rigoll, “Hidden Markov model based continuous online gesture recognition,” in *International Conference on Pattern Recognition.* IEEE Computer Society, August 1998, pp. 1206–1208.

[18] W. Freiseisen and P. Pau, “A generic plane-sweep for intersecting line segments,” RISC Report Series, Univeristy of Linz, Austria, Tech. Rep. 98-18, Nov 1998.

[19] M. Fujimoto, T. Kanahori, and M. Suzuki, “Infty editor - a mathematics typesetting tool with a handwriting interface and a graphical front-end to openXM servers,” in *Computer Algebra - Algorithms, Implementations and Applications.* RIMS Kokyuroku, 2003, pp. 217–226.

[20] A. Grbavec and D. Blostein, “Mathematics recognition using graph rewriting,” in *International Conference on Document Analysis and Recognition.* IEEE Computer Society, Aug 1995, pp. 417–421.

[21] D. Guillevic and C. Y. Suen, “HMM word recognition engine.” in *International Conference on Document Analysis and Recognition.* IEEE Computer Society, 1997, pp. 544–547.

[22] S. Günter and H. Bunke, "Ensembles of classifiers for handwritten word recognition," in *International Journal of Document Analysis and Recognition*, vol. 5, no. 4. Springer, Cagliaris, Italy, 2003, pp. 224–232.

[23] I. Guyon, L. Schomaker, R. Plamondon, M. Liberman, and S. Janet, "Unipen project of on-line data exchange and recognizer benchmarks," in *Proceedings of the 12th International Conference on Pattern Recognition.* IEEE Computer Society, 1994, pp. 29–33.

[24] J. Ha, R. Haralick, and I. Phillips, "Understanding mathematical expressions from document images," in *International Conference on Document Analysis and Recognition.* IEEE Computer Society, 1995, pp. 956–959.

[25] S. Impedovo, "Frontiers in handwriting recognition," in *Fundamentals in Handwriting Recognition.* Springer-Verlag, New York, USA, 1994.

[26] N. Joshi, G. Sita, and A. Ramakrishnan, "Comparison of elastic matching algorithms for online tamil handwritten character recognition," in 9*th International Workshop on Frontiers in Handwriting Recognition.* IEEE Computer Society, Oct 2004, pp. 444–449.

[27] J. O. Jr, J. M. de Carvalho, C. de A.Freitas, and R.Sabourin, "Feature sets evaluation for handwritten word recognition," in 8^{th} *International Workshop on Frontiers in Handwriting Recognition.* IEEE Computer Society, August 2002, pp. 446–551.

[28] B. Keshari and S. M. Watt, "Hybrid mathematical symbol recognition using support vector machines," in *Internatiional Conference on Document Analysis and Recognition.* IEEE Computer Society, Sept 2007, pp. 859–863.

[29] B. Keshari and S. M. Watt, "Streaming-archival InkML conversion," in *Internatiional Conference on Document Analysis and Recognition.* IEEE Computer Society, Sept 2007, pp. 1183–1187.

[30] J. Kittler and F. Roli, *Multiple Classifier Systems.* Springer, Cagliaris, Italy, 2000.

[31] A. Koerich, R. Sabourin, and C. Suen, "Large vocabulary off-line handwriting recognition: A survey," in *Pattern Analysis and Applications*, vol. 6, no. 2. Springer, 2003, pp. 97–121.

[32] A. L. Koerich, R. Sabourin, and C. Y. Suen, "Lexicon-driven HMM decoding for large vocabulary handwriting recognition with multiple character models," in *International Journal of Document Analysis and Recognition*, vol. 6, no. 2. Springer, Cagliaris, Italy, 2003, pp. 126–144.

[33] M. Koschinski, H.-J. Winkler, and M. Lang, "Segmentation and recognition of symbols within handwritten mathematical expressions," in *International Conference on Acoustics, Speech and Signal Processing*, vol. 4. IEEE Press, May 1995, pp. 2439–2442.

[34] A. Kosmala, G. Rigoll, S. Lavirotte, and L. Pottier, "On-line handwritten formula recognition using hidden Markov models and context dependent graph grammars," in *International Conference on Document Analysis and Recognition.* IEEE Computer Society, Sep 1999, pp. 107–110.

[35] J. M. Kurtzberg, "Feature analysis for symbol recognition by elastic matching," in *IBM Journal of Research and Development.* IBM Corp, January 1987, pp. 91–95.

[36] J. Laaksonen, "Subspace classifiers in recognition of handwritten digits," Ph.D. dissertation, Helsinki University of Technology, 1997.

[37] J. J. Lee, J. Kim, and J. H. Kim, "Data driven design of HMM topology for on-line handwriting recognition," in 7^{th} *International Workshop on Frontiers in Handwriting Recognition.* IEEE Computer Society, September 2000, pp. 239–248.

[38] G. Leedham, W. K. Tan, and W. L. Yap, "Handwritten country name identification using vector quantisation and hidden Markov model," in *Proceedings of the Sixth International Conference on Document Analysis and Recognition.* IEEE Computer Society, Sep 2001, pp. 685–688.

[39] Y. Linde, A. Buzo, and R. Gray, "An algorithm for vector quantizer design," in *IEEE Transactions on Communications.* Institute of Electrical and Electronics Engineers, 1980, pp. 702–710.

[40] X. Liu and M. Blumenstein, "Experimental analysis of the modified direction feature for cursive character recognition," in 9*th International Workshop on Frontiers in Handwriting Recognition.* IEEE Computer Society, Oct 2004, pp. 353–358.

[41] A. Louie, "Bachelor's thesis: Handwriting analysis for MathML generation," Department of Applied Mathematics, the University of Western Ontario, 2000.

[42] J. Makhoul, T. Starner, R. Schwartz, and G. Chou, "On-line cursive handwriting recognition using hidden Markov models and statistical grammars," in *HLT '94: Proceedings of the workshop on Human Language Technology*, 1994, pp. 432–436.

[43] S. Marukatat and T. Artières, "Handling spatial information in on-line handwriting recognition," in 9*th International Workshop on Frontiers in Handwriting Recognition.* IEEE Computer Society, Oct 2004, pp. 14–19.

[44] N. E. Matsakis, “Recognition of handwritten mathematical expressions,” Master’s thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1999.

[45] B. Murthy, “Handwriting recognition using supervised neural networks,” in *International Joint Conference on Neural Networks*, vol. 4. IEEE Press, 1999, pp. 2899–2902.

[46] K. Nathan, J. Bellegarda, D. Nahamoo, and E. Bellegarda, “On-line handwriting recognition using continuous parameter hidden Markov models,” in *International Conference on Acoustics Speech and Signal Processing.* IEEE Press, April 1993, pp. 121–124.

[47] M. Okamoto and K. Yamamoto, “On-line handwritten character recognition method using directional features and clockwise/counterclockwise direction-change features,” in *International Conference on Document Analysis and Recognition.* IEEE Computer Society, Sep 1999, pp. 491–494.

[48] M. Okamoto, S. Sakaguchi, and T. Suzuki, “Segmentation of touching characters in formulas,” in *Document Analysis System.* Springer-Verlag, 1998, pp. 151–156.

[49] L. R. Rabiner, “A tutorial on hidden Markov model and selected applications in speech recognition,” in *Proceedings of the IEEE*, vol. 77-2, 1989, pp. 257–286.

[50] G. Rigoll and A. Kosmala, “A systematic comparison between on-line and off-line methods for signature verification with hidden Markov models,” in *International Conference on Pattern Recognition.* IEEE Computer Society, August 1998, pp. 1755–1757.

[51] H. Shu, “On-line handwriting recognition using hidden Markov models,” Master’s thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1996.

[52] E. Smirnova and S. M. Watt, “A context for pen-based mathematical computing,” in *Proceedings of Maple Conference*, July 2005, pp. 409–422.

[53] E. Smirnova and S. M. Watt, “Communicating mathematics via pen-based computer interfaces,” in *Communicating Mathematics in Digital Era*, 2006.

[54] E. Smirnova and S. M. Watt, “Mining expirical data to improve pen-based mathematical character recognition,” in *Communicating Mathematics in Digital Era*, Agust 2006.

[55] E. Smirnova and S. M. Watt, “Aspects of mathematical expression analysis in arabic handwriting,” in *Internatiional Conference on Document Analysis and Recognition.* IEEE Computer Society, Sept 2007, pp. 1183–1187.

[56] E. Smirnova and S. M. Watt, “A cross-application architecture for pen-based mathematical interfaces,” in *Electronic Proc. Mathematical User Interfaces*, June 2007.

[57] S. Smithies, K. Novins, and J. Arvo, “A context for pen-based mathematical computing,” in *Proceedings of Graphics Interface.* Morgan Kaufmann Publishers, June 1999, pp. 84–91.

[58] C. So, “An analysis of mathematical expressions used in practice,” Master’s thesis, Department of Computer Science, the University of Western Ontario, 2005.

[59] C. M. So and S. M. Watt, “Determining expirical properties of mathematical expression use,” in *Fourth International Conference on Mathematical Knowledge Management.* Springer, July 2005, pp. 361–375.

[60] S. Srihari, S. Cha, H. Arora, and S. Lee, “Individuality of handwriting,” in *Journal of Forensic Sciences*, vol. 47, no. 4. ASTM International, July 2002, pp. 1–17.

[61] E. Tapia, “Understanding mathematics: A system for the recognition of on-line handwritten mathematical expressions,” Ph.D. dissertation, the University of Fu Berlin, 2004.

[62] C. Tappert, C. Suen, and T. Wakahara, “On-line handwritten recognition - a survey,” in *International Conference on Pattern Recognition.* IEEE Computer Society, Nov 1988, pp. 1123–1132.

[63] C. Tappert, C. Suen, and T. Wakahara, “The state of the art in online handwriting recognition,” in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 8. IEEE Computer Society, Aug 1990, pp. 787–808.

[64] J. Tokuno, N. Inami, S. Matsuda, M. Nakai, H. Shimodaira, and S. Sagayama, “Context-dependent substroke model for HMM-based on-line handwriting recognitions,” in 8^{th} *International Workshop on Frontiers in Handwriting Recognition.* IEEE Computer Society, Aug 2002, pp. 78–83.

[65] O. Trier, A. Jain, and T. Taxt, “Feature extraction methods for character recognition - a survey,” in *Pattern Recognition*, vol. 29, no. 4. Elsevier Science, 1996, pp. 641–662.

[66] B. Wan, “An interactive mathematical handwriting recognizer for the Pocket PC,” Master’s thesis, Department of Computer Science, the University of Western Ontario, 2002.

[67] W. Wang, A. Brakensiek, A. Kosmala, and G. Rigoll, “Multi-branch and two-pass HMM modeling approaches for off-line cursive handwriting recognition,” in *Proceedings*

of the Sixth International Conference on Document Analysis and Recognition. IEEE Computer Society, Sep 2001, pp. 231–235.

[68] Z. Wang and C. Faure, "Structural analysis of handwritten mathematical expressions," in *International Conference on Pattern Recognition.* IEEE Computer Society, 1988, pp. 32–43.

[69] J. R. Ward and T. Kuklinski, "A model for variability effects in handprint with implications for the design of handwriting character recognition systems," in *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 18, no. 3. Institute of Electrical and Electronics Engineers, May 1988, pp. 438–450.

[70] S. M. Watt, "Strategies for pen-based mathematics," in *Applications of Computer Algebra*, 2004.

[71] S. M. Watt, "New aspects of InkML for pen-based computing," in *Internatiional Conference on Document Analysis and Recognition.* IEEE Computer Society, Sept 2007, pp. 457–460.

[72] H.-J. Winkler, "HMM-based handwritten symbol recognition using on-line and off-line features," in *International Conference on Acoustics Speech and Signal Processing.* IEEE Press, May 1996, pp. 3438–3441.

[73] X. Wu, "Achieving interoperability of pen computing among heterogeneous devices and digital ink formats," Master's thesis, Department of Computer Science, the University of Western Ontario, 2004.

[74] Z. Xuejun, L. Xinyu, Z. Shengling, P. Baochang, and Y. Y.Tang, "On-line recognition handwritten mathematical symbols," in *4th International Conference Document Analysis and Recognition.* IEEE Computer Society, August 1997, pp. 645–5648.

[75] R. Yamamoto, S. Sako, T. Nishimoto, and S. Sagayama, "On-line recognition of handwritten mathematical expressions based on stroke-based stochastic context-free grammar," in *International Workshop on Frontiers in Handwriting Recognition.* IEEE Computer Society, Oct 2006, pp. 249–254.

[76] R. Zanibbi, D. Blostein, and J. R. Cordy, "Recognizing mathematical expressions using tree transformation," in *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 11, 2002, pp. 1455–1467.

[77] P. H. Zein, "Chinese calligraphy," http://www.zein.se/patrik/chinen9p.html, Valid on: Nov 9, 2007.

[78] L. Zhang and R. Fateman, "Survey of user input models for mathematical recognition: Keyboards, mice, tablets,voice," University of California, 2003.

VITA

NAME:	Xiaofang Xie
PLACE OF BIRTH:	ShanXi, China
YEAR OF BIRTH:	1974
POST-SECONDARY EDUCATION AND DEGREES:	The University of Western Ontario London, Ontario 1999-2001 M.Sc.
	Nanjing University of Science and Technology Nanjing, P.R.China 1992-1996 B.E.
HONORS AND AWARDS:	Ontario Graduate Scholarship 2002-2003
	Ontario Graduate Scholarship in Science and Technology 2003-2004
	The University of Western Ontario International Graduate Student Scholarship Special University Scholarship 1999-2001
RELATED WORK EXPERIENCE:	Teaching Assistant and Research Assistant University of Western Ontario London, Ontario 1999-2004

PUBLICATIONS:

Refereed Conference Papers:

Recognition for Large Sets of Handwritten Mathematical Symbols, Stephen M. Watt and Xiaofang Xie, pp. 740-744, Proc. IEEE International Conference on Document Analysis and Recognition, August, 2005, Seoul Korea.

Prototype Pruning by Feature Extraction in Handwritten Mathematical Symbol Recognition, Stephen M. Watt and Xiaofang Xie, pp. 423-437, Proc. Maple Conference, July, 2005, Waterloo, Canada, Maplesoft.

Refereed Abstracts:

Components for Pen-Based Mathematical Interfaces, Elena Smirnova, Clare So, Stephen M. Watt and Xiaofang Xie, Proc. 2005 Conference on the Applications of Computer Algebra, July, 2005, Nara, Japan.

Refereed Posters:

Handwritten Mathematical Symbol Recognition for Computer Algebra Applications, Stephen M. Watt and Xiaofang Xie, 2005 East Cost Computer Algebra Day, March, 2005, Ashland, OH, USA.

An Experimental Handwritten Mathematical Symbol Recognition System, Stephen M. Watt and Xiaofang Xie, 11th Annual East Coast Computer Algebra Day, March, 2004, Waterloo, Canada.