Online Handwritten Word Recognition for Indic Scripts using Hidden Markov Models and Data-driven Modeling of Writing Styles

THESIS

Submitted in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

Bharath A.

Under the supervision of

Dr. Sriganesh Madhvanath

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE

PILANI (RAJASTHAN) INDIA

2009

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
PILANI (RAJASTHAN)

CERTIFICATE

This is to certify that the thesis entitled Online Handwritten Word Recognition for Indic Scripts using Hidden Markov Models and Data-driven Modeling of Writing Styles and submitted by Bharath A. ID No 2005PHXF040P for award of Ph.D. Degree of the Institute embodies original work done by him/her under my supervision.

Signature in full of the Supervisor: ___________________________

Name in block letters: Dr. SRIGANESH MADHVANATH

Designation: Senior Research Scientist, Hewlett-Packard Labs India

Date:

To my beloved parents

Acknowledgments

I am immensely thankful to Prof. L. K. Maheshwari, Vice-Chancellor, BITS, Pilani for providing me this opportunity to pursue the off-campus PhD of the Institute. I express my gratitude to Prof. Ravi Prakash, Dean, Research and Consultancy Division (RCD), BITS, Pilani for his constant official support, encouragement and making the organization of my research work through the past few years easy.

I thank Dr. Hemanth Jadav, Mr. Dinesh Kumar, Ms. Monica Sharma, Mr. Sharad Shrivastava, Mr. Gunjan Soni, Mr. Amit Kumar and Ms. Sunita Bansal, nucleus members of RCD, BITS, Pilani, without their cooperation and guidance it would not have been possible for me to pursue such goal oriented research during each of the past few semesters. I also express my gratitude to the office staff of RCD whose secretarial assistance helped me in submitting the various evaluation documents in time and give pre-submission seminar smoothly.

I thank my Doctoral Advisory Committee (DAC) members, Dr. Rajkumar Gupta and Dr. Mukesh Kumar Rohil, who spared their valuable time to go through my draft thesis and were audience to my pre-submission seminar in order to provide several valuable suggestions that immensely helped in improving the quality of my PhD thesis report.

It has been a stimulating experience to purse my PhD research at HP Labs India under the BITS - HP Labs PhD Fellowship Programme. I would like to thank all my present and former colleagues at HP Labs India for supporting me during this work.

First of all, I am deeply grateful to my supervisor, Dr. Sriganesh Madhvanath, for his guidance, support and encouragement during my PhD. It has been an excellent learning experience working under him. His expertise in the field has played a pivotal role in this research.

I would also like to thank Dr. Kuchibhotla Anjaneyulu for offering me this fellowship and providing an opportunity to work in an environment of high standards.

Special thanks are due to Dinesh Mandalapu for the numerous insightful discussions on handwriting recognition and pattern recognition.

I would like to extend my appreciation to former colleagues - Deepu Vijayasenan for introducing me to the world of handwriting recognition during my Masters' internship, Dr. Kalika Bali for the guidance on data collection and defining the symbol sets, Jagannadhan V. for organizing and accompanying me during Tamil data collection, and Neemala Sridhar Krishna for the discussions on Hidden Markov Models.

I am thankful to the interns Jagadeesh, Prashanth, Praveen, Kalyan and Vibhas for their assistance in creating the Devanagari dataset. I am also grateful to the members of the Lipi Toolkit development team and intern Saravanan for their support. Without their help it would have taken much longer time to complete this thesis. I also thank Muthu, Balu, Tarun and Vandana for making this experience a fun-filled and memorable one.

Finally, I would like to thank my parents for their contribution in my PhD. I am extremely grateful for their love, prayers and untiring support, without which I could have never accomplished this endeavor.

Bharath A.

Abstract

Online handwriting recognition (OHWR) refers to the problem of machine recognition of handwriting captured in the form of pen trajectories, via a digitizing tablet and stylus. Over the years, a number of algorithms have been proposed for recognizing online handwriting, and today there are several commercial systems for recognizing European and Oriental scripts. OHWR technology holds significant promise for the Indic family of scripts, given that the Indic languages are used by a sixth of the world's population, and the greater ease of use of handwriting-based text input compared to keyboard-based methods for these scripts. The structure of the scripts and the variety of shapes and writing styles pose some unique challenges that make adoption of existing word recognition systems developed for English or other languages, difficult. While there has been considerable research on recognition of isolated symbols and characters in Indic scripts, research for recognizing larger writing units such as words or phrases is in its early stages. The systems developed to date have assumed various constraints on writing, or have been script-specific, or both.

In this thesis, we address the problem of online handwritten word recognition for Indic scripts. In contrast to prior approaches, we propose techniques that are *script-independent* and *data-driven*, involving minimal manual intervention during training, and validate our approach for two important Indic scripts - Devanagari and Tamil. Our approach employs Hidden Markov Models (HMM) to model strokes, symbols and words in the script. The HMMs are trained using a large number of word samples collected from over a hundred writers for each script, cleaned and annotated at the symbol level.

From the perspective of recognition, this thesis addresses two central issues that arise in the context of online Indic scripts: (i) variations in writing style of individual symbols, (ii) symbol order variations within and across characters. Variations in writing style at the symbol level result from variations in the shape, position, number, ordering, or direction of the constituent strokes. In this work, the writing styles of a symbol are automatically discovered from the training data and a *stroke-based modeling* approach is adopted to represent them. A novel technique based on *constrained stroke clustering* that exploits constraints in the handwriting domain is proposed to identify the optimal set of strokes and styles in the dataset. The results obtained for Devanagari recognition demonstrate substantial improvement in accuracy when compared to alternate techniques such as unsupervised clustering of stroke samples.

For word recognition, we explore lexicon-driven and lexicon-free approaches based on HMMs. The lexicon-driven approach works well when the symbol order is standard or known in advance. In order to deal with the second central issue - that of symbol order variations, we propose a lexicon-free approach that uses a novel *Bag-of-Symbols* (BoS) representation of words. When compared to the lexicon-driven scheme for word recognition, our lexicon-free approach performs significantly better for samples having symbol orders different from the standard order. In the case of Devanagari, our investigation reveals that a simple combination of lexicon-driven and lexicon-free recognizers results in considerably higher accuracy than using either of them alone. Maximum accuracies obtained for 20,000 word lexicons are 87.13% for Devanagari when the lexicon-driven and lexicon-free recognizers are combined, and 91.8% for Tamil using the lexicon-driven approach.

Over the last decade, there has been tremendous growth in mobile devices in the Indian subcontinent. Many of these devices now offer natural interaction using touchscreens where one can simply use a finger, instead of the stylus, to tap or write on the surface. Given that one of the main applications of OHWR is text input, it is important that the developed recognition technology caters to these newer devices supporting touch-based interaction. In the last part of the thesis, we propose a novel Input Method Editor (IME) named FreePad, that allows one to "overwrite" a complete word on a small touch surface by ignoring the standard left to right writing order and the relative positions of the strokes. A different preprocessing step that operates at the stroke level is used along with the word recognition schemes developed earlier, to recognize such *position-free handwriting*. In this context, we also explore the relevance of position information for recognizing handwriting in Devanagari and Tamil, which like other Indic scripts, is two-dimensional in nature. Our experiments show that it is indeed possible to recognize handwriting written as discrete symbols reliably, even in the absence of any position information. Maximum accuracies obtained for 20,000 word lexicons are 88.78% for Devanagari and 93.18% for Tamil when the lexicon-driven and lexicon-free recognizers are combined.

The script-independent and data-driven approach developed in this thesis for online handwritten word recognition for Indic scripts is promising. Given that there is no prior work on Devanagari or Tamil online word recognition, we hope that the recognition performances reported in this work will serve as a benchmark for future efforts.

Contents

List of Tables

List of Figures

Abbreviations

OHWR	- Online Handwriting Recognition
HMM	- Hidden Markov Model
DTW	- Dynamic Time Warping
NN	- Nearest Neighbor
BoS	- Bag of Symbols
IME	- Input Method Editor
CJK	- Chinese, Japanese and Korean
OTS	- Optimal Text Selection
ML	- Must-Link
CL	- Cannot-Link
AHC	- Agglomerative Hierarchical Clustering
CCL	- Constrained Complete Link

Chapter 1

Introduction

1.1 Background

Handwriting recognition refers to the problem of machine recognition of handwritten script. It is broadly classified as online and offline recognition. In the case of Online Handwriting Recognition (OHWR), special devices such as pen-based tablets are used to capture handwriting in the form of digital ink. Digital ink normally contains the pen trajectory represented as X-Y coordinates along with the time information. In the offline scenario, handwriting is captured in the form of digitized images using a scanner or a camera. Online data embeds additional information such as writing order, temporal information (velocity, pressure etc.) and pen lift events. However the same dynamic information proves to be a burden at times, since it captures variations in stroke number and order which do not change the identity of what is written. A stroke refers to the ink in between a pen-down and the successive pen-up event. The major advantages of OHWR are the possibilities for user interactivity and adaptation [3].

OHWR is commonly used to enable handwriting input for general-purpose applications on fixed and mobile devices such as Table PCs and mobile phones. It is also used for specific applications in education, manufacturing and so on. Decades of research have made handwriting input a reality for several European and Oriental languages represented respectively using the Latin and the Chinese, Japanese, Korean (CJK) scripts [4, 5, 6, 7]. Handwriting input methods are now commonly built into mobile devices and Tablet PCs and feature acceptably high recognition accuracies. However similar technology for the Indic languages and scripts is still in its infancy.

1.2 Motivation

India plays host to 22 official languages and 10 scripts, in addition to a large number of others which do not have 'official' status. The official languages invariably have large numbers of speakers (e.g. approximately 500 million speakers of Hindi, 200 million of Bangla [8]). Many Indic languages have substantial global presence – Tamil for instance is also one of the official languages in countries such as Singapore, Malaysia, and Sri Lanka. The users of these languages and scripts constitute approximately a sixth of the world's population.

In India, Information Technology (IT) is still largely limited to the small fraction of the population that is English-literate. One reason for this has been the complexity of text input in local Indic languages. Over the years a number of QWERTY-overlays and specific keyboard layouts have been devised for different Indic languages, but they remain non-standard and difficult to learn and use. The large alphabet size typically requires multiple keystrokes for entering a character and mandates complex key-character mappings to be remembered, presenting a substantial barrier to use, especially for occasional users. The use of handwriting, on the other hand, is widely entrenched in the home, government and business, and forms the basis for record-keeping and communication.

In this setting, technology for OHWR in Indic languages and scripts can play a significant role in promoting IT in local languages. As compared to speech, the processing of handwriting input for text entry is less expensive computationally on mobile platforms, and potentially more accurate, especially in the presence of ambient noise.

The structure of the scripts and the variety of shapes and writing styles pose some unique challenges that make adoption of existing word recognition systems developed for English or other languages, difficult. While there has been considerable research on recognition of isolated symbols and characters in Indic scripts, research for recognizing larger writing units such as words or phrases is in its early stages. The systems developed to date have assumed various constraints on writing, or have been script-specific, or both.

1.3 Problem Definition

The input to a typical OHWR system is in the most general case, a digital ink document or stream that contains several lines of handwritten text, with each line comprised of a set of words separated by spaces. Once lines in the document are separated out and each line is in turn segmented into words, the problem of *online handwritten word recognition* is defined as recognizing a given isolated word, i.e. finding the optimal sequence of characters from the script given the corresponding digital ink.

The thesis addresses the problem of online handwritten word recognition in the context of Indic scripts. The problem has two parts:

1. To propose an *approach* for online word recognition for Indic scripts with the following desired characteristics:
   - **Script-independent**: The approach should not be strongly coupled to the nature of shapes present in a particular script, but should be generalizable to all Indic scripts.
   - **Automatic and data-driven**: Minimal manual intervention should be required during training of the system.
   - **Support for writer-independent and unconstrained handwriting recognition**: The approach should work for a large class of users and should not impose any constraints on the writing style that makes the system unusable in practice.
2. To propose a *handwriting-based text input solution* for entering Indic scripts on small, touch-based mobile devices based on the proposed approach.

While the first part is concerned with developing recognition techniques that are most appropriate for natural handwriting in Indic scripts, the second part looks at an important practical application scenario wherein conventional writing may not be possible.

1.4 The Structure of Indic Scripts

The 10 official Indic scripts - Devanagari, Tamil, Gurmukhi, Telugu, Kannada, Gujarati, Oriya, Bangla, Malayalam and Urdu - differ by varying degrees in their visual characteristics, but share some important similarities. With the exception of the Urdu script which is of Perso-Arabic origin, they have evolved from a single source, the phonographic Brahmi script, first documented extensively in the edicts of Emperor Asoka of the third century BC. They are defined as “syllabic alphabets” or *abugidas* in that the unit of encoding is a syllable of speech, however the corresponding orthographic units show distinctive internal structure and a constituent set of graphemes or symbols [9, 10]. A word in these scripts is written as a sequence of these orthographic syllabic units. For simplicity, we will henceforth refer to these units as “characters”, a term commonly used to refer to the building blocks of recognition systems (as in “isolated *character* recognition”). Some common classes of characters are described below with examples:

- V: An independent vowel */ii/*[1]

  Devanagari: ई Tamil: ஈ Telugu: ఈ

- C: An isolated consonant */ta/* with inherent neutral vowel */a/*

  Devanagari: त Tamil: த Telugu: త

- CH: An isolated consonant */t/* with the inherent vowel muted by a special diacritic called *halanth*. This form is used rarely; the linear form (CH)(C) of two consecutive consonant sounds is instead generally represented by a single consonant conjunct character (C’C), where C’ denotes the half-form of the first consonant. A notable exception is Tamil, wherein the use of the CH form is the norm.

  Devanagari: त् Tamil: த் Telugu: త్

- CV: A consonant */ta/* combined with vowel */ii/* to produce */tii/*. The vowel overrides the inherent */a/* and is indicated using a *matra* or diacritic symbol (also see Fig. 1.1).

  Devanagari: ती Tamil: தீ Telugu: తీ

- CVM: A CV character with a modifier M to indicate nasalization of the vowel V. Devanagari has four different kinds of nasalization, indicated using different kinds of diacritic

[1]In this thesis, we have used ITRANS Romanisation scheme (http://www.aczoom.com/itrans/tblall/tblall.html) for indicating the sounds of Indic symbols and characters

marks, e.g. तीं */tii.n/*

- C'C: A conjunct that combines two consonant sounds - */k/* with */ta/* to produce */kta/*. In Devanagari, the leading consonant */k/* is indicated by a 'half form' to suggest that it is missing its inherent vowel. However in Telugu it is just the reverse - the leading consonant */ka/* is shown in its full form but */ta/*, indicated by the horizontal diacritic at the bottom, is shown in its half form. As mentioned earlier, Tamil does not use half forms. Instead, consonant conjuncts are "unraveled" into a linear sequence of (CH) characters followed by the base consonant C.

  Devanagari: क्त Tamil: க்த Telugu: క్త

- CC: A distinct conjunct wherein the constituent consonants are not identifiable e.g. consonant श् */sh/* combines with consonant र */ra/* to produce a distinct shape श्र */shra/* in Devanagari. Many C'C conjuncts have alternative representations as distinct conjuncts.

ITRANS Encoding														
ka	kaa	ki	kii	ku	kuu	ke	kE	kai	ko	kO	kau	k.n	kaH	k
Devanagari														
क	का	कि	की	कु	कू	के	-	कै	को	-	कौ	कं	कः	क्
Telugu														
క	కా	కి	కీ	కు	కూ	కె	కే	కై	కొ	కో	కౌ	కం	కః	క్
Tamil														
க	கா	கி	கீ	கு	கூ	கெ	கே	கை	கொ	கோ	கௌ	-	-	க்

Figure 1.1: Combinations of the consonant */ka/* with different vowels and the vowel muting *halanth*

The above classes represent subsets of the set of all possible characters. In its most complex form a character is composed of a *base consonant* C, surrounded by modifiers for other consonants C', a vowel unit V and modifier M. For example, क्तीं */ktii.n/* in Devangari has the structure C'CVM. This example also shows an alternative representation of */kta/* as a distinct conjunct. The first two classes of characters C and V are sometimes collectively called "simple characters" and have been the focus of early efforts at recognizing Indic scripts. Most Indic scripts have an order of 600 CV characters and as many as 20,000 C'CV ones in theory, although a much smaller subset of C'CV characters are used in practice. While Devanagari has 35 consonants, 11 vowels and 4 vowel modifiers, Telugu has 18 vowels and 36 consonants. The total numbers of characters in both the scripts run into the thousands (around 1500 for Devanagari and 5000 for Telugu), though smaller numbers are encountered in common usage. Tamil has 12 vowels, 18 consonants, 6 Grantha letters (that can combine with vowels for writing Sanskrit words) and a special symbol, with the number of characters summing up to 325 [11]. The much smaller number compared to Devanagari and Telugu is largely due to consonant conjunct characters such as C'C being written as linear sequences of separate characters CH and C, as previously mentioned.

1.5 Challenges in Online Indic Script Recognition

The structure of characters in Indic scripts was introduced in the previous section. In this section we describe some of the challenges for Online HWR of characters and words in Indic scripts, and highlight key differences from Latin and CJK scripts. Some of these stem from the structure of these scripts; others from established writing practice.

1.5.1 Large Alphabet Size

As mentioned earlier, most Indic scripts use a large (> 1000) number of characters, as opposed to less than 100 in English. The internal graphemic structure makes a divide and conquer approach to recognition feasible in theory. However many consonant conjuncts are represented by visually distinct conjuncts bearing no resemblance to the constituent consonant shapes (Fig. 1.2). Similarly, many consonant and vowel combinations give rise to new symbols (e.g. 'து' */thu/* in Tamil) which cannot be segmented into the base consonant and matra. These may need to be dealt with as opaque symbols and exceptions in a divide and conquer recognition strategy.

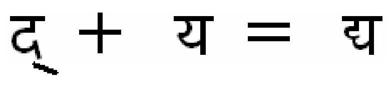


Figure 1.2: Combination of consonants resulting in a distinct conjunct character in Devanagari

1.5.2 Two-dimensional Structure

As is evident from Fig. 1.1, *matras* or vowel diacritics can occur to the left, right, bottom, top, or even as multiple components surrounding the base consonant. Some possible vowel matras for a consonant symbol in Devanagari are shown in Fig. 1.3(a). In Fig. 1.3(b), a two-part matra with components occurring on the left as well as on the right of a base consonant in Tamil is shown. Similarly, half-consonant forms in consonant conjuncts can occur in different positions around the base consonant. In the case of Telugu, they are aligned vertically below the consonant as shown in Fig. 1.3(c). Position is also important to distinguish certain matras and consonant forms that have very similar shapes and differ only in their position relative to the base consonant e.g. half-consonant */n/* and matras */e/* and */uu/* shown in Fig. 1.4. Thus Indic scripts exhibit a two-dimensional structure much like the CJK scripts. Modeling Indic characters for recognition in terms of the constituent strokes or graphemes requires modeling of their spatial relationships in addition to their shapes. Further, the two dimensional structure results in added variability in symbol and stroke order across writers, unlike the linear left-to-right ordering of the Latin script.

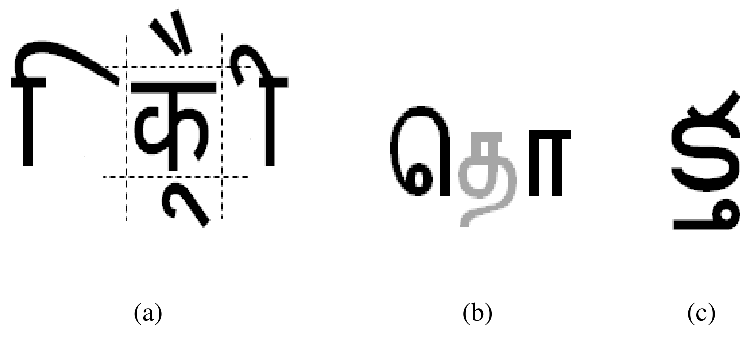


Figure 1.3: Two-dimensional structure: (a) some possible matras for a consonant in Devanagari (b) two-part matra surrounding the consonant in Tamil (c) consonant conjunct in Telugu

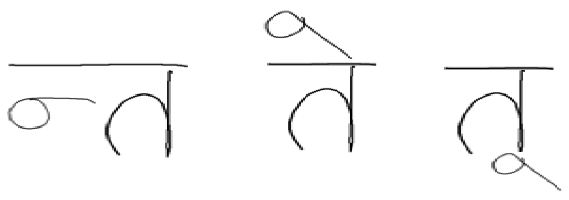


Figure 1.4: Similar-looking half-consonant and matras varying only in their position relative to the base consonant */ta/*

1.5.3 Inter-class Similarity

In certain Indic scripts, there is intrinsically high inter-class similarity between some pairs of symbols. Fig. 1.5(a) shows two characters from Malayalam that look very similar except for the small loop present in the first. Fig. 1.5(b) shows two Tamil characters with a subtle difference in the shapes of their matras. This calls for reliable, highly distinctive features to describe the shapes of characters and graphemes.

(a) (b)

Figure 1.5: Similar-looking characters in (a) Malayalam (b) Tamil

1.5.4 Issues with Writing Styles

Indic scripts with the exception of Urdu are written as a left-to-right sequence of characters (syllabic units). As already discussed, the characters themselves have two-dimensional arrangements of graphemes corresponding to consonants, vowels and vowel modifiers. Since characters can vary widely in width, height and complexity, there is no boxed style. This is a key difference from CJK scripts wherein characters are complex but of approximately the same size and may be written in boxes.

In general, character transitions are marked by pen-lifts. Writing an entire word cursively is possible in some Indic scripts (e.g. Bangla), but rare. However cursiveness is common within characters, and found generally wherever a pen-up requires additional effort.

While writing a character, users are generally concerned with reconstructing its visual appearance rather than its phonological structure. Various factors such as the relative positions of different strokes, the effort required to move from one stroke to the next given the overall flow of writing, and writing styles taught in school - all have an influence on the stroke order that is eventually used. The consequences for Online HWR are many:

Different character forms

An Indic character could be written in different forms that are identical phonetically but significantly different in their visual appearance. As a result, a given character with a single UNICODE value could have more than one representation in terms of the constituent symbols. Fig. 1.6 shows two different ways of writing the same conjunct character */nna/* in Devanagari (UNICODE value 0928-094d-0928). It is important to take into account such alternate forms when lexicon entries encoded in UNICODE are used for recognition.

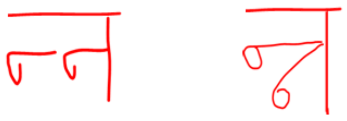


Figure 1.6: Two different ways of writing the Devanagari character */nna/*

Symbol Order Variations

The sequence of writing of consonant and vowel units in a character need not correspond to the phonological order of their occurrence in the corresponding syllable. For instance, the 'ெ' matra in Tamil is often written before writing the base consonant, since it occurs to its left. In contrast, the UNICODE representation of CV characters in Indic scripts is based on their phonological structure, and always encodes the consonant before the vowel. Fig. 1.7 shows an example from Devanagari where a character contains two symbols (marked in different colors). While one might expect the base consonant (shown in black) to be written first, the order of writing is likely to vary across writers.

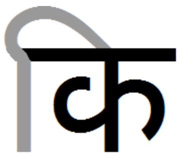


Figure 1.7: Symbol order variations within a character

Stroke Order Variations Spanning Multiple Symbols

Strokes from different graphemes may be interleaved while writing a character. For example, a two-stroke matra may be written partially and completed only after the base consonant is written. This is somewhat related to the phenomenon of delayed strokes in English, wherein some strokes are entered only after the completion of the entire word. However for Indic scripts this happens at the level of individual characters, and the variations are widespread and not limited to a small number of strokes such as t-crossings and i-dots in English. Further, because of the 2-D nature of the scripts, it is often not possible to use heuristics to reorder out-of-order strokes as is common practice for dealing with delayed strokes in English.

Stroke Number, Order and Direction Variations within Symbols

In general, stroke order, number and direction variations are quite high in Indic characters and constitute one of the central challenges in online recognition of Indic scripts. For example, Fig. 1.8 shows different styles of writing the consonant */pha/* in Devanagari, where each style (shown along the columns) differs in the number and/or the order of strokes.

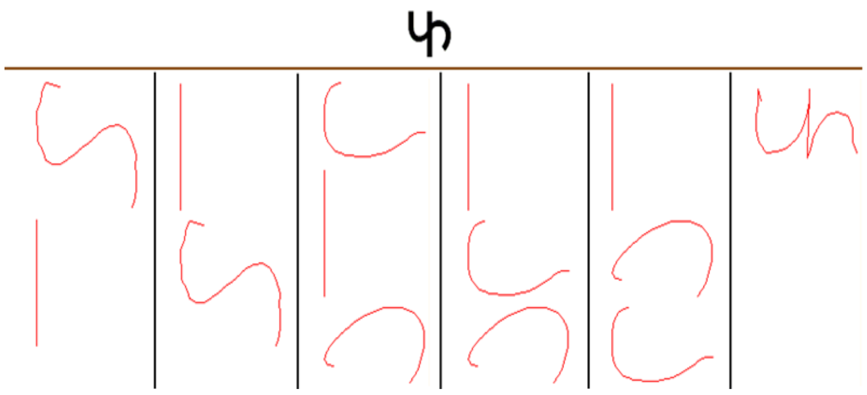


Figure 1.8: Writing styles identified for a Devanagari character

1.5.5 Language-specific and Regional Differences in Usage

Marked differences in the use of symbols may be observed in the use of a script like Devanagari across languages such as Hindi, Sanskrit, Marathi and Nepali. For instance, the *halanth* (vowel muting diacritic) and the CH form are used frequently in Sanskrit, but rarely in Hindi. The shapes of symbols may also show regional variations, influenced by other languages and scripts in use in the region and its surrounding areas. Due to the fact that languages such as Tamil and Bangla span multiple countries, one may also expect country-specific differences in the use of the corresponding scripts. In all these cases, models of symbol shape, as well as any language models, used by the handwriting recognition system to improve accuracy need to be appropriately customized for the specific language or region.

1.5.6 Comparison with Latin and CJK Scripts

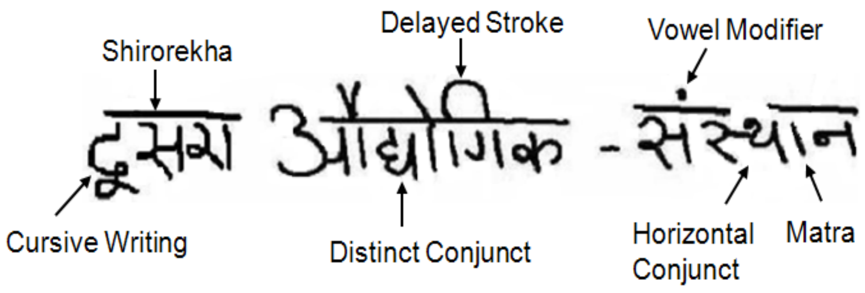


Figure 1.9: Some challenges for Online HWR of Devanagari script

The challenges for online recognition of Indic scripts are sufficiently different from those for Latin. Chief among them as already mentioned are the large number of classes, the stroke order and number variations within and across characters, and the two dimensional nature of the script. There are several others. For example, small vowel modifiers may get interpreted as noise in the input, and the *shirorekha* or headline which is often written after completing the word requires special treatment (Fig. 1.9).

Indic script recognition also differs from that of CJK scripts in a few significant ways. In the case of CJK scripts, the shape of each stroke in a character is generally a straight line and hence stroke direction based features are often sufficient. But in the case of Indic scripts, the basic strokes are often nonlinear or curved, and hence features that provide more information than just the directional properties are required. Moreover, in CJK scripts, a word is generally written discretely and hence segmenting it into characters is much easier when compared to Indic scripts where the most common style of writing is run-on. Table 1.1 summarizes the salient characteristics of Indic, Latin and CJK scripts from the perspective of Online HWR.

1.6 Outline of Thesis

In this thesis, we address the problem of online handwritten word recognition for Indic scripts. In contrast to prior approaches, we propose techniques that are *script-independent* and *data-driven*, involving minimal manual intervention during training, and validate our approach for two important Indic scripts - Devanagari and Tamil. Devanagari, being the script used for

Table 1.1: Comparison of Indic, Latin and CJK scripts

Property	**Indic**	**Latin**	**CJK**
writing system	syllabic alphabet	alphabetic	Chinese and Kanji: pictographic-ideographic, Kana: syllabary, Hangul: syllabic alphabet
number of units	> 1500	< 100	Chinese and Kanji: few thousands, Kana: 48, Hangul: few thousands
common style of writing a word	run-on	cursive	discrete
structure of writing	2-D	1-D	2-D
stroke shape complexity	high	high	low
stroke order/number variation	high	low	high

languages such as Sanskrit, Hindi (500 million speakers), Nepali (30 million speakers), Marathi (70 million speakers) and Konkani, is by far the most significant Indic script. Tamil, on the other hand, has 80 million native speakers and substantial global presence, being a national language of countries such as Singapore, Malaysia, and Sri Lanka. Our approach uses HMMs to model strokes, symbols and words in the script. HMMs are suitable for handwriting recognition for a number of reasons. Since these are stochastic models, they can cope with noise and variations in the handwriting. The observation sequence that corresponds to features of an input word can be of variable length, and most importantly, word HMMs can solve the problem of segmentation implicitly.

When handwritten symbols are modeled using HMMs, an important problem to address is modeling variations in writing style of individual symbols. Variations in writing style at the symbol level result from variations in the shape, position, number, ordering, or direction of the constituent strokes. In this work, the writing styles of a symbol are automatically discovered from the training data and a *stroke-based modeling* approach is adopted to represent them. A novel technique based on *constrained stroke clustering* that exploits constraints in the handwriting domain is proposed to identify the optimal set of strokes and styles in the dataset.

For word recognition, we explore lexicon-driven and lexicon-free approaches based on HMMs. The lexicon-driven approach represents the words in the lexicon as a prefix-tree assuming a standard symbol order. In order to deal with symbol order variations, we propose a lexicon-free approach that uses a novel *Bag-of-Symbols* representation of words.

In this thesis, we also propose a novel Input Method Editor named FreePad, that allows one to "overwrite" a complete word on a small touch surface by ignoring the standard left to right writing order and the relative positions of the strokes. For recognition of such *position-free handwriting*, we apply the symbol modeling and word recognition techniques developed earlier for normal handwriting, but with a different preprocessing step that operates at the stroke level.

The high-level architecture of our word recognition system is shown below in Fig. 1.10. The input word is first preprocessed to address writer-specific variations in size and speed of the handwriting. Then features are extracted at each point on the trajectory. During training, different writing styles of each symbol are automatically identified, and symbol models that share strokes are built. These symbol models are used along with the lexicon for recognition of a feature-extracted word sample.

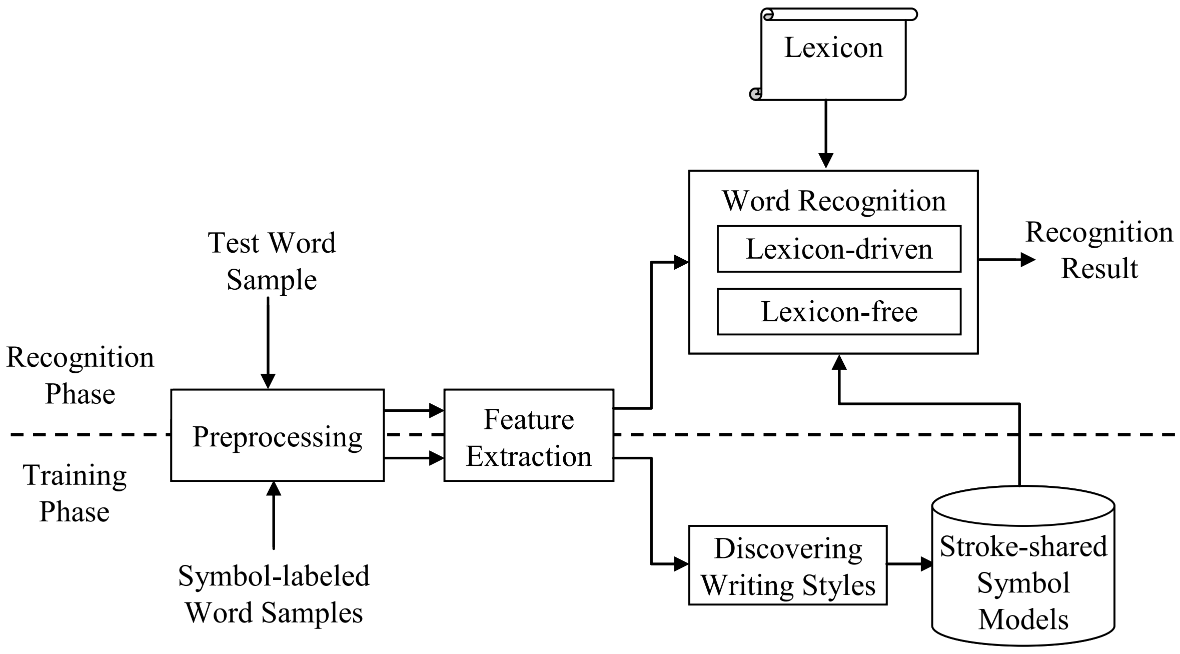


Figure 1.10: High-level architecture of the word recognition system.

In the next chapter, an overview of the OHWR literature for Latin, and Chinese, Japanese and Korean scripts, with an emphasis on HMM-based approaches, is presented along with the research to date for online Indic script recognition. Chapter 3 describes the different stages in the creation of word datasets for Devanagari and Tamil in order to support the training and evaluation of our system. These include definition of appropriate symbol sets for these scripts, data collection from a large number of writers, and annotation (labeling) of the collected word samples at the symbol level. In Chapter 4, the steps of preprocessing and feature extraction are described, along with an algorithm for shirorekha detection for the Devanagari script. Chapter 5 describes our approach for modeling symbols using HMMs, based on *constrained stroke clustering* for automatically identifying writing styles from the dataset. Using these symbol models, we propose and evaluate lexicon-driven and lexicon-free recognition strategies for word recognition for both Devanagari and Tamil in Chapter 6. Our novel BoS representation for words to address the problem of symbol order variation is also described and evaluated in this chapter. In Chapter 7, we describe our solution for Indic handwriting input on small, touch-based mobile devices and evaluate the recognition accuracies of *position-free handwriting* for both Devanagari and Tamil. Conclusions and future directions for our research are outlined in the final chapter.

Chapter 2

Literature Review

This chapter presents an overview of prior work and common strategies for online recognition of Latin and CJK scripts, with a special emphasis on HMM-based approaches. This is followed by a detailed discussion of the state of the art for Indic script recognition covering features, classification techniques and systems for isolated character and word recognition.

2.1 Latin Script Recognition

The Latin script falls into the category of "alphabetic" writing systems. An alphabet consists of a set of letters, and the letters are written sequentially to form words. The Latin alphabet consists of 52 letters (both uppercase and lowercase), ten numerals, punctuation marks and some symbols such as &, $ and @. Some languages that use the Latin script have diacritical marks (accents) added to the fundamental letters. A Latin word is written from left to right and often cursively in one stroke, except for the diacritical marks, dots in 'i' and 'j', and 't' and 'x' crossings. Recognition of cursive words is considered a difficult problem [12] due to ambiguous character boundaries, co-articulation effects on the shape of a character due to its left and right neighbors, and the existence of ligatures (inter-letter connecting patterns). A summary of strategies for Latin script recognition is presented in the following subsections.

2.1.1 Segmentation-based Approach

The segmentation-based approach is also referred to as the 'analytical' or 'model-based' approach [1]. The word is broken down into simpler recognition units and the recognition results on the individual units are then combined to form the word hypotheses. The recognition units may be characters or sub-characters. Identification of characters or sub-characters in a cursive word is not an easy problem. To illustrate the difficulty in segmentation, ambiguities in segmenting the cursive word ‘invade’ are shown in Fig. 2.1. To identify the subunits from a word sample, two approaches are commonly followed - explicit segmentation and implicit segmentation.

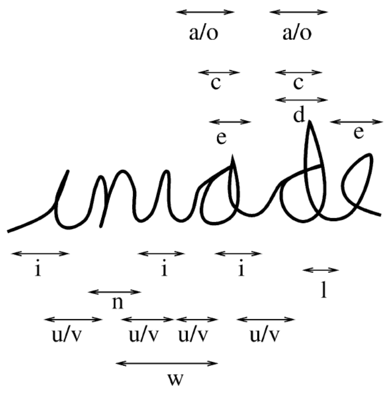


Figure 2.1: Segmentation ambiguities in the cursive word *invade* (adapted from Madhvanath and Govindaraju [1])

Explicit segmentation - Explicit segmentation is a ‘double segmentation’ process [13]. In the first stage, also know as ‘pre-segmentation’, graphemes (subunits of a word) are extracted from the word using landmark features. Each grapheme could correspond to one or more characters in the word. In the second stage, the segmentation points or boundaries are modified using contextual information (or recognition). Explicit segmentation is often error prone because of Sayre’s paradox [14]: “To recognize a letter, one must know where it starts and where it ends; to isolate a letter, one must recognize it first”. The simplest way to carry out explicit segmentation is by constraining the writing, as with the boxed or discrete forms. In the case of non-cursive and discrete character writing, vertical projections of the handwritten word provide cues for segmentation. However, in the recognition of unconstrained writing, explicit segmentation is usually carried out by applying heuristics. An example of this approach is the early work of Frishkopf and Harmon at Bell Labs in the 1960s. Their approach was based on detecting “landmark” features such as ascenders and descenders in a word. Segments are then determined by centering landmarks within the estimated character width. However, the scheme does not work for characters such as ‘u’, ‘n’, ‘m’ etc. which do not contain landmarks.

Implicit segmentation - Implicit segmentation bypasses the segmentation problem by adopting the strategy of ‘over segmentation’. The word is split into several small pieces with the only criterion that none of the pieces should contain parts of more than one character. In other words, the intention is that the actual segmentation points are always a subset of the segmentation boundaries obtained by over segmenting the word. The right groupings of segments that could form characters are then determined based on recognition results. Contextual information such as lexicon may also be used for this purpose. As a result of recognition, the word is automatically segmented into characters. Hence, segmentation is a byproduct of recognition. For this reason, this approach is also known as 'recognition-based segmentation' [15]. The two most popular techniques employed for implicit recognition are Dynamic Programming (DP) and HMMs.

2.1.2 Segmentation-free or Holistic Approach

Techniques that follow the segmentation-free or holistic approach circumvent the problem of segmentation completely by considering a word as a single indivisible unit. Holistic methods follow a two-step process [13]. In the first step, high level features such as ascenders, descenders and word length are extracted from the word. In the second step, the features are compared with those of words present in the lexicon and the most similar word is considered the recognition result. The training of holistic recognition algorithms is inseparable from the lexicon because adding a new word to the lexicon typically requires retraining the recognizer with the samples of that word. When the lexicon is large, similarities between words increase and hence the recognition accuracy deteriorates. Therefore, the approach works well only for applications featuring static and small lexicons such as bank check recognition and postal mail sorting. However, the approach has its own advantages. It is robust as it deals with global word shape and is not as sensitive to local shape variations that are predominant in unconstrained writing. Holistic methods are believed to perform better than segmentation-based methods when the word is badly written and segmenting it into constituent characters is virtually impossible (Fig. 2.2).

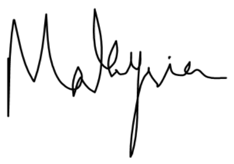


Figure 2.2: A badly written sample of the word *Malaysia*

2.1.3 Human Reading Inspired Approach

Findings from research on human reading capability have inspired the development of recognition systems that attempt to model the human reading process. These systems can be either segmentation-based or holistic, based on the underlying theory. *Word superiority effect* [14] - the observation that it is easier to recognize a letter embedded in a word than in isolation is one such finding. In other words, this effect describes the influence of the word context on the individual letter recognition. A system developed based on this theory is described by Vinciarelli [14]. The architecture of the system consists of three levels corresponding to features, letters and words. The levels are interlinked and work in parallel. The output from a particular level inhibits or excites the neighboring levels. For instance, detection of an ascender in the feature level excites letters such as 'l' and 'k' but inhibits 's' and 'o' at the letter level. Similarly detection of a letter can excite a word at the word level, which in turn can provide feedback to excite a letter that was not previously detected at the letter level. The inhibitory links work in a similar fashion to reduce the excitation of incompatible letters or words.

An experiment to uncover the usage of geometrical features by human readers for the Latin script was carried out by Schomaker and Segers [16]. The experiment was based on enhancing an obscured handwritten word image under a time constraint. The results show that humans pay most attention to the initial and final parts of the word. The results also confirm the findings of previous studies that the sequence of ascenders, descenders, crossings and points of high curvature in the handwriting pattern aid the recognition process.

Steinherz et al. [17] argue that human reading does not happen from left to right; instead, it always tries to identify characters which can be recognized without any ambiguity. Handwritten word recognition based on this theory of human reading works as follows. Letter templates are represented as sequences of features. The features extracted from a handwritten word are compared with the letter templates, and presence or absence of a letter is determined. Features that do not match with any letter template are denoted by blank spaces. The temporary string containing hypothesized letters and blank characters is then replaced with the most similar string in the lexicon.

Word Ending Postulation technique suggested by Powalka [18] is also based on theories of human psychology. People often get sloppy when they are about to complete writing a word. Readers cope with this by using the beginning part of the word, and the context to resolve the identity of the word. Powalka's approach attempts to break up the word into the 'stem' (distinct part) and the indistinguishable segment that needs to be postulated. The postulation point is determined relative to the word length. Recognition of the stem shortlists candidate choices from the lexicon, and the characters in the rest of the word are postulated using features such as the number of vertical down strokes and the width of the segment present in the illegible portion.

2.1.4 HMM-based Approaches

The application of HMMs to Latin script recognition has its roots in speech recognition where it has seen remarkable success. Indeed there are many HWR systems [19] for Latin script built by just replacing the features in existing speech recognition systems with features from handwriting. This is because of the similarities between speech and cursive handwriting. Both speech and handwriting can be considered as signals varying over time. The phonemes in speech correspond to characters in handwriting. Both need to handle co-articulation effects, and another important similarity is that both can make use of language models for improving recognition accuracy. However, there are also dissimilarities. Unlike in speech, word segmentation in Latin HWR is relatively simpler. Further, preprocessing of handwriting has seen greater success than speech because judging the uniformity achieved does not require expertise as it is often visually evident [20]. On the other hand, Latin script HWR suffers from the issue of delayed strokes (or stroke order variation in general) which is not applicable to speech.

The most commonly used HMM topology for both speech and handwriting is the left-to-right model, also know as the Bakis model. One may further constrain the model by not permitting state skipping, resulting in a strictly left-to-right HMM. In this model (Fig. 2.3), the state index is non-decreasing as time increases, the state transition probability a_{ij} is zero for all $j < i$ and $j > i + 1$, and $\pi_i = 1$ only when i is equal to one, or else it is zero. It is interesting to note that there is no evidence that models allowing more complex state transitions would result in better recognition accuracy [20]. These left-to-right HMMs intrinsically impose the temporal order of the signal, wherein preceding states account for observations earlier to those of subsequent states [21].

Most of the efforts for Latin script recognition use HMMs to model isolated letters, and for modeling words, the constituent letter models are concatenated [13, 15, 14]. The advantages of such an approach are: (i) The model set does not increase with the size of the vocabulary and (ii) Adding a new word to the lexicon does not require training of the models. Oh et al.

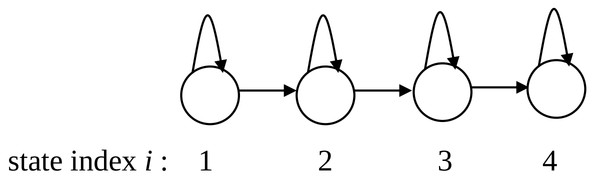


Figure 2.3: Left-to-right HMM without state skipping

[22] see a handwritten word as an alternating sequence of characters and ligatures, and propose a circularly connected network with each edge containing a character or a ligature model. Due to this circular topology, the network is capable of recognizing words of arbitrary length. For training the system, isolated character samples are collected for each style (cursive and discrete) and HMMs are built for each style of the character.

A two-stage approach which combines implicit and explicit segmentation is proposed by Cavalin et al. [23]. Two different sets of character models trained on different features are employed in stages. In the first stage, the character models in the first set are concatenated to form word models and by implicit segmentation (recognition-based segmentation) possible segmentation points are identified and the top N hypotheses are determined. In the second stage, the other set of character models which use different features is used to score the segmentation obtained for each character in the first stage just on the basis of recognition confidence. The combination of the two scores is used to re-rank the hypotheses and the results are found to be better than using only the first stage.

Hu et al. [20] use a left-to-right HMM without any state skipping and the basic modeling units are called "nebulous stroke models". The authors call them "nebulous" because these stroke segments are learnt automatically from the training data and do not correspond to any logical writing unit. Nebulous stroke models are concatenated according to the sequence specified in the character lexicon to form the character model. The advantages mentioned for having shared stroke models are a reduced model set, and having a large number of training samples available per model. The word model is formed by concatenating the individual character and ligature models but the ligature models are allowed to be bypassed in the network to support mixed (cursive as well as discrete) styles of writing.

An HMM topology consists of number of states, connectivity between the states and number of Gaussians used to model the feature vectors in a state. The work by Li et al. [24] deals with optimizing the HMM topology for handwriting recognition. The number of states in a model should ideally depend on the complexity of the character being modeled. For instance it may not be appropriate for the characters '.' and 'W' to have the same number of states. This is made possible by having the number of states be a fraction of the average number of feature vectors per sample of a class or the mode of the feature-vector-count histogram of the class. This is shown to yield better performance than having the same number of states for all the letters. Li's work uses Bayesian Information Criterion (BIC) to find the optimal topology, which is found to produce results comparable with that of heuristic-based approaches, but with fewer parameters. The work also suggests that when each style of lexeme (letter allograph) is modeled by a separate HMM, robustness to writer style variability increases.

Zimmermann and Bunke [25] also compare different schemes such as fixed length modeling, Bakis length modeling and quantile length modeling, to optimize the number of states in a left-to-right HMM used for handwriting recognition. In the work of Han Shu [26], fixed length modeling where each letter is modeled by a seven-state left-to-right discrete HMM is employed. Word models are then formed by concatenating the letter models.

Nathan et al. [27] address the space and time complexity issues involved in applying HMMs for real-time recognition of unconstrained handwriting. A two-stage approach involving "fast match" in the first stage followed by a "detailed match" is proposed. In the fast match stage, a single state degenerate model is used to represent a character. The top N results are then subjected to the detailed match, which re-ranks the recognition results.

2.2 CJK Script Recognition

Due to the pictographic-ideographic nature of CJK scripts [7], they pose different challenges for OHWR compared to the Latin script. The complexity of CJK *character* recognition is said to be comparable with that of Latin *word* recognition [7]. For instance, the basic building units for Latin words are characters, whereas radicals combine together to form Chinese and Japanese characters. Similarly, in the case of Korean Hangul script, there are 51 graphemes consisting of vowels and consonants. These graphemes are combined based on certain rules and arranged in a two-dimensional fashion to form a character. Ligatures are common within and between graphemes, but not across characters.

While numerous approaches have been proposed for online CJK character recognition [6, 7], we only describe HMM-based approaches in this section.

2.2.1 Challenges for Recognition

The recognition challenges in CJK script recognition are different from those in Latin script recognition. In this section we list these challenges and mention a few techniques that have been proposed in the literature to address them.

Stroke-order variation - A multi-stroke character can be written in different ways by following different stroke orders. Even though the overall shape of the character doesn't change with different stroke orders, the change in temporal sequence of strokes poses problems to an online recognizer. Integrating techniques from offline recognition can help solve this problem of stroke-order variation [28]. Methods based on graph (e.g. Attributed Relational Graph (ARG) [29]) matching, and stroke correspondence [6] also address the problem. In a complete ARG, the nodes represent stroke segments and the edges describe the relations between the segments. The shapes of stroke segments are described by their direction codes and length. The relationship between two stroke segments is described using the relative locations of their bounding box centers as (top, below, or aligned), (left, right, or aligned), and whether they cross each other or not.

Stroke-number variation - In cursive Chinese or Korean character writing, two or more strokes can be written in a single stroke causing the stroke number to differ from the standard form of writing the character [28, 30]. The inter-stroke connecting patterns, known as ligatures, also form the principal source of shape variations. Such cursive writing needs to be segmented before recognition. In the work by Sin and Kim for Korean character recognition [31], the problem of stroke-number variation is addressed by modeling ligatures as separate entities along with the grapheme models.

Shape variations - The same character, even if written using a particular stroke order and number, may differ in shape especially across different writers. To handle such variations, a two-stage classification scheme is proposed by Wakahara and Okada [28]. The first stage performs rigid stroke matching and identifies potential candidates. In the second stage, Stroke-based Affine Transformation (SAT) is applied on the input pattern for each of the candidates from the first stage. The transformed input pattern represents the deformation required to match with the reference pattern. The reference pattern which has the minimum distance with the deformed input is declared as the recognized character.

Handling the 2-D nature of the script - Latin characters are normally written in one stroke except for delayed strokes and the diacritical marks over the lowercase characters. These diacritical marks are typically identified using simple heuristics before the recognition of the main stroke is handled. However in the case of CJK, each character is often written in several strokes at different positions (left, right, top or bottom) within the character. Due to the two-dimensional nature of these scripts, unlike in Latin script recognition, spatial information about the constituent strokes plays an important role.

Spatial information can be captured in two forms: absolute and relative. Absolute position of a component (grapheme or stroke) in a character is often determined with respect to the bounding box of the character. Relative positions are determined with respect to other components in the character. Absolute position information is effective for simple characters whereas relative position information improves accuracy for complex characters [32]. Marukatat and Artieres [32] encode the spatial information between strokes in a character using discrete and continuous attributes. Discrete attributes described include vertical position (above/aligned/below), horizontal position (left/aligned/right) and connectivity (touching or not). Continuous attributes corresponding to a stroke include fraction of its bounding box area that is above, below or aligned with the bounding box of another stroke. Attributes that take into account the directions of the strokes are also described.

2.2.2 HMM-based Approaches

Due to the two dimensional nature and stroke order/number variations in CJK scripts, HMMs are not as popular as they are for Latin script recognition [7]. Most of the efforts that apply HMMs for CJK scripts try to break a character down into subunits (radicals, or consonant and vowel graphemes) and then use HMMs to model these subunits.

Kim et al. [33] apply HMMs for Korean character recognition. An HMM network is built to represent the entire set of characters wherein each path corresponds to a Korean character. Since searching through all the paths in the network is computationally very expensive, the authors propose an efficient modified level building algorithm whose time complexity is dependent on the number of graphemes and ligature models in each level, and not on the number of search paths. The graphemes and ligatures are modeled using left-to-right HMMs trained using samples manually segmented from characters. A modified level building algorithm is also employed for Chinese character recognition by Kim et al. [34]. The Chinese characters are represented by a finite state network which also incorporates the grammatical constraints that exist in the radical sequences to form characters. The major disadvantage of the approach is that it does not handle stroke order variation.

Sin and Kim [31, 35] propose a network of HMMs called *BongNet*, for recognition of the characters in the Hangul script. Each HMM models a grapheme or a ligature pattern. While each node in the network represents a cluster of similar graphemes, the transitions represent ligature classes. A path from the start node to the end node corresponds to a Hangul character. The network also accounts for relative position between the graphemes in each character by clustering the graphemes based on end point of the previous grapheme and start point of the following grapheme. The maximum probability path through the network corresponds to the recognized character.

A substroke-based approach for online Kanji character recognition is proposed by Nakai et al. [36]. The Kanji character set has around 6000 characters. As a result, building character-level models requires a large amount of training data and may not fare well in terms of space and time complexity required for real-time recognition. To overcome these problems, substroke models are proposed. Twenty five substrokes based on the eight directions are classified into four categories: long pen-down strokes (8), short pen-down strokes (8), pen-up strokes (8) and pen-down-up stroke (1). Delta x and delta y features are extracted from the input ink and the substrokes are modeled using HMMs (3-state HMMs for pen-down substrokes and single state HMMs for pen-up substrokes). After the substrokes are defined, for all the Kanji characters, a hierarchical dictionary (Fig. 2.4) is built manually. In the hierarchy, the character occupies the topmost level, followed by the radicals that make up the character and finally the sequence of substrokes that form the radicals at the bottom level. To deal with the problem of stroke-order variation, alternate substroke sequences are defined in the dictionary. One major advantage of this approach is the possibility of writer adaptation with fewer samples. Since the substroke model set is shared across all the characters, samples of a few characters are typically sufficient to train the rest. However, the limitations of this approach are: (i) substantial effort is needed to build the dictionary of character, radicals and substroke sequences, (ii) the technique cannot handle unseen stroke orders. Context-dependent substroke models using the same approach are described by Tokuno et al. [37]. These models incorporate not only the variations in a substroke, but also the co-articulation effects due to preceding and succeeding substrokes.

Hasegawa et al. [38] propose a discrete HMM for the problem of Japanese character recognition. The slope features obtained from the input ink trajectory are quantized and used along with pen-up/pen-down information. Each character is modeled by a left-to-right HMM.

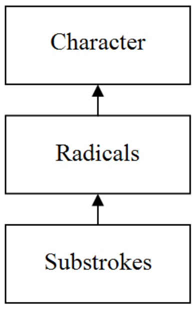


Figure 2.4: Hierarchical representation for Kanji character recognition

In order to deal with the stroke order and shape variations, separate models for different styles are created. The decision to create a separate model is taken based on the normalized log-likelihood ratio – the ratio of the likelihood of a trained model H producing a trained sample, to that of the likelihood of model H producing the current input sample. If this ratio exceeds a predefined threshold a new model is created and trained using the input sample.

2.3 Indic Script Isolated Character Recognition

In the first chapter, we saw that characters (syllabic units) of varying complexity form the basic building blocks of Indic scripts, and are difficult to recognize for a number of reasons. A number of early efforts in the literature have focused on recognition of simple characters such as independent vowels and isolated consonants. The primary challenges in recognizing these characters, as previously outlined, come from their number, shape complexity and symbol and stroke order variations.

2.3.1 Strategies

There are several strategies possible for recognition of isolated characters:

1. Characters may be viewed as compositions of strokes
2. Characters may be viewed as compositions of C, C', V and M graphemes
3. Characters may be viewed as indivisible units

In the stroke-centric strategy [39, 40, 41, 42], a set of unique stroke shapes that constitute all characters is found and characters expressed as combinations of these strokes. The number of unique strokes in most Indic scripts is believed to be less than 300. A key problem with this strategy is that these strokes are not known in advance for a given script, and are a function of writing styles. The determination of unique strokes is performed manually by analysis of training data, and are estimated as 123 and 98 for Devanagari and Tamil respectively [42]. However these approaches are rule-based, script-specific and may require significant manual intervention in the training phase.

The second strategy leverages the internal graphemic structure of the character. Recognition of a character is a result of recognizing the (much smaller set of) constituent graphemes. Potential segmentation points are typically at stroke transitions, and alignment using Dynamic Programming may be used for segmentation. HMMs may also be used to solve this problem implicitly. For Telugu script recognition [43], basic graphemes including core characters and ligatures, summing up to 141 when their positions are ignored, are manually identified and then HMMs are used to model each one of them. A major advantage of this strategy is the reduced effort involved in data collection as only samples of the identified graphemes are required. In its simplest form, this strategy does not address stroke order variations across symbols in the character, symbol order variations, co-articulation effects, and the cases where opaque symbols are created by CC and CV combinations. It is highly effective when constraints may be imposed on the writer to aid recognition.

The last strategy of treating complex characters directly as pattern classes has to deal with their large numbers. It also requires large quantities of data for training. Consequently this strategy has been explored primarily for simple characters such as isolated vowels and consonants in different Indic scripts. It has also been used for Tamil characters, which as mentioned earlier are limited in number due to the linearization of consonant conjuncts [44, 45, 41, 46]. An obvious advantage of this strategy is that standard character recognition techniques in the literature may be used without much knowledge of the structure of the script, as is evident from the results of the Tamil character recognition competition organized in conjunction with IWFHR-10 [47]. Joshi et al. [48] assume that Devanagari can be “linearized” like Tamil by constraining writers to unravel consonant clusters into sequences of vowel-muted consonant characters. The linearization assumption reduces the number of characters to 441 - still a large number - when vowel modifiers are not considered. An accuracy of around 90% is reported for writer-dependent recognition using the subspace based method.

2.3.2 Preprocessing

Preprocessing techniques for Indic scripts are similar to those used for other scripts. Dehooking, smoothing, resampling, size normalization etc. are commonly performed. In the work by Swethalakshmi [42], different types of size normalization for Devanagari strokes are investigated. Genetic Programming has also been explored as a way of designing an optimal scaling function that reduces classification error [49].

Nonlinear normalization which has been found to be effective for CJK scripts does not appear useful for Indic scripts [42]. In the case of Devanagari, the *shirorekha* is often detected and removed prior to recognition. During word recognition the shirorekha also serves as a valuable cue for detecting core lines.

2.3.3 Features

Features used for Latin scripts have been found to be useful for Indic scripts as well. Low-level features such as the normalized x and y coordinates have been widely used. Additional features such as normalized first and second derivatives and curvature have shown promising results for Tamil and Telugu [50, 51]. Structural features such as cusps, bumps, loops and semi-loops have also been explored for Tamil [41, 52] and Devanagari [42] character recognition. In some early work on Tamil character recognition [53], angle features, Fourier coefficients and Wavelet features are compared using a Neural Network classifier. Angle features are shown to be susceptible to noise leading to high intra-class variability. On the other hand, Fourier coefficients do not capture subtle differences between two characters as the change in the values of x and y over a small interval of time gets nullified over the entire frequency domain. Wavelet features are shown to be the most effective for Tamil as they retain both the intra-class similarity and inter-class differences. In general, directional coding approaches popular for CJK scripts are not effective for Indic scripts since the strokes do not have simple shapes.

In the work of Toselli et al. [50], a combination of time-domain and frequency-domain features is shown to improve recognition accuracy on Tamil characters. For Telugu character recognition, Rao and Ajitha [54] propose the use of x and y extrema, direction of pen motion (clockwise/anticlockwise) and relative displacement from the previous point of the same extrema category (x or y).

Offline features which model the input as a raster image rather than a trajectory may also be used to improve recognition accuracy when compared to using online features alone [55]. Being invariant to stroke order, number and direction variations, offline features are very promising in the context of Indic script recognition.

2.3.4 Classification

A number of different classification techniques have been applied to the problem of Indic script character recognition. While some approaches [42] are aimed at recognizing all the characters in the script, others [55, 43] only address specific subsets. These classification approaches may broadly be categorized as follows:

- Template matching
- Rule-based approaches
- Neural Networks
- Hidden Markov Models
- Subspace-based approach

Template matching - In this approach, the features extracted from the test character are compared with those of stored prototypes or templates. The test character is assigned the label of the template that is most similar to it. The templates could either be samples selected from the training set or a categorical representation [56]. In the context of Indic script recognition, efforts based on template matching are aplenty and a majority of them have reported encouraging results. For example, a two stage classification scheme [46] using Nearest Neighbor (NN) classifiers is described for *writer-dependent* Tamil character recognition. The first stage filters the templates based on the Euclidean distance from the test sample, and the second stage computes the more expensive Dynamic Time Warping (DTW) distance (Fig. 2.5) from the shortlisted templates. The label of the nearest template is assigned to the test sample.

The same scheme has also been applied for *writer-independent* Telugu and Tamil character recognition using a different set of features [51]. For the problem of Telugu character recognition, Rao and Ajitha [54] perform a coarse matching with the templates using the number of X-Y extrema points in the test sample and a fine matching using Dynamic Programming.

In another effort on Tamil character recognition [57], templates are identified from the training set using Agglomerative Hierarchical Clustering and Learning Vector Quantization

(LVQ) with DTW as the distance measure. A DTW-based NN classifier is then employed for matching the test sample.

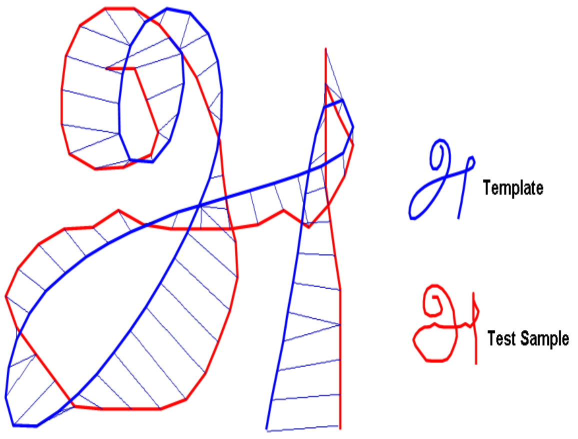


Figure 2.5: DTW for matching two samples of a Tamil character

Rule-based approaches - These approaches do not have an explicit training phase; instead they exploit human knowledge about the problem. The task of classification now becomes a deterministic verification procedure. Although the approach has the advantage of requiring minimal training data, it suffers from being labor-intensive and highly script-specific. Another disadvantage is that the approach typically does not provide alternate recognition choices. In a recent effort on Devanagari character recognition [42], strokes are first classified using Support Vector Machines (SVM) and predefined rules are then used for grouping the stroke labels into characters.

In the work of Ranade [56] for the Devanagari script, a set of stroke templates is derived from analysis of common writing styles of different Devanagari characters, and each character is represented by a set of combinations of these stroke templates.

In another effort [41], Tamil strokes are represented as strings of shape features. In order to recognize an unknown stroke, its equivalent feature string is computed. The test stroke is then identified by searching the database using a flexible string matching algorithm. Once all the strokes in the input are known, the character is determined using a Finite State Automaton.

Prior knowledge about popular writing styles has also been exploited to design a first stage classifier for Tamil characters [52]. The authors observe that the start of any Tamil character is either a line, semi-loop or a loop. Accordingly, the candidate choices are pruned during recognition.

Neural Networks - In the work of Kunte and Samuel [58], Feed-forward Neural Networks with a single hidden layer are used for the recognition of handwritten Kannada characters. The authors use approximation coefficients derived from Wavelet decomposition on the preprocessed (x,y) as features for representing characters. The input character is initially classified into its consonant group (defined as a consonant and any of its vowel combinations), and separate Neural networks are used for further classification within the consonant group.

The Neural Network based approach is also adopted by Sundaresan and Keerthi [53] for the recognition of online Tamil characters. Their work compares the performance of Time Delay Neural Network (TDNN) and a single hidden layer network for the classification task. Due to the presence of similar-looking characters and high dimensionality of the input, TDNN exhibits poor performance when compared to the single hidden layer network. The work also studies the relevance of different features such as (x,y) coordinates, sequence of directions, curvature, sequence of cosine angles, and wavelet features.

Another example is the use of Multi-layer Perceptrons (MLP) trained on eight-direction code histogram features for the problem of Bangla character recognition [59].

Hidden Markov Models - While HMMs have been used widely for English character recognition, they have seen limited application for Indic scripts. Connell et al. [55] use a combination of two HMM classifiers trained with online features, and three NN classifiers each trained on different sets of offline features, for Devanagari character recognition. The combination of online and offline classifiers is shown to improve the accuracy from 69.2% (online, HMM alone) to 86.5%.

HMMs have also been used for Telugu [43] and Tamil [50] character recognition. Each character is modeled as a left-to-right HMM and a combination of time-domain and frequency-domain features is shown to result in accuracies higher than using either of them alone.

Subspace based approach - In this approach, Principal Component Analysis (PCA) is applied separately to feature vectors extracted from the training samples of each class. The subspace formed by the first few eigenvectors is considered to represent the model for that class. During recognition, the test sample is projected onto each subspace and the class corresponding to the one that is closest is declared as the recognition result (Fig. 2.6). Joshi et al. [48] apply the subspace method to writer-dependent Devanagari character recognition. The shirorekha and vowel modifiers are pre-classified using heuristics before recognizing the core character using the subspace method. The approach has also been employed for Tamil character recognition [44]. In these efforts, the feature vectors used for eigenanalysis are composed of the normalized (x,y) coordinates extracted following equi-spaced resampling of the character trajectory. Consequently different feature dimensions may not have stable interpretations (e.g. as salient points along the trajectory), and the method as used is outperformed by DTW-based template matching [45].

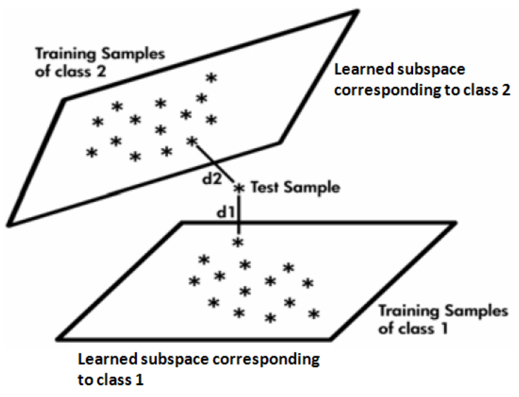

Training Samples of class 2
Learned subspace corresponding to class 2
d2
Test Sample
d1
Training Samples of class 1
Learned subspace corresponding to class 1

Figure 2.6: Subspace based approach for character recognition (adapted from Balaji et al. [2])

In a recent effort by Sundaram and Ramakrishnan [60], two-dimensional Principal Component Analysis is applied for the recognition of Tamil characters. In this approach, multi-dimensional features extracted at each point of the character sample are stacked in the form a matrix called the ‘character matrix’. Recognition of a test character is carried out by computing its Mahalanobis distance to the projection on each character subspace learnt during the training phase. An accuracy improvement of 3% is reported over the simpler PCA-based method described previously.

Recognition accuracies of some of the systems described are shown in Table 2.1. The accuracies in the case of Devanagari are not comparable as the datasets used and the classes addressed differ widely.

2.4 Indic Script Word Recognition

Word recognition for Indic scripts is a nascent area of research, and published literature on the topic is limited. The general strategies for word recognition for Indic scripts may be classified

Table 2.1: Reported accuracies for Indic script isolated character recognition

System	**Number of Classes**	**Features**	**Classification**	**Accuracy**
		Devanagari		
Swethalakshmi [42]	123 strokes	X-Y coordinates, Fourier descriptors and structural features	SVM and rule-based stroke regrouping	89.88%
Connell et al. [55]	40 simple characters	Online and offline	Combination of HMM and NN classifiers	86.5%
		Tamil		
Vision Objects [47]	156 characters	Online and offline	Neural Networks	93.53%
Bulacu M. [47]	156 characters	X-Y coordinates	NN Classifier using DTW distance	91.20%
Toselli et al. [47, 50]	156 characters	Time-domain and frequency-domain	HMM	90.72%
		Telugu		
Prashanth et al. [51]	141 core characters and ligatures	X-Y coordinates, normalized first & second derivatives and curvature	NN Classifier using Euclidean and DTW distance	89.77%
Babu et al. [43]	141 core characters and ligatures	Time-domain and frequency-domain	HMM	91.6%
		Bangla		
Bhattacharya et al. [59]	50 simple characters	8-direction code histogram	MLP	83.61%

broadly along the lines of those available for Latin scripts (Section 2.1) into analytic approaches based on explicit segmentation, those based on implicit segmentation, and holistic approaches.

Some efforts that adopt the explicit segmentation approach [48, 61] presuppose that Indic scripts may be written as a sequence of space separated or boxed characters. As already observed, this is not natural given that syllabic units can vary greatly in size and complexity. Further, there may be multiple ways of decomposing syllabic units into simpler symbols, and different possible symbol orders adopted while writing in boxed form. However preliminary studies suggest that "standard" ways of decomposing complex characters and ordering symbols can be successfully learned as part of a training phase [61].

The stroke-based approach adopted by Swethalakshmi [42] does not require boxed input, but performs explicit segmentation based on proximity analysis for grouping strokes into characters. Proximity analysis is based on the spatial information between consecutive strokes in a character. For scripts such as Devanagari where the most proximal strokes need not be consecutively written (e.g. */ko/*, 'को'), a two-stage proximity analysis is proposed. In the first stage, consecutive strokes are grouped together based on the spacing between their bounding rectangles. In the second stage, character-like units whose bounding boxes overlap with one another are grouped together. Any residual over-segmentation or under-segmentation of characters is resolved in a postprocessing step when the stroke labels are used to classify the characters. Given the interpretation of individual syllabic units, the best word level interpretation is determined using a lexicon or other language model.

Explicit segmentation has also been applied for cursive Bangla word recognition [62]. The segmentation points are determined based on the intersection of the ink trajectory with the computed 'headline'. Based on the segmentation results, the input word is represented as a sequence of stroke segments. During recognition, each stroke segment of the test word is recognized as one of the 73 groups (determined manually during the training phase) using the Modified Quadratic Discriminant Function (MQDF) classifier. From the stroke labels, the constituent characters are determined through table lookup. Word recognition accuracy of 82.34% is reported even in the absence of a dictionary or language models. However, since the training of the system involves significant manual intervention during stroke categorization and construction of the table of character entries, the extensibility of the approach to other Indic scripts may not be straightforward.

2.5 Summary

In this chapter a brief overview of the established approaches for Latin and CJK scripts was presented, with an emphasis on HMM-based approaches. In the second part of the chapter, we attempted to provide a bird's eye view of the state of the research for online recognition of Indic scripts. It can be seen that research in recognition of larger units of writing such as words and phrases is at a very nascent stage, and to date, there has been no work on recognition of unconstrained handwritten words.

Chapter 3

Creation of Word Datasets for Devanagari and Tamil

Given the scarcity of online handwriting datasets for Indic script recognition research, the structure of the scripts, and ambiguities in representation and lack of common definitions (e.g. for units smaller than a character), the creation of standard datasets for training and evaluation is the first challenge one encounters. This chapter describes the methodology we adopted in creating datasets of handwritten word samples for Devanagari and Tamil.

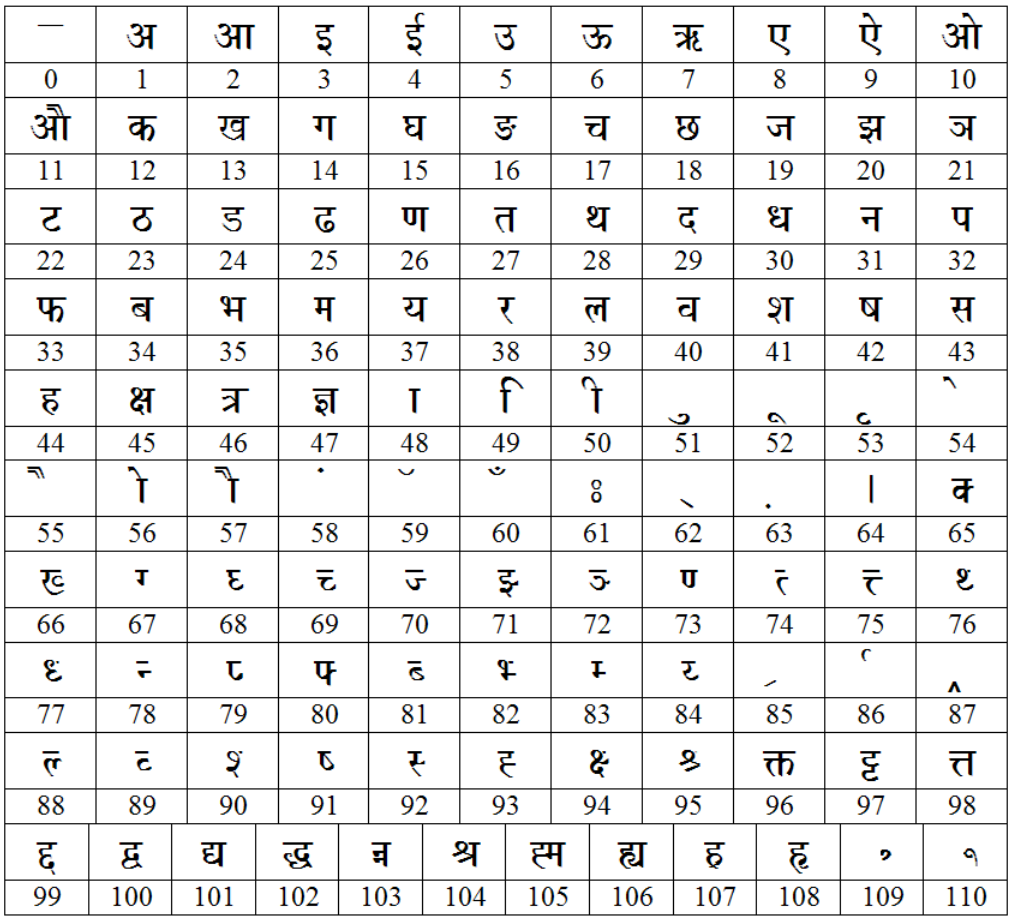
3.1 Identification of Symbols

It is important to first identify the basic units of writing in the script. These basic units, which are referred to as "symbols" in the thesis, form the building blocks of our recognition approach. Figures 3.1 and 3.2 show the symbols we have identified for Devanagari[1] and Tamil respectively.

In the list for Devanagari, symbol 0 is *shirorekha*, symbols 1 to 11 are independent vowels, symbols 12 to 44 correspond to consonants with implicit vowel sound, 48 to 63, 109 and 110 are matras, symbol 64 is to indicate sentence ending, the half-consonants range from 65 to 95, and symbols 45 to 47 and 96 to 108 correspond to conjuncts.

In Tamil, symbols 0 to 10 correspond to vowels, 11 (*aytham*) is a special symbol, 12 to 33 correspond to consonants with implicit vowel sound, and symbols 72 to 80 correspond to matras. A consonant gets converted to its half form when symbol 72 (vowel-muting diacritic) is

[1]The symbols for Devanagari are shown with *shirorekha*, only for better readability. While labeling the word samples shirorekha is considered as an independent symbol.



—	अ	आ	इ	ई	उ	ऊ	ऋ	ए	ऐ	ओ
0	1	2	3	4	5	6	7	8	9	10
औ	क	ख	ग	घ	ङ	च	छ	ज	झ	ञ
11	12	13	14	15	16	17	18	19	20	21
ट	ठ	ड	ढ	ण	त	थ	द	ध	न	प
22	23	24	25	26	27	28	29	30	31	32
फ	ब	भ	म	य	र	ल	व	श	ष	स
33	34	35	36	37	38	39	40	41	42	43
ह	क्ष	त्र	ज्ञ	ा	ि	ी	ु	ू	ृ	े
44	45	46	47	48	49	50	51	52	53	54
ै	ो	ौ	ं	ॅ	ँ	ः	्	़	।	क
55	56	57	58	59	60	61	62	63	64	65
ख्	ग्	[illegible]	ट्ट	[illegible]	झ्	[illegible]	ण्	[illegible]	[illegible]	[illegible]
66	67	68	69	70	71	72	73	74	75	76
[illegible]	[illegible]	[illegible]	फ्	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	^
77	78	79	80	81	82	83	84	85	86	87
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	ह	क्ष	[illegible]	क्त	ट्ट	त्त
88	89	90	91	92	93	94	95	96	97	98

द्	द्व	द्य	द्ध	न्न	श्र	ह्म	ह्य	हृ	हु	[illegible]	[illegible]
99	100	101	102	103	104	105	106	107	108	109	110

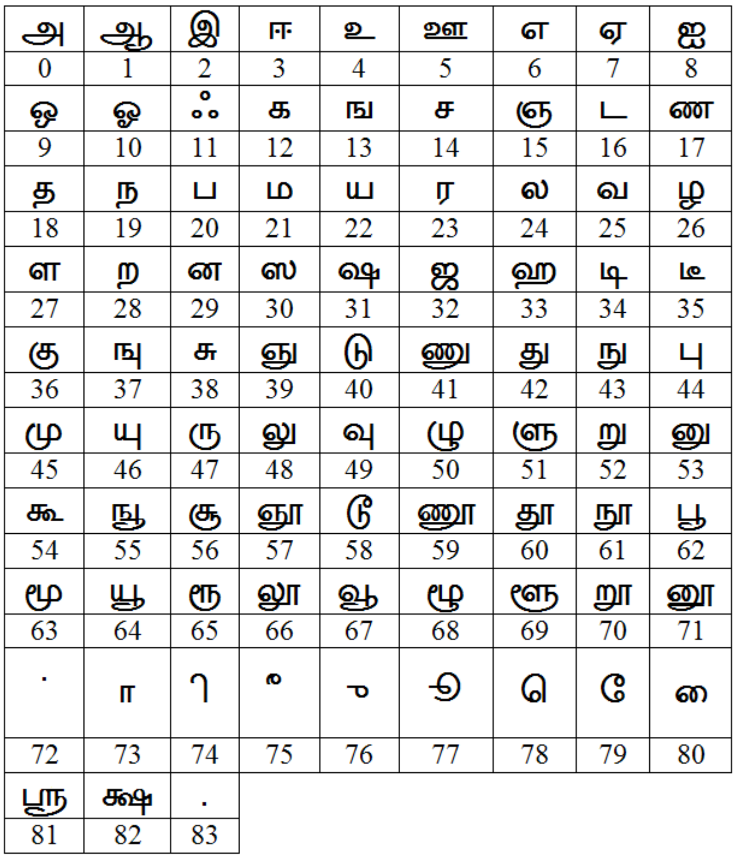
Figure 3.1: Symbol set defined for Devanagari

placed above it. Symbol 81, which always occurs with 75, and symbol 82 are conjuncts. Symbol 83 corresponds to the period. The rest of the symbols are distinct syllabic units formed by consonant-vowel combination where both the consonant and the matra have lost their individual identities, and hence are best represented as unique symbols.

The following important aspects were taken into account while defining the symbol set:

1. *Coverage* - A general-purpose recognition system is expected to recognize all the characters and words in the script. Therefore, the symbol set for such a system would have to be large enough to allow any character (syllabic unit) to be expressed as a combination of one or more symbols in the set. This is unlike Latin scripts where the symbols for recognition and the letters in the alphabet are essentially the same.

2. *Position* - As mentioned in Chapter 1, there are certain matras in the script which are very similar in shape but vary only in their relative positions in the context of a character. The position of a matra may also change with respect to the base consonant to which it is



அ	ஆ	இ	ஈ	உ	ஊ	எ	ஏ	ஐ
0	1	2	3	4	5	6	7	8
ஒ	ஓ	ஃ	க	ங	ச	ஞ	ட	ண
9	10	11	12	13	14	15	16	17
த	ந	ப	ம	ய	ர	ல	வ	ழ
18	19	20	21	22	23	24	25	26
ள	ற	ன	ஸ	ஷ	ஜ	ஹ	டி	டீ
27	28	29	30	31	32	33	34	35
கு	ஙு	சு	ஞு	டு	ணு	து	நு	பு
36	37	38	39	40	41	42	43	44
மு	யு	ரு	லு	வு	ழு	ளு	று	னு
45	46	47	48	49	50	51	52	53
கூ	ஙூ	சூ	ஞூ	டூ	ணூ	தூ	நூ	பூ
54	55	56	57	58	59	60	61	62
மூ	யூ	ரூ	லூ	வூ	ழூ	ளூ	றூ	னூ
63	64	65	66	67	68	69	70	71
·	ா	ி	ீ	ு	ூ	ெ	ே	ை
72	73	74	75	76	77	78	79	80
ஸ்ரீ	க்ஷ	.						
81	82	83						

Figure 3.2: Symbol set defined for Tamil

attached. For instance, the */uu/* matra in Devanagari appears on the right for */ra/* (Symbol 110 in Fig. 3.1) but at the bottom for all other consonants (Symbol 52). Therefore, these are captured as two different symbols while defining the symbol set.

3. *Shape Consistency* - While there are fused CV characters (e.g. Tamil symbols 34 to 71 in Fig. 3.2) which are often written as a single symbol, there are symbol patterns that may not have any linguistic interpretation but occur consistently as distinct shapes across different characters. For instance, the s-shaped lower part of Telugu */ka/* (Fig. 1.1 of Chapter 1) appears across all CV combinations of the consonant */ka/* but by itself does not represent any character. Such shapes may be considered as independent symbols based on knowledge of the script and the popular writing styles.

4. *Linguistic interpretation* - Defining symbols that have linguistic interpretation has several advantages from the perspective of recognition. For instance, it becomes possible to directly adopt the grammar of the script to prune invalid recognition results. Further, one can also incorporate language models to improve word recognition accuracy. In ad-

dition, when lexicons are used, conversion of UNICODE to symbol IDs becomes simpler when the symbols correspond to linguistic units. The use of linguistically-valid symbols also simplifies data collection, as they may otherwise appear unfamiliar to the writers, especially when seen in isolation.

While shape consistency is important from a practical stand point, linguistic interpretation is essential to develop standard representation units. In defining the symbol sets above, we have strived to find the right tradeoff between the two aspects [63].

3.2 Handwriting Data Collection

In this section, we describe the process used for data collection, including details of the devices and the software tools used. In order to collect data samples, writers were prompted to write a set of words, and allowed to write naturally without imposing any constraints on their writing style.

3.2.1 Identification of Words for Data Collection

The first step in data collection was to identify a *minimal* set of words that covers all the defined symbols and can be used as prompts for handwriting data collection. Collection of word samples, as opposed to isolated symbols or syllabic units, is important in order to understand various factors such as: (i) the degree of cursiveness in the script as practised by native writers, (ii) the change in the shape of a symbol due to the influence of its neighbors (also known as co-articulation effect), (iii) symbol order variations within and across syllabic units, and (iv) stroke order variations that occur at the word level (i.e. delayed strokes).

Unlike Latin scripts where *pangrams* (sentences that use every letter of the alphabet at least once) are popular for data collection, similar sentences for Indic scripts are uncommon owing to the large number of characters. In order to identify the minimal set of words, we made use of a large text corpus [64]. We identified the unique words in the corpus and eliminated the low-frequency words, assuming that these unfamiliar words would lead to unnatural writing if collected. We then ran an Optimal Text Selection (OTS) program [65] on the remaining words, which in turn applied the Set Cover Algorithm to determine an optimal subset that covers all the symbols.

The OTS program identified a total of 85 words for Tamil and 70 words for Devanagari, which were then used as prompts for handwriting data collection.

3.2.2 Data Collection for Tamil

Word samples for Tamil were collected using a Tablet PC which had a sampling rate of 1200 Hz and a spatial resolution of 2500 dpi (dots per inch) along both X and Y directions. A total of 132 writers belonging to different age groups contributed handwriting samples. Each writer was prompted to provide two samples of 30 words selected randomly from the 85 words selected by the OTS program. A majority of the writers who participated in the data collection activity were native Tamil speakers and used the Tamil script everyday. Figure 3.3 shows a screenshot of the data collection tool that runs on a Tablet PC.

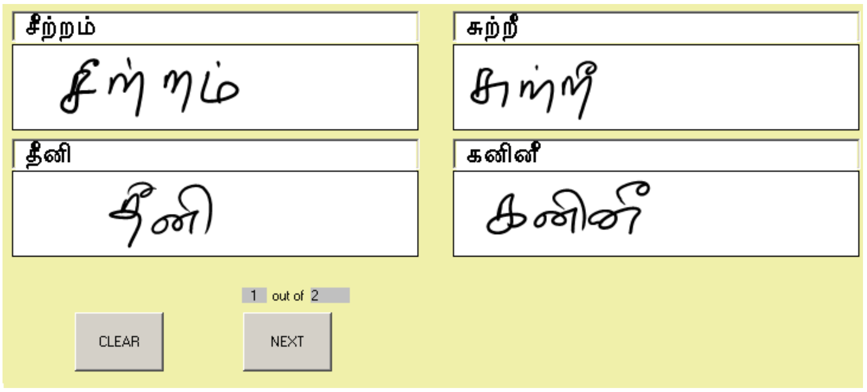
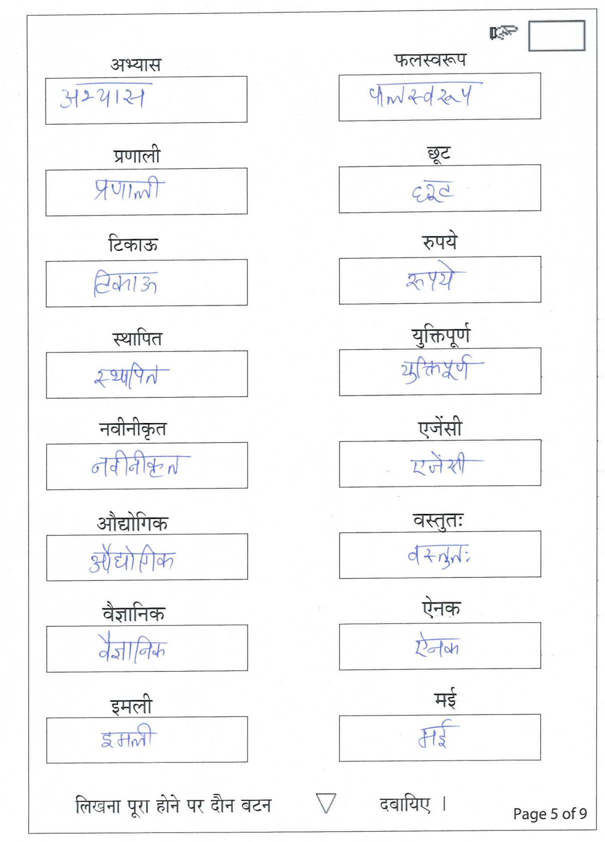


Figure 3.3: Tablet PC based data collection tool used for Tamil

3.2.3 Data Collection for Devanagari

Even though interactive devices such as the Tablet PC and PDA are appropriate and convenient for online handwriting data collection, they fail to recreate the feel of writing on paper. For Devanagari, we experimented with the ACECAD Digimemo device [66] for collecting handwriting samples. The Digimemo is a portable electronic clipboard which digitally captures and stores what is written on ordinary paper using a special pen, and thus provides a more natural interface for collecting handwriting data. It also provides an affordable alternative to devices such as TabletPCs and PDAs for larger scale data collection efforts. Devanagari handwriting was collected from a total of 110 writers of different age groups and each writer was provided with an A4-sized booklet of printed paper forms clipped to the Digimemo pad. Each form contained the words to be written and empty boxes for writing them in (Fig. 3.4).



अभ्यास	अभ्यास	फलस्वरूप	फलस्वरूप
प्रणाली	प्रणाली	छूट	छूट
टिकाऊ	टिकाऊ	रुपये	रुपये
स्थापित	स्थापित	युक्तिपूर्ण	युक्तिपूर्ण
नवीनीकृत	नवीनीकृत	एजेंसी	एजेंसी
औद्योगिक	औद्योगिक	वस्तुतः	वस्तुतः
वैज्ञानिक	वैज्ञानिक	ऐनक	ऐनक
इमली	इमली	मई	मई

लिखना पूरा होने पर दौन बटन ▽ दवायिए ।

Page 5 of 9

Figure 3.4: A sample data collection ‘paper’ form for Devanagari

Both the Tablet-based and Digimemo-based data collection tools[2] store the collected ink in the UNIPEN [67] format along with writer information such as name, age, gender, educational background, and left/right handedness.

3.3 Data Cleanup

Some of the collected word samples contained different types of noise due to erratic pen movements made by the writer, or as a result of fatigue faced by the writer especially when writing the last few samples, or due to hardware errors in the device while registering the ink. We developed a simple HTML-based data cleanup tool that displays all the samples from a selected

[2]The data collection tools are a part of the open source Lipi Toolkit from HP Labs and are freely available for download at http://lipitk.sourceforge.net

writer, and allows one to mark the samples that are noisy. The marked samples were deleted using a batch script. A screenshot of the tool is shown in Fig. 3.5.

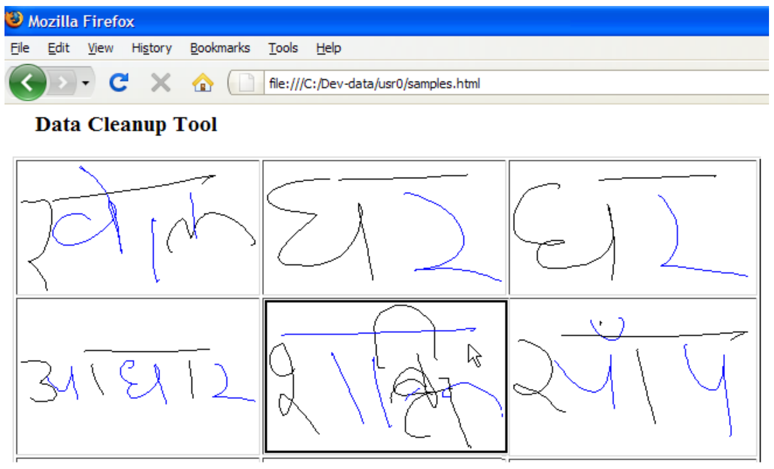

Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Dev-data/usr0/samples.html
Data Cleanup Tool

Figure 3.5: Data cleanup tool

3.4 Annotation

The cleaned word samples were annotated manually using a GUI tool (Fig. 3.6). The annotation of each word sample was carried out at the symbol level by labeling one or more strokes that together form a particular symbol, with the ID (truth) of that symbol as in Figures 3.1 and 3.2. Additional annotation captured information about the quality of the symbol samples, whether two or more symbols are written together in a single stroke, and if a stroke is a delayed stroke written out of sequence. Once a word sample is annotated, the tool updates the ink file with its annotation information using the UNIPEN format.

The word samples and associated annotation were organized in the form of a file hierarchy categorized progressively by writer ID, word ID and trial number.

3.5 Summary

This chapter described the methodology we adopted for creating handwriting datasets for Devanagari and Tamil, beginning with the definition of a symbol set covering the graphemes in the scripts while meeting various other criteria such as coverage, position, shape consistency and linguistic interpretation. The chapter also briefly described the different steps in the process

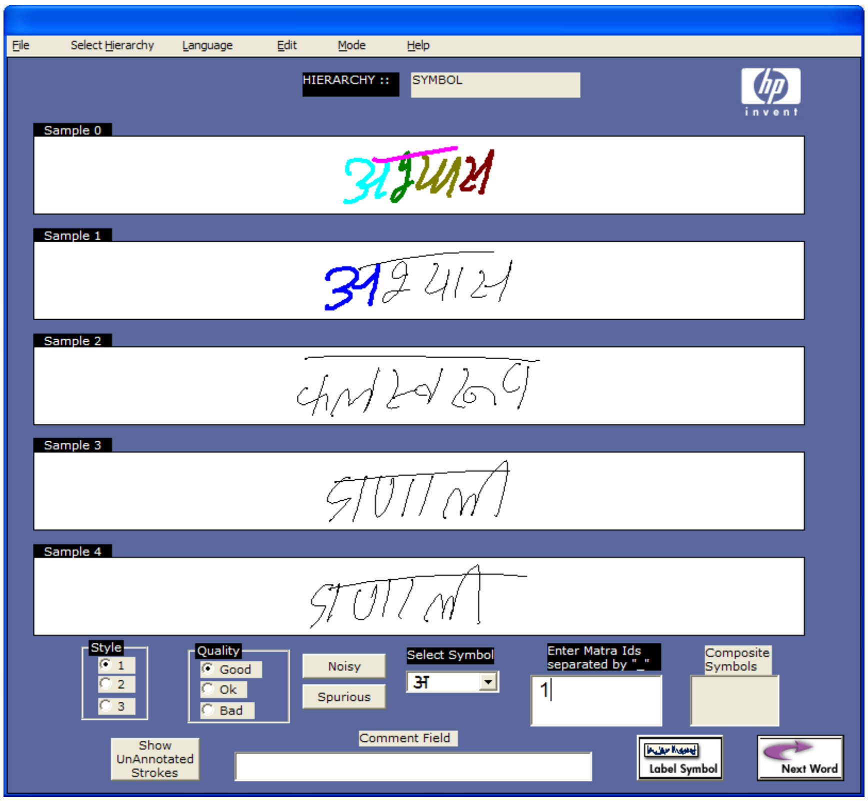


Figure 3.6: Data annotation tool

such as data collection, cleaning and annotation, and the tools that we used in each step. The result of the data collection exercise was the development of datasets for Devanagari and Tamil containing word samples from over a hundred writers each, organized by writer and annotated at the symbol level. These datasets were used for training and evaluation of the symbol modeling and word recognition approaches described in the remaining chapters.

Chapter 4

Preprocessing and Feature Extraction

In this chapter, the different steps involved in preprocessing the raw ink corresponding to a given word sample, and extracting features from it are described. Preprocessing includes the steps of size normalization and resampling, both of which are quite common for the recognition of Latin scripts as well. However, in the case of Devanagari, the first step of preprocessing is to detect the stroke(s) that correspond to *shirorekha*, the top horizontal stroke of a character or word.

4.1 Shirorekha Detection for Devanagari

A distinct feature of the Devanagari script is the presence of shirorekha (Fig. 4.1). When written as part a word, shirorekhas of individual characters are often, but not always, merged into a single stroke barring a few exceptions. In this section, we first summarize some of the issues in detecting the shirorekha present in an online handwritten Devanagari word or character. We then describe our approach for shirorekha detection, including the features and the algorithm used.

The published literature for online shirorekha detection is very limited. Namboodiri and Jain [68] address the problem of identifying the individual scripts present in a multi-script document and exploit the characteristics of the shirorekha stroke in this context. Even though their work does not address the problem of shirorekha detection directly, the features extracted based on height and width of the strokes are noteworthy. Joshi et al. [48] describe a heuristic-based shirorekha detection algorithm in the context of online Devanagari character recognition. For each stroke in the character, features such as mean (x, y), length and 8-directional histogram are computed. All strokes whose directional histograms have more than 60% frequency along the East or West direction are added to Shirorekha Candidate Set (SCS). This set is further pruned by removing the lowermost stroke of the character, if present. The stroke that has the maximum length in the SCS is then declared as the shirorekha. A major limitation of their approach having been developed for isolated characters is that it cannot identify multiple shirorekha strokes present in larger input units such as words. Moreover, since the technique does not model the position of the strokes it may lead to more ambiguities, e.g. when the lengths of the strokes in the SCS are comparable.

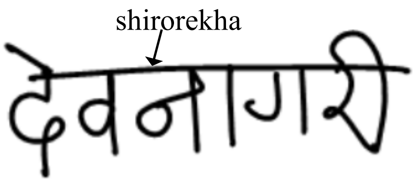


Figure 4.1: Shirorekha - the top horizontal stroke in a Devanagari character or word

4.1.1 Issues in Online Shirorekha Detection

Some of the issues to be addressed in order to accurately identify the shirorekha stroke(s) in a given word sample are the following:

1. *Multi-part nature* - Even though shirorekha is normally written as a single stroke spanning the entire word, there are a few notable exceptions. For instance when consonants such as भ */bha/*, ध */dha/* and थ */tha/* are present in the middle of a word, the shirorekha is broken up as shown in Fig. 4.2.

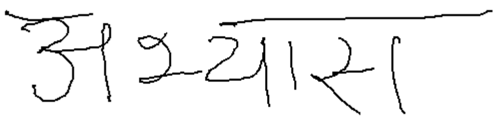


Figure 4.2: Multi-part shirorekha

2. *No fixed vertical position* - Although shirorekha is perceived as the topmost stroke in a word, the vertical position of the stroke with respect to the word may change significantly based on the presence of matras as shown in Fig. 4.3.

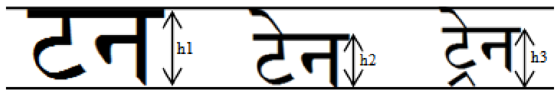


Figure 4.3: Variability in vertical position of shirorekha

3. *No fixed writing order* - Our analysis of the collected word samples shows that when the shirorekha is written as a single stroke, it is the last stroke in approximately 90% of the cases. On the other hand, in the case of a multi-part shirorekha, the constituent strokes may get temporally interleaved with other symbols and there is no guarantee that they are written sequentially or from left to right.

4. *Segmentation* - Sometimes the shirorekha and a symbol are written together as a single stroke (Fig. 4.4) calling for a sophisticated segmentation technique to separate them.

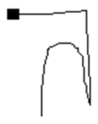


Figure 4.4: Shirorekha written along with the character (black square shows the starting point of the stroke)

5. *Variability in writing* - Even as the length of the shirorekha increases with the length of the word, one may find that due to the inherent variability in handwriting, the stroke is often not written perfectly horizontally. Also in the case of a multi-part shirorekha, the constituent strokes may not be perfectly aligned vertically. These differences necessitate better modeling of the shirorekha stroke.

6. *Shape similarity* - In a word or a character in Devanagari, there are many other strokes (e.g. the horizontal strokes in अ */a/* or स */sa/*) that resemble the shirorekha in shape. Therefore it is important to capture the vertical position of the strokes for accurate determination of the shirorekha.

4.1.2 Features

In our approach for shirorekha detection, we compute and use three features from each stroke in the input. While the *aspect ratio* feature is absolute in that it depends only on the size of the stroke under consideration, *height fraction* and *width fraction* are relative to the word size.

1. *Aspect ratio* of a stroke is defined as:
$$Aspect\_Ratio = \frac{height(stroke)}{width(stroke)}$$
For a stroke to be a candidate for shirorekha, its aspect ratio is expected to be less than ASPECT_RATIO_MAX.

2. *Height fraction* captures the vertical location of the stroke with respect to the word or character and is defined as follows:
$$Height\_Fraction = 1 - \left(\frac{\bar{y}(stroke)}{y_{max}(word)}\right)$$
where $\bar{y}(stroke)$ is the average y value of the stroke and $y_{max}(word)$ corresponds to the y value of the lower most point in the word. For two strokes to form a part of the shirorekha, their difference in $Height\_Fraction$ is not expected to be more than a threshold (NON_ALIGN_MAX).

3. *Width fraction* captures the extent to which the stroke spans horizontally across the word, according to the equation below:
$$Width\_Fraction = \frac{width(stroke)}{width(word)}$$
where $width(stroke)$ and $width(word)$ correspond to the bounding box width of the stroke and the word (including the stroke under consideration) respectively. The sum of $Height\_Fraction$ and $Width\_Fraction$ of a shirorekha stroke is expected to be at least SIZE_POS_MIN.

The optimum values for ASPECT_RATIO_MAX, NON_ALIGN_MAX and SIZE_POS_MIN were determined empirically from the histograms of the feature values using the training data.

4.1.3 Algorithm

The features extracted are used to identify the shirorekha stroke(s) in the input word sample, as described in Algorithm 1.

4.1.4 Evaluation

The proposed algorithm for shirorekha detection was evaluated using a word dataset, different from the training dataset used for learning the thresholds. The training dataset contained approximately 14,000 word samples written by 44 writers. The test dataset contained approximately

Algorithm 1: Pseudo-code for shirorekha detection in Devanagari

```
Input: Set of ink strokes corresponding to a word or a character where $n$ is the number
    of strokes
Output: Shirorekha List $S$ containing indices of strokes that correspond to shirorekha
1: $S \leftarrow \phi$, Candidate List $CS \leftarrow \phi$
2: for $i \leftarrow 1$ to $n$ do
3:     Compute $Aspect\_Ratio_i$, $Height\_Fraction_i$ and $Width\_Fraction_i$ for stroke
       with index $i$
4:     Compute $Size\_Pos_i = Height\_Fraction_i + Width\_Fraction_i$
5:     if $Aspect\_Ratio_i <$ ASPECT_RATIO_MAX and $Size\_Pos_i >$ SIZE_POS_MIN then
6:         Add $i$ to $CS$
7:     end if
8: end for
9: if $CS$ is empty then
10:    return $S$
11: end if
12: Sort $CS$ in descending order using $Size\_Pos$
13: Add $CS[1]$ to $S$ /* adding stroke with largest $Size\_Pos$ to the shirorekha list */
14: for $j \leftarrow 2$ to size($CS$) do
15:    if $|Height\_Fraction_{CS[1]} - Height\_Fraction_{CS[j]}| <$ NON_ALIGN_MAX then
16:        if stroke $CS[j]$ not horizontally subsumed by any stroke in $S$ then
17:            Add $CS[j]$ to $S$
18:        end if
19:    end if
20: end for
21: return $S$
```

2500 word samples contributed by a different set of 20 writers. The accuracy of shirorekha detection was found to be 99.3%. The shirorekha in the input word sample was considered to be recognized correctly only if all the strokes forming it were detected with no false positives.

It is important to note that the proposed technique is capable of identifying even the absence of shirorekha. However, it does not segment the shirorekha when written along with a symbol in a single stroke (Fig. 4.4), which is quite rare. Some of the word samples where not all the shirorekha strokes were detected correctly are shown in Figure 4.5.

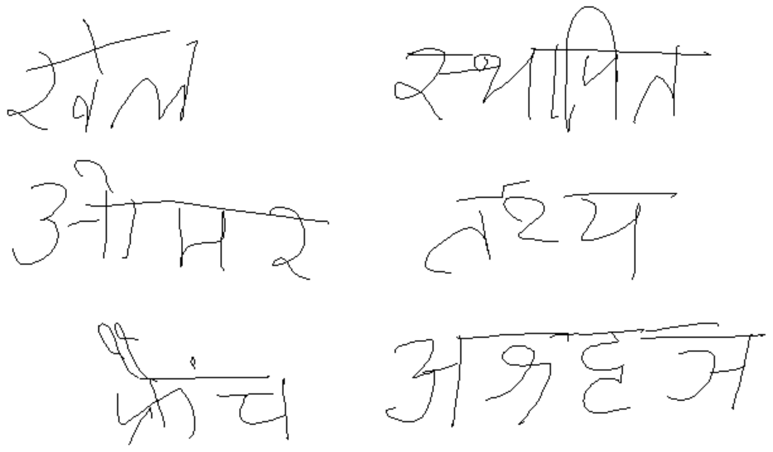
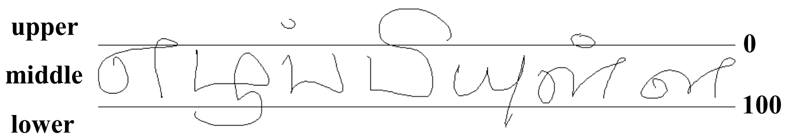


Figure 4.5: Examples of strokes not detected or falsely detected as shirorekha by the proposed algorithm

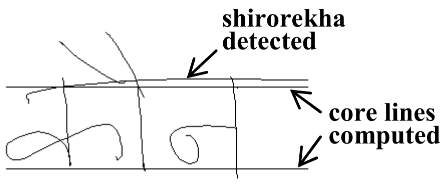
4.2 Word Size Normalization

Word size normalization is frequently performed to deal with writer-specific variations in size, and is often preceded by determination of virtual core lines present in the handwritten word. For Indic scripts, upper and lower core lines are generally used to delineate the upper matra zone, the middle zone and the lower matra zone.

We implemented a simple algorithm for determining the core lines. The mean value of y across all points in the word sample was first computed. The strokes that intersect with the line $y = y_{mean}$ were then identified. The mean values of y for the lower most and upper most points in the intersecting strokes were taken to be the lower and upper core lines respectively. Figure 4.6(a) shows the estimated core lines for a word sample from Tamil. To achieve size normalization, the distance (h) between the two lines was scaled to a constant value (100) while retaining the aspect ratio of the word sample.



(a)



(b)

Figure 4.6: Normalization of word size for (a) Tamil (b) Devanagari

In the case of Devanagari, the upper core line of a word was determined from the average y value of the shirorekha strokes. However, when no shirorekha was found, the core lines were computed as described earlier. A Devanagari word normalized using the automatically-determined shirorekha stroke is shown in Fig. 4.6(b).

4.3 Resampling

Resampling is performed to obtain a set of points along the trajectory that are uniformly sampled in space, whereas the input data captured from the device is the result of uniform sampling in time. This step is especially necessary for a writer-independent system to deal with writing speed variations across writers. Typically, the original points are replaced with a new set of points regularly spaced in arc length using piece-wise linear interpolation. In our resampling method (Fig. 4.7), the distance between two consecutive points was fixed to a fraction ($1/15$th) of the distance (h) between the core lines. When a stroke was identified as a dot based on a size threshold, the average values of x (i.e. x_{avg}) and y (i.e. y_{avg}) in the stroke were computed and the entire stroke was replaced with the point (x_{avg}, y_{avg}), duplicated a constant number of times.

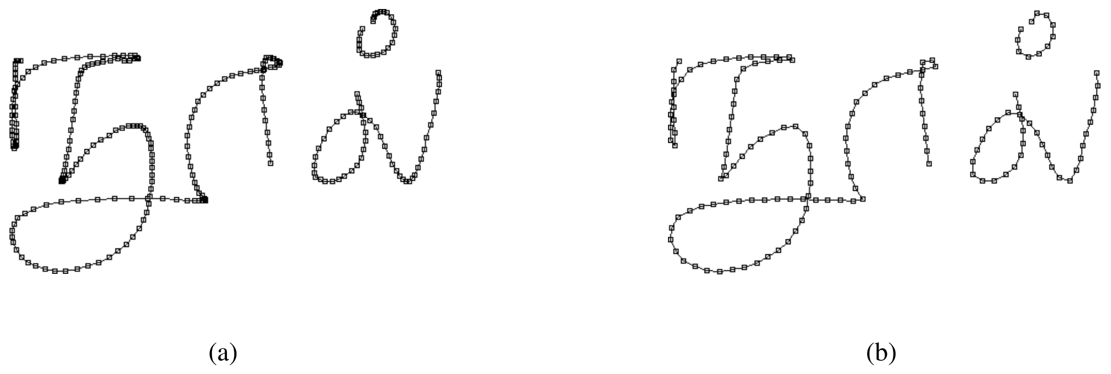


Figure 4.7: Word sample from Tamil: (a) before resampling (b) after resampling

4.4 Feature Extraction

While the objective of preprocessing is to eliminate variations in the input that might adversely impact recognition, in the feature extraction step, characteristics that best represent the shape of the trajectory are extracted. After experimenting with a number of features, we used a set of 9 features adapted from the NPen++ recognition system [69]. The features extracted at each point in the trajectory include normalized y value, four angle features that capture writing direction and curvature, 'aspect', 'curliness', 'linearity' and 'slope'. While each feature is described here in brief, detailed descriptions of these features may be found in the NPen++ paper [69].

1. *Normalized* y - The y value at each point of the normalized and resampled word. This is the most common feature used in the literature for online word recognition.

2. *Angle features* - They are widely used in word recognition systems due to their translation and scale invariant nature. The angle features extracted capture the writing direction and curvature of the trajectory. The writing direction is represented using the cosine and sine of the angle subtended by the line segment joining the neighboring points on either side with the horizontal line. The curvature at a point is represented by the cosine and sine of the angle formed by the line segments joining the point of consideration and its second neighboring points on either side.

3. *Aspect* - It captures the ratio of the height of the bounding box to its width, and is given by

$$A(t) = \frac{\Delta y(t) - \Delta x(t)}{\Delta y(t) + \Delta x(t)}$$

where $\Delta x(t)$ and $\Delta y(t)$ are the width and the height of the bounding box containing the points in the 'vicinity' of the point under consideration. In all our experiments, we have used a vicinity of length 11 i.e. five points to the left and right of the point at which the feature is extracted.

4. *Curliness* - Curliness at a point gives the deviation of the points in the vicinity from the line joining the first and last point. It is expressed as:

$$C(t) = \frac{L(t)}{max(\Delta x, \Delta y)} - 2$$

where $L(t)$ is the sum of all the line segments along the trajectory in the vicinity of the point under consideration.

5. *Linearity* - It is defined as the average squared distance between every point in the vicinity and the line joining the first and last point. It is expressed as:

$$LN(t) = \frac{1}{N} * \sum_i {d_i}^2$$

6. *Slope* - It is given as the cosine of the angle formed by the line joining the first and last point of the vicinity with the x-axis.

4.5 Summary

In this chapter, the different steps in preprocessing the input handwritten word and extracting features from it, were described. For Devanagari preprocessing, identification of shirorekha in the input word sample is an essential first step. An algorithm that makes use of coarse stroke features, and threshold values learnt from training data was proposed to identify one or more shirorekha strokes in the input. The performance of the algorithm was also evaluated on a test set and was found to be more than 99% accurate. We then proposed a simple technique to identify the virtual core lines present in a word sample in order to normalize its size. In the case of Devanagari, the vertical position of the shirorekha stroke(s) identified was assumed to correspond to the upper core line. Finally, we described a set of features extracted from each point along the trajectory of the input ink which included writing angles, aspect, curliness, linearity and slope.

In the next chapter, we describe how HMM-based symbol models may be constructed from the extracted features.

Chapter 5

Symbol Modeling

5.1 Introduction

In this chapter, we address the subproblem of modeling symbols in Indic scripts using HMMs. As discussed in the introductory chapter, Indic characters[1] are often characterized by variations in the temporal ordering and number of strokes written. This phenomenon is by no means limited to Indic scripts. For example, Figure 5.1 shows character samples from the IRONOFF dataset [70] where the uppercase English character ‘H’ is written in different ways. It is clear that the number, order and shapes of strokes can vary widely. Apart from the ordering of strokes, the direction of a stroke may also vary across writers. Such alternate ways of writing a character are broadly referred to as *styles* in this chapter. They have also been referred to as ‘lexemes’ [71] or ‘dynamic allographs’ [72] in the literature. Two samples of a character class are said to be written in different styles if they differ in the shape (including position), number, ordering, or direction of the constituent strokes.

While a single writer may have a specific style of writing a character, styles are bound to differ across writers. Even though such variations may not alter the visual appearance of the character written, they may have serious implications for OHWR as most of the commonly-used algorithms such as HMM and DTW are sequence-dependent. Therefore it is important for a writer-independent recognition system to model at least the common writing styles during the training phase, using data collected from a large user population.

[1]In this chapter, the terms ‘character’ and ‘symbol’ are used interchangeably. This is to emphasize that the proposed approach is applicable not only for symbols in Indic scripts but also for characters in alphabetic scripts such as the Latin.

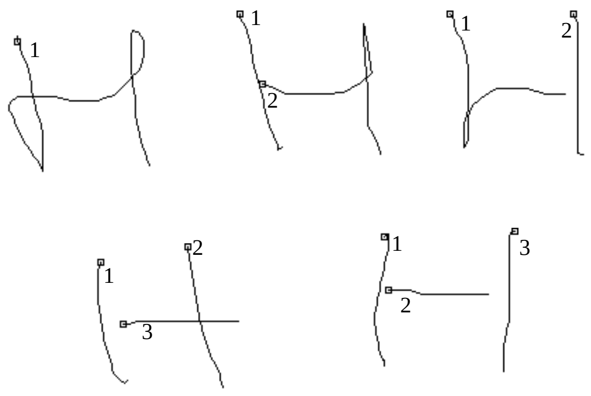


Figure 5.1: Different writing styles of ‘H’ identified from the IRONOFF dataset. The numbers indicate the order of writing. The start of each stroke is indicated with a square box.

5.1.1 Modeling Writing Styles

When using model-based approaches such as HMMs, many studies have shown that it is advantageous to independently model each style of writing instead of having a single model represent all the styles of a character class [71][73][74][55][75]. When all the samples of a character class are modeled using a single HMM following the commonly-used left-to-right topology, the HMM may under-fit the training data due to substantial differences in the feature values of samples belonging to different styles.

As previously mentioned, different character writing styles may be thought of as variations in the number, order, shape etc. of the strokes constituting the character. There are two popular strategies to model these variations: *character-based modeling* and *stroke-based modeling*. In the character-based approach, the variations are modeled implicitly at the character level by clustering the samples of the character as a whole and treating each resulting cluster as a subclass within the character class. On the other hand in the stroke-based approach, the variations in the character are modeled explicitly as sequences of stroke labels, where these stroke labels represent ‘distinct’ strokes obtained by clustering the available stroke samples within and/or across character classes. Thus, character-based modeling of writing styles requires clustering at the character level whereas stroke-based modeling requires clustering at the stroke level. When HMMs are used for modeling, an HMM is built for each character or stroke cluster accordingly.

5.1.2 Advantages of Stroke-based Modeling

When compared to character-based modeling, stroke-based modeling of a character is especially suitable for scripts with a large number of multi-stroke characters, such as the Indic family of scripts, and offers numerous advantages. Stroke clustering carried out as part of stroke-based modeling can potentially identify the minimal set of distinct strokes that occur across different styles of the same character or different character classes, and be used to describe all the character writing styles represented in the available data, in an optimal manner. Figure 5.2 shows how an HMM model of the Devanagari character */ka/* may be built. Each path corresponds to a writing style, and each labeled circle corresponds to a stroke HMM. The figure shows how stroke HMMs may be shared across different styles of the character class.

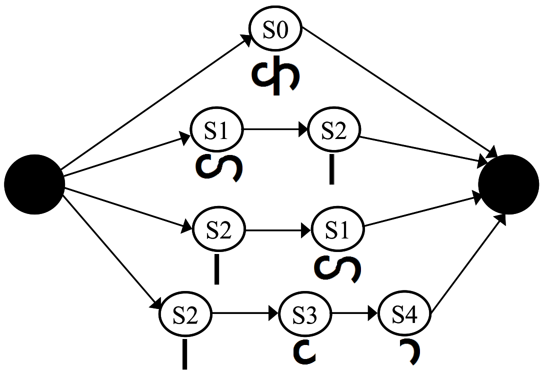


Figure 5.2: Stroke-based HMM Model of the Devanagari character */ka/* representing different writing styles (paths) and with sharing of stroke models.

The identification of such common strokes and modeling characters as sequences of strokes significantly reduces the storage requirements for capturing distinct character writing styles, and can reduce the computation time in the context of character recognition. Since the strokes are shared across styles and/or character classes, fewer character samples may be sufficient to train the models [20]. As a corollary, the recognition system can be adapted to a specific user with a smaller number of training samples. Moreover, a new style of writing differing only in stroke order, can easily be supported by simply defining the corresponding style in the dictionary [36], thereby circumventing the need for more samples and fresh training. The sharing of stroke labels across character classes also allows the design of stroke-based recognition strategies [76][36][77], and enables the realization of certain incremental handwriting-based text entry methods (e.g. QuickStroke [78]).

For these reasons, our primary concern in this chapter is with stroke-based modeling of character writing styles by stroke-level clustering, in order to obtain a *minimal set of strokes and styles*. The main contributions may be summarized as follows:

1. While unsupervised clustering techniques are popular in the OHWR literature, we propose a novel approach based on *constrained clustering* for clustering strokes to determine

the minimal stroke set. We show how simple handwriting domain specific constraints can be exploited to obtain better stroke clusters when compared to unsupervised approaches.

2. We demonstrate how the aforementioned constraints can be derived automatically from the data, and independent of the script.

3. We apply the approach to online Devanagari character (symbol) recognition using HMM, and show that our approach allows the modeling of variety of styles, while significantly reducing the number of states when compared to character-based modeling, with almost no compromise on accuracy.

5.2 Related Work

In this section, we briefly survey the literature for character-level and stroke-level clustering and modeling of writing styles. While character-level clustering and modeling has been applied widely for Latin scripts, stroke-level clustering and modeling is popular for oriental scripts - Chinese, Japanese and Korean. This is because of the inherently multi-stroke nature of these scripts and presence of similar-looking strokes across different characters.

5.2.1 Character-level Clustering and Modeling

In this subsection, we review some of the techniques described in the literature for character-level clustering and modeling. The approaches for character-level clustering can broadly be classified as *supervised* and *unsupervised*, based on whether or not the labels of character samples are used to obtain desired clusters, and have been used primarily for prototype selection. Character-level clustering has also been used for HMM modeling of writing styles in various scripts.

Supervised Clustering

Supervised clustering approaches such as those based on Learning Vector Quantization (LVQ) [79] and Minimum Classification Error (MCE) [80] exploit the class labels of the character samples to derive informative prototypes (from the perspective of recognition accuracy) across all classes. With LVQ, the character prototypes from unsupervised clustering serve as initial seeds for an iterative procedure that morphs them over successive iterations based on their labels and classification results. In contrast, MCE based approaches pose prototype selection as an optimization problem wherein the objective is to minimize the classification error of the training samples. The prototypes (or clusters) resulting from these approaches are expected to contain discriminative information in them, yielding good classification performance.

We focus in the rest of this section on prior work related to unsupervised techniques for character-level clustering. For their supervised counterparts, the reader is referred to the study by Liu and Nakagawa [81].

Unsupervised Clustering

A majority of past efforts that employ unsupervised character-level clustering address the problem of English character recognition (e.g. digits, lowercase and uppercase characters). DTW is a popular distance measure for clustering character samples [82][83][84][85]. Some of these efforts disallow two samples that differ in the number of strokes from being assigned to the same cluster, by setting their DTW distance to infinity [84][82]. Among the different clustering algorithms, Hierarchical Clustering has been widely adopted for identifying writing styles [82][86][84][87][88]. Vuori and Laaksonen [82] study different versions of Agglomerative Hierarchical Clustering (AHC) with different cluster validity indices and note that determining the correct number of clusters is a difficult task and may require human intervention. Connell and Jain [83] represent each sample by an N-dimensional feature vector where each dimension i corresponds to its proximity with the ith sample of the class. Clustering is carried out using the Means Squared Error (MSE) criterion and they address the problem of deciding the optimum number of clusters by determining the 'knee' of the MSE versus number-of-clusters plot. When AHC is employed, one may also apply an empirically-determined threshold on the merging distance to obtain a desirable set of clusters [86][84].

Modeling of Writing Styles

While character-level clustering is popular in the context of Nearest-Neighbor (NN) classification to reduce memory and computation time [83], it has also been used when HMMs are used for recognition. A character is modeled as a parallel network of one or more "style HMMs", depending on the number of character clusters i.e. distinct styles of writing the character. Peronne and Connell [75] proposes an integrated approach for clustering and training style HMMs for English word recognition, where the distance measure for K-Means clustering is the class-conditional probability obtained from the style HMM. Apart from English, character modeling using multiple character style HMMs has also been successfully adopted for other scripts such as Kanji [74], Hangul [73] and Devanagari [55].

5.2.2 Stroke-level Clustering and Modeling

Supervised clustering techniques used for clustering characters are in general not applicable for stroke-level clustering due to the non-availability of class labels for the strokes forming the character samples. Manual labeling at the stroke level is a laborious task, often potentially biased by the individual's judgement, and hence generally not adopted. One example of such manual labeling in the context of the Devanagari script is the work of Shwethalakshmi et al. [42] wherein the distinct strokes and styles in the script are identified manually. By and large, the techniques used for stroke-level clustering and modeling are unsupervised and as mentioned earlier, have widely been applied to CJK scripts.

Nakai et al. [36] identified 25 substrokes in Kanji characters based on their direction and length, and claim that any Kanji character can be expressed as a sequence of these substrokes resulting in a reduced model set. While a hierarchical dictionary consisting of substrokes, strokes, radicals and characters is manually built for Kanji character recognition, the new stroke orders are learnt by generating permutations of the known stroke orders in the dictionary and matching with the samples in the training data [76].

Yamasaki [77] proposes a two-stage stroke clustering approach for Japanese character recognition. In the first stage, the samples of a character class are categorized according to the number of strokes they contain. The sets of strokes having the same time index within each category are considered as the initial stroke clusters. Due to stroke order variations, a stroke cluster corresponding to a time index may have samples that are dissimilar. Therefore, a 'top-down cluster splitting step' is adopted to divide clusters that have different stroke shapes. The decision to partition a cluster ensures that the farthest sample from the mean stroke is at a distance less than the threshold. The same criterion is also applied in the 'bottom-up cluster merging' step where clusters of strokes with different time indices are grouped together. Finally, the stroke representatives of the clusters within a character class are clustered across character classes. The resulting stroke prototypes are used to determine different styles of a character that retain the original stroke sequence and position information.

Prior work on identifying writing styles in Devanagari is relatively limited. While Connell

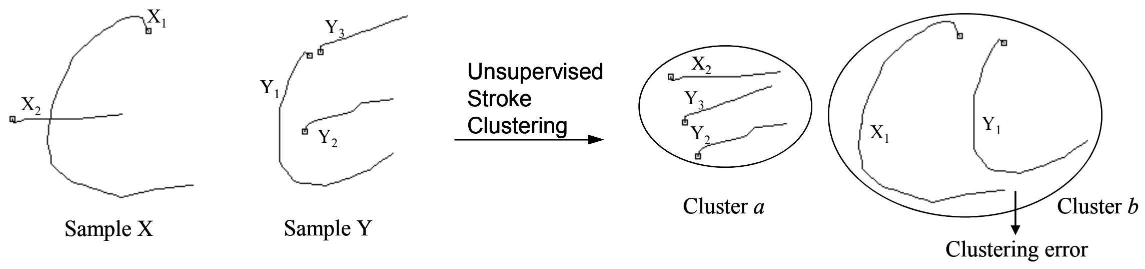


Figure 5.3: Clustering error from unsupervised stroke clustering

et al. [55] employ character-level clustering for identifying writing styles in Devanagari, to the best of our knowledge, there has been no previous work on automatically determining the writing styles with stroke sharing for Devanagari character recognition, which is the focus of our approach.

5.2.3 Limitations of Unsupervised Stroke Clustering

In an unsupervised stroke clustering based approach, resulting styles are typically represented as sequences of stroke cluster labels. In our earlier work on style identification [89], when stroke clustering was attempted across the basic Devanagari characters, we found that some of the resulting styles were 'invalid'. When we disregarded the ordering and considered each style merely as a collection[2] of stroke cluster labels, we found that there were pairs of styles of the same character class where one style was a sub-collection of the other.

We illustrate this with an example in Figure 5.3, which shows two samples of uppercase letter 'E' written in two different styles. Sample X contains two strokes whereas Sample Y has three. Unsupervised stroke clustering correctly assigns strokes X_2, Y_2 and Y_3 to the same cluster (say with label a), but wrongly clusters stroke X_1 and Y_1 into one cluster (say with label b). As a result, the style corresponding to Sample X is identified as $b{\rightarrow}a$ and that of Sample Y as $b{\rightarrow}a{\rightarrow}a$. However, when the temporal order of the strokes is ignored and each style is regarded as a collection of strokes, this leads to the strange situation wherein the style of Sample X i.e. $\{a, b\}$ becomes a sub-collection of the style of Sample Y i.e. $\{a, b, b\}$. Given that both samples belong to the same character class 'E', one cannot contain an extra shape in addition to the shapes present in the other, implying that one of them is not a valid style of the character

[2]The term 'collection' is used instead of 'set' to indicate that the stroke labels forming a style may not be unique.

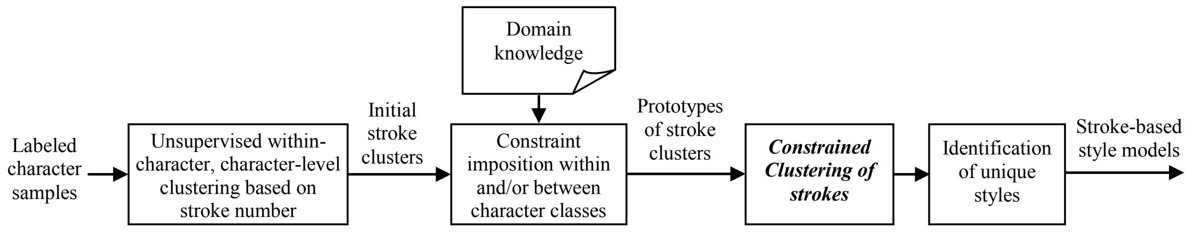

Domain knowledge
Labeled character samples
Unsupervised within-character, character-level clustering based on stroke number
Initial stroke clusters
Constraint imposition within and/or between character classes
Prototypes of stroke clusters
Constrained Clustering of strokes
Identification of unique styles
Stroke-based style models

Figure 5.4: Proposed approach for incorporating handwriting domain knowledge into stroke clustering for obtaining optimal set of strokes and styles

class. This situation could have been avoided if strokes X_1 and Y_1 had been correctly assigned to two different clusters by the clustering algorithm.

The *sub-collection rule* that was violated in our example is generally applicable to any two samples of a character class. A few exceptions do exist, e.g. uppercase letter 'Z', which may be written in a single stroke as 'Z' or with an additional horizontal bar in the middle as 'Ƶ'. As a result, the sub-collection rule may not be applicable in this case. The application of the sub-collection rule to such cases may result in redundant clusters, i.e. two clusters containing the same shape 'Z', which may still be acceptable when compared to having invalid styles resulting from unconstrained over-clustering.

In the next section we describe a new approach for stroke clustering that avoids the issue of invalid styles by enforcing the sub-collection rule.

5.3 Writing Style Identification using Domain Constraints

While a majority of the approaches for stroke clustering are either unsupervised or heuristic-based, in this section, we propose a novel approach based on *constrained clustering* that attempts to identify the minimal set of strokes and writing styles for the given samples of handwritten characters. The approach, illustrated in Figure 5.4, involves 4 steps:

5.3.1 Initial Stroke Clustering

Given the labeled character samples, the objective of the first step in our approach is to obtain initial stroke clusters. The stroke clusters are obtained from unsupervised character-level clustering. Unlike approaches where all the samples of a character class are clustered together, the samples are first categorized based on the number of strokes they contain [77] and within each of the resulting groups of samples, character-level clustering using a conventional *unsu-*

pervised clustering algorithm is carried out. The initial stroke clusters are then obtained from each character cluster by assigning the same label to the stroke samples that have the same time index. For instance, from a character cluster that has two-stroke samples, two stroke clusters are derived by assigning all the first strokes to one cluster and the second strokes to the other. It is important to note a difference between this first step and the approach adopted by Yamasaki [77]. There, character-level clustering is not carried out but instead all the character samples with the same number of strokes are assigned to the same group. The strokes with the same time index are then clustered using a heuristic-based, divisive clustering algorithm within a group and agglomerative clustering across groups and other character classes.

Initial categorization based on stroke number leverages the fact that two samples that differ in their stroke number will necessarily belong to two different styles. As a result, there must be at least as many number of styles for a character class as there are different stroke numbers using which its samples are written. Instead of explicit categorization, one may also set the distance between two samples that vary in their stroke number to infinity [84][82] and cluster all the samples together. Preliminary categorization based on the stroke number has twofold advantages: Firstly, within a group containing samples of equal stroke number, any two samples that do not belong to the same style will differ in the ordering, direction or shape of strokes they contain. Since either of these variations will typically result in significant dissimilarity in the feature space, one may expect the clusters within each group to be well-separated. Secondly, the smaller number of samples within each group when compared to all the samples of the character helps reduce both time and memory requirements of subsequent clustering.

5.3.2 Constraint Generation

The stroke clusters obtained using the character-level clustering process are only the initial clusters. A reduced set of stroke clusters is then obtained by combining similar-looking clusters across groups formed within different character classes by performing a second level of clustering. However, as mentioned in Section 5.2.3, if the second-level stroke clustering is *unsupervised*, it may result in invalid styles. Therefore, our approach pro-actively attempts to avoid the possibility of deriving invalid writing styles by imposing constraints between the initial stroke clusters. The constraints encode handwriting domain specific knowledge, are script-independent and can automatically be determined from the samples using only the character class labels. We generate three types of constraints:

1. *Trivial across-character constraint* - This is the most intuitive constraint one may apply during stroke clustering. According to this, two single-stroke samples belonging to different classes cannot be assigned to the same cluster. Thus, the constraint allows us to leverage the class label information available at the character level to obtain better stroke clusters.

2. *Trivial within-character constraint* - When there are two samples in a character class - one written in a single stroke and the other in two strokes, none of the strokes in the latter sample can possibly fall into the same cluster as that of the former. If it does, then the style of the first sample becomes a sub-collection of that of the second, which violates the *sub-collection rule* described earlier in Section 5.2.3.

3. *Non-trivial within-character constraint* - This is a generalization of the previous constraint and can be stated as follows. For a given character class, if there is an m-stroke sample S_m and an n-stroke sample S_n where $m < n$, and if one-to-one correspondence has been established between any $m - 1$ strokes of S_m and $m - 1$ strokes of S_n, then the remaining stroke of S_m cannot fall into any of the clusters of the remaining $n - m - 1$ strokes of S_n. Let us examine how such a constraint could have averted the clustering error earlier shown in Fig. 5.3. If we could 'automatically' determine that stroke X_2 is similar to stroke Y_2 (or Y_3) i.e. they should be assigned to the same cluster, then we can impose a constraint that stroke X_1 cannot fall into the clusters of strokes Y_1 and Y_3 (or Y_2). Thus, we can eliminate the possibility of deriving invalid styles described in Section 5.2.3.

While the constraints introduced above were in the context of individual samples, they can be directly extended to the level of clusters. For example, the first constraint may be restated as: two different stroke clusters that contain single-stroke samples of two different character classes cannot be merged. The constraints once defined at the cluster level may be imposed on only the representatives (or prototypes) of the clusters, in order to reduce both space and time requirements of subsequent clustering.

5.3.3 Constrained Stroke Clustering

The final clustering, given the stroke prototypes and the constraints, is posed as a well-established 'clustering with constraints' or 'semi-supervised clustering' problem. Constrained clustering

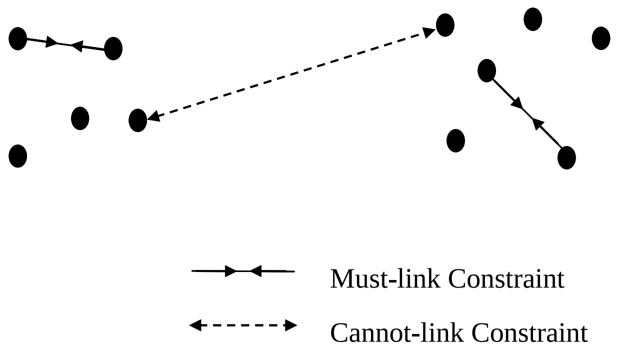


Figure 5.5: Clustering with constraints

has been an active area of research in the machine learning community over the last decade [90][91][92][93]. Unlike conventional unsupervised clustering where the data points are the only input, in constrained clustering, the clustering algorithm is also provided with two sets of constraints: *must-link* and *cannot-link*. The constraint sets contain pairs of data points as their elements and are disjoint. A must-link (ML) constraint between two data points indicates that they must be assigned to the same cluster, whereas a cannot-link (CL) constraint between them indicates the opposite (Fig. 5.5). It is apparent that every constraint generated as described in the previous section, can be treated as either an ML or CL constraint. Specification of these constraints also has other implications [94]. For instance, given four data points a, b, c and d, and the constraint sets, one can infer the following [95]:

- *Transitivity of Must-link constraints* - $ML(a, b)$ and $ML(b, c) \rightarrow ML(a, c)$
- *Implication of Cannot-link constraints* - $ML(a, b)$, $ML(c, d)$ and $CL(a, c) \rightarrow CL(a, d)$, $CL(b, c)$, $CL(b, d)$

The constrained clustering algorithm is expected to exploit the information provided in the form of pairwise constraints to discover suitable clusters. The algorithm could learn a new distance measure between the data points and/or ensure that the constraints are *maximally* respected. As one might expect, empirical results reported in the literature indicate that the quality of clusters obtained from constrained clustering is often better than the unsupervised one for various machine learning tasks [90]. In Section 5.4.4, we describe the implementation of a constrained clustering algorithm in detail.

5.3.4 Identifying Unique Writing Styles

Once each stroke is assigned to a cluster, identifying unique styles of writing a character is straightforward [89]. Each character sample is represented as a sequence of the cluster labels to which the constituent strokes are assigned, and the unique sequences across all the samples of a character class then represent unique styles of writing the character.

5.4 Finding Writing Styles in Devanagari

As mentioned earlier in Chapter 1, the characters in the Devanagari script are typically written in multiple strokes and as a result, stroke order and stroke number variations pose a severe problem for online recognition. In this section, we apply the proposed approach to characters in the Devanagari script. While the number of characters in the script is very large, for the purpose of demonstration of the proposed approach, we restrict ourselves to a set of 47 basic characters (symbols 1 to 47 in Fig. 3.1), which includes vowels (1 to 11), consonants (12 to 44) and conjuncts (45 to 47).

5.4.1 Preprocessing and Feature Extraction for Clustering

The character samples were extracted from the size-normalized word samples, described in Chapter 4, using the annotation information. Each stroke in the character sample was further re-scaled to a fixed size retaining its aspect ratio. This size normalization is necessary from the perspective of stroke clustering to increase the chances of similar-looking shapes to be assigned to the same cluster. It is important to note that even though the size of the stroke was altered, its vertical position (i.e. the y-value of the center of its bounding box) was retained. Each size-normalized stroke was also resampled to obtain a constant number of points (set to 30) in order to use the Euclidean distance for clustering.

The 7 features extracted for clustering include normalized X and Y values at each point and angle-based features such as writing direction (cosine and sine), curvature (cosine and sine) and slope (cosine), mentioned before in Section 4.4. The values of these features were also scaled to have the same dynamic range. This is important especially when one uses the Euclidean distance metric where equal weight would be given to each dimension.

5.4.2 Unsupervised Within-Character, Character-level Clustering

The first step in our multi-stage approach is character-level clustering based on the number of strokes. As described in Section 5.3.1, the samples belonging to a character class were categorized into groups based on the number of strokes each sample contains. Within each group, clustering was carried out using the AHC algorithm [96] with complete-linkage as the inter-cluster distance measure. Complete-linkage generally forms compact clusters when compared to e.g. single-linkage, which forms elongated clusters. Squared Euclidean Distance was chosen as the distance measure to compute dissimilarity between two stroke samples. In the first level of the AHC algorithm, each data point forms a cluster on its own and in subsequent levels the two most similar clusters are merged. Clustering is stopped at an appropriate level where any further merging would only result in undesirable clusters. We experimented with two different stopping criteria to determine the optimum level (or optimum number of clusters): L-method [97] and Longest Lifetime [98]. While L-method finds the knee of the plot between number of clusters and the merging distance, Longest Lifetime suggests clusters formed immediately before the largest change in the merging distance. We found that L-method found very fine clusters where even minor shape variations in strokes across character samples were distinguished. On the other hand, Longest Lifetime criterion found coarse clusters that either varied in the ordering of strokes or featured significant differences in the shapes of the constituent strokes. Since we were not concerned about the subtle variations that happen within the shape of a stroke, especially while determining the style of a character sample, Longest Lifetime criterion was adopted as the stopping criterion. Once the character clusters were obtained, within each group, character clusters were converted into stroke clusters by simply grouping the stroke samples that have the same time index within each cluster.

5.4.3 Generating Constraints

All three types of constraints described in Section 5.3.2 were generated in the context of Devanagari characters (Fig. 5.6). In particular, the third type was generated as follows. Two stroke clusters were declared similar if the similarity between their mean representatives μ_1 and μ_2 was above an empirically determined threshold. We used the similarity measure proposed by

Manor and Perona [99]:

$$\begin{aligned} similarity(cluster_1, cluster_2) &= similarity(\mu_1, \mu_2) \\ &= exp\left(\frac{-d^2(\mu_1, \mu_2)}{\sigma_1 \sigma_2}\right) \end{aligned}$$

where $\sigma_i = d(\mu_i, S_k)$ is the Euclidean distance between the mean (μ_i) and the Kth nearest neighbor (S_K) in the stroke cluster. The value of K was set to 7 in our experiment. Once it was established that certain pairs of stroke clusters present across different character clusters were similar (must-link), the cannot-link constraints were accordingly generated using the criterion specified in Section 5.3.2.

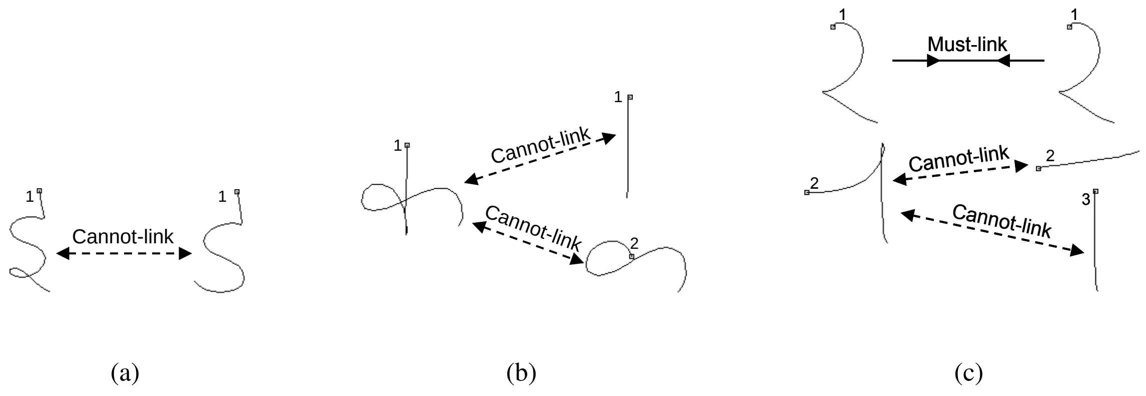


Figure 5.6: Constraint generation for Devanagari character samples: (a) Trivial across-character constraint between character 3 and character 24 (b) Trivial within-character constraint in character 12 (c) Non-trivial within-character constraint in character 43

5.4.4 Constrained Stroke Clustering

The stroke clusters formed in the first stage were merged within and between character classes using the constraints generated in the previous section. As mentioned in Section 5.3.2, instead of using all the samples of the clusters, we used the mean samples of the clusters as their representatives for subsequent clustering. Alternatively one may use the median or adopt any other technique to identify the sample(s) that would represent the cluster well. In order to construct the inter-stroke distance matrix for constrained clustering, the similarity measure between stroke samples described in the previous section, was converted to a distance measure by subtracting the value from 1. Once the distance matrix and the constraints were obtained, clustering was carried out using the Constrained Complete Link (CCL) algorithm proposed by Klein et al. [100].

CCL is a constrained or semi-supervised version of the conventional Complete Link AHC algorithm. Along with the distance matrix of the data points, the algorithm also accepts the constraint sets ML and CL. The ML constraint between a pair of data points is incorporated by setting the distance between them to zero. For the constraints to be effective, the algorithm should not only impose the constraints between the pairs of data points specified in the constraint sets but also propagate the effect to their neighborhood so that the implications of the constraints (Section 5.3.3) are experienced by the pairs of points that were not originally present in the ML or CL set. In order to propagate the ML constraints, each data point was considered as a node in a graph and constraint-imposed pairwise distances (zero between ML nodes) were used as the edge weights. Executing the ALL-PAIRS-SHORTEST-PATHS (Floyd-Warshall) algorithm on such a graph provides the shortest distance between each pair of nodes (data points) which may then be used as the modified distance measure reflecting the effect of propagation, including transitivity. After propagating the ML constraints, the CL constraints were enforced in the distance matrix by setting the distance between the corresponding pairs to the maximum value, i.e. one. However, the CL constraints were not propagated explicitly in CCL. Under Complete Linkage, the distance between two clusters is defined as the distance between the farthest neighbors in them. As a result, when a pair of data points participating in a CL constraint was present in two different clusters, the distance between those two clusters will always be maximum and would not qualify for merging. Hence, the CL constraint was implicitly propagated by CCL.

Figure 5.7 shows the plots of the number of clusters versus merging distance as constrained stroke clustering progresses. The initial and final portions of the graph correspond to merging of clusters having ML and CL pairs respectively in between them. Clustering was stopped when the merging distance exceeded an empirically determined threshold. The initial stroke clusters whose mean prototypes were assigned to the same cluster were then merged to form the final clusters of strokes. Once the final stroke clusters were determined across character classes, the unique styles of writing a character were determined as explained in Section 5.3.4.

5.4.5 Results

The proposed approach was applied to all the samples (32,192) of the 47 characters in the Devanagari dataset, and stroke clusters and writing styles were determined automatically. Table 5.1 provides some details about the clusters and styles determined.

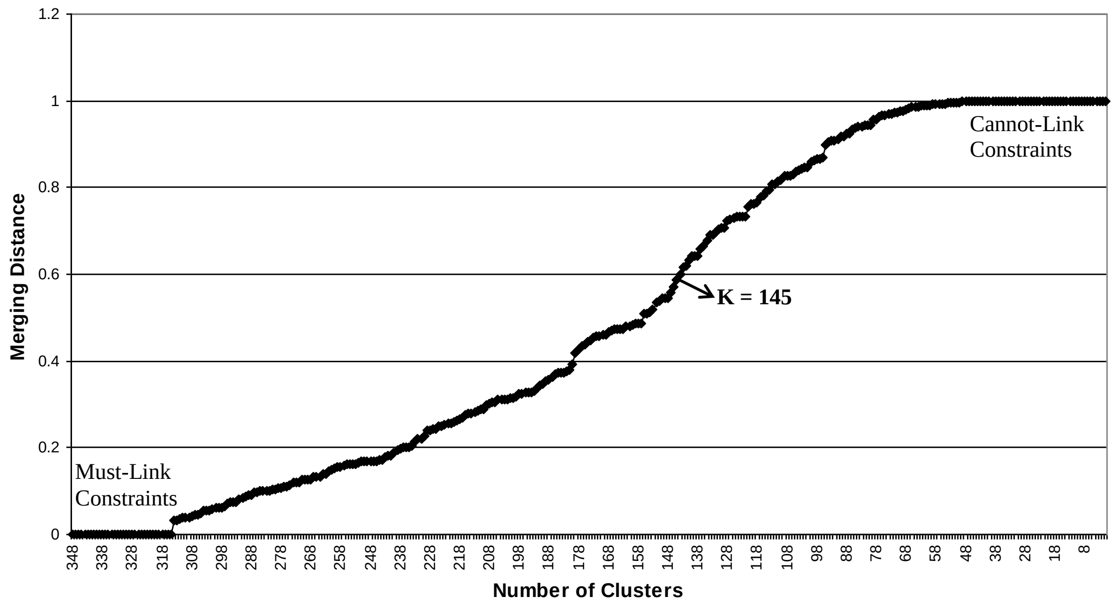


Figure 5.7: Progression of constrained clustering of stroke prototypes.

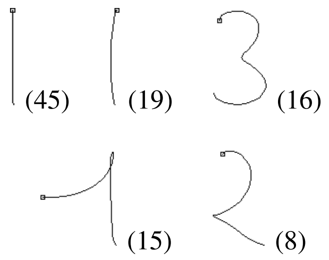
- **Stroke clusters**: A total of 149 clusters were determined by constrained stroke clustering. The vertical line '|' is the most commonly shared stroke across styles and character classes in Devanagari, and it is interesting to note that the number of times it occurs, across different styles, is as high as 45. One may note that the second most commonly shared stroke is also close in appearance to the first, indicating possible over-representation of the shape due to under-clustering in this instance.

- **Writing styles**: A total of 179 writing styles were identified from the entire dataset, an average of 4 styles per character. This provides a good indication of the style variation problem in online Devanagari. The writing styles identified by the proposed approach for certain character classes are shown in Figure 5.8. Each row corresponds to a particular style of writing the character. The stroke labels show how strokes are shared across characters. Shirorekha strokes are not a part of the character samples as they are detected and removed during word-level preprocessing.

5.5 Character Modeling using Stroke HMMs

Once the stroke clusters and styles have been identified, HMM models for the different characters may be built as described in Section 5.1.2, and illustrated in Fig. 5.2. We will henceforth

Table 5.1: Results of Constrained Stroke Clustering on the entire dataset

Number of character classes	47
Total number of character samples	32192
Total number of stroke samples	47205
Final number of stroke clusters	149
Number of styles	179
Average number of styles per character class	4
Number of strokes shared at least twice across styles	57
Most commonly shared stroke shapes with number of times it appears across styles	

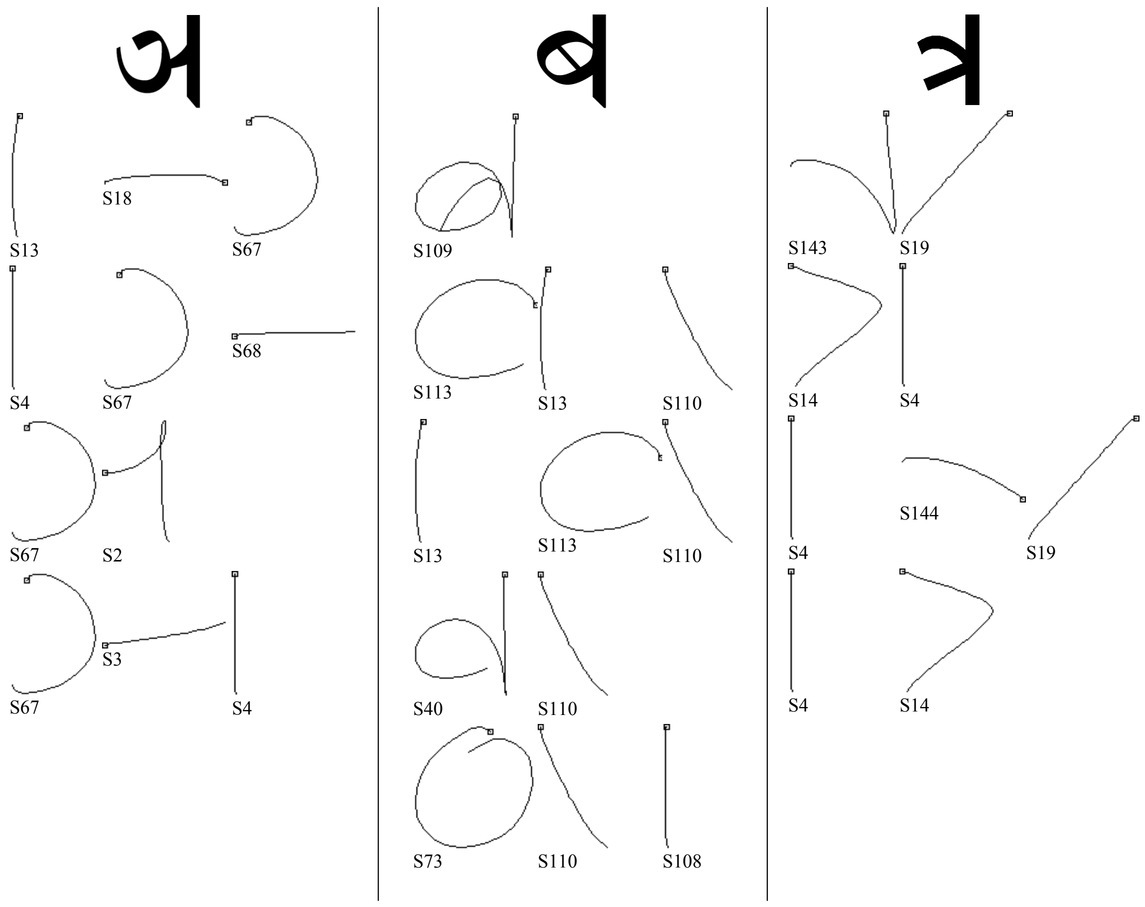


Figure 5.8: Writing styles identified by the proposed approach

refer to this approach for character modeling as CC-STROKE-HMM which denotes that the model is comprised of stroke HMMs obtained using constrained clustering.

5.5.1 HMM Training

For character modeling using HMMs, the features originally extracted from the word samples (described in Chapter 4) were used. We used the 'strictly left-to-right' HMM topology (Fig. 2.3) for modeling samples assigned to a cluster. The number of Gaussians per state (G) was fixed to 3. The number of states per HMM was computed as a fraction of the average number of points (an indicator of shape complexity) present in all the samples of the cluster. This has been shown to be better than having a fixed number of states for all the classes [20]. The training of the HMM was carried out using the Baum-Welch procedure.

5.6 Experimental Evaluation

The proposed approach (CC-STROKE-HMM) for character modeling was evaluated with respect to recognition accuracy and reduction in the number of states needed for HMM modeling, and compared with alternate modeling techniques. The alternate models we evaluated are described below.

1. *Single HMM per character (SINGLE-HMM)* - This may be considered as the simplest way to model a handwritten character. All the samples of a character class were utilized to train a single HMM to represent the class.

2. *Multiple character HMMs per character using first-stage character clusters (US-FIRST-STAGE-CHAR-HMM)* - In this modeling technique, the samples assigned to each cluster, obtained by the procedure described in Section 5.4.2, were used to model a 'subclass HMM' or 'allograph-HMM' [80], and the subclass HMMs were connected in parallel to model the character class. The key difference between the approach followed here, and our previous work [101] is that in the latter, each character sample was resampled to a fixed number of points and all the samples of a character class were clustered together without pre-categorization based on their stroke number.

3. *Multiple style HMMs per character using unsupervised stroke-level clustering (US-STROKE-HMM)* - In this approach, samples of strokes from different character classes were clustered together in an unsupervised manner. Clustering all stroke samples together demands substantial memory and computation time. Hence, we adopted a two-stage approach for unsupervised stroke clustering. In the first stage, the stroke samples belonging to each character class were clustered independently, and subsequently in the second stage, the mean prototypes of the first-stage clusters alone were further clustered across character classes to get a reduced set of stroke clusters. While the AHC algorithm and the distance measure specified in Section 5.4.4 were used for both stages, the stopping criteria used for the first and second stages were Longest Lifetime [98] and fixed number of clusters, respectively. For purposes of comparison, the number of clusters for the second stage was fixed to the same number as that used by the CC-STROKE-HMM approach.

   Once the stroke clusters were formed, unique writing styles of a character class were determined as described in Section 5.3.4. The samples of these clusters were used to train left-to-right *stroke HMMs* and connected according to the writing style sequence to form *style HMMs*. The different style HMMs built for a character class were then connected in parallel to form the final character model.

5.6.1 Evaluation Methodology

While samples of some characters were present in large numbers in the dataset based on their frequency in the words collected, there were 685 samples per character class on an average. In order to evaluate the recognition performance of our approach, the dataset was partitioned into 10 groups based on the writer IDs such that each group had the same number of writers. This allowed the definition of 10 folds wherein each fold specified one group of writers for testing, and the remaining 9 groups for training. Since the Devanagari dataset contains word samples from 110 writers, the first fold specified writers with IDs 0 to 10 for testing and IDs 11 to 109 for training. Since our objective was to evaluate writer-independent recognition, no writer was present in both the training and test sets. In all the experiments reported in this subsection, the accuracy was computed as the average over the 10 folds.

For all of the alternative models described above, HMM training was carried out as described in Section 5.5.1 for CC-STROKE-HMM. However, for the SINGLE-HMM approach, given that a single HMM had to model all the styles, the number of Gaussians per state (G) was set to 5 since it returned the highest accuracy over the 10 folds (Fig. 5.9).

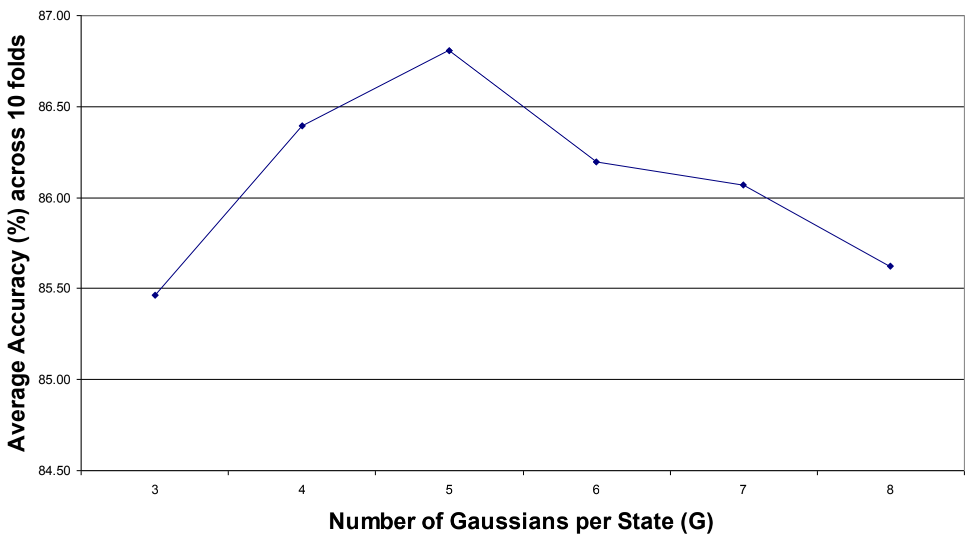


Figure 5.9: Effect of number of Gaussians per state on the SINGLE-HMM accuracy

5.6.2 Results and Discussion

The character recognition performance of the four different approaches of modeling using HMM, for each fold, is shown in Figure 5.10, and the average accuracy across all 10 folds is shown in Table 5.2. In terms of purely recognition accuracy, US-FIRST-STAGE-CHAR-HMM performs best (91.38%), followed by CC-STROKE-HMM (90.74%), SINGLE-HMM (86.81%) and US-STROKE-HMM (81.62%).

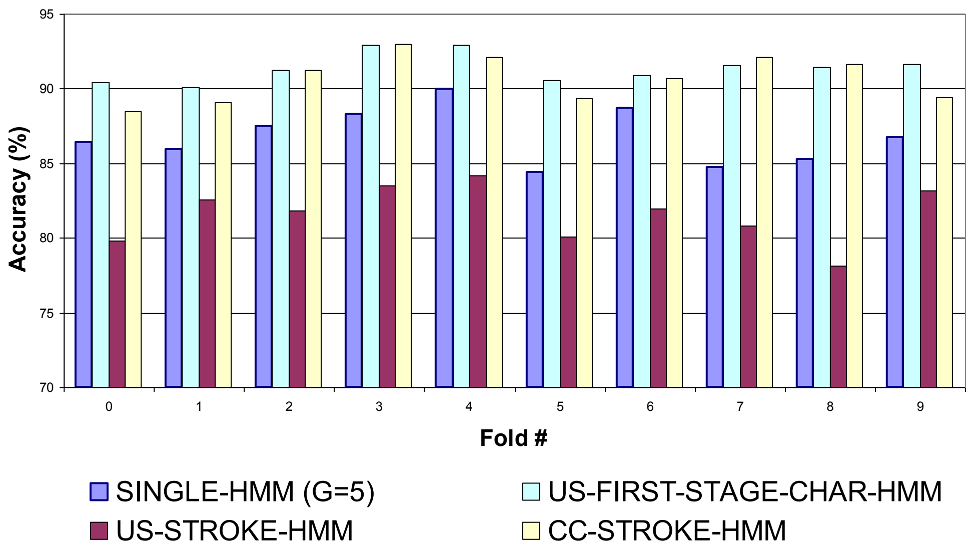


Figure 5.10: Devanagari character recognition accuracy for different approaches and dataset folds

Table 5.2: Devanagari character recognition accuracy and total number of states to train, averaged across 10 folds

Approach	**Accuracy**	**Number of States**
SINGLE-HMM	86.81%	901
US-FIRST-STAGE-CHAR-HMM	91.38%	4359
US-STROKE-HMM	81.62%	2563
CC-STROKE-HMM	90.74%	2440

The first observation is that the accuracy of CC-STROKE-HMM is marginally lower than the US-FIRST-STAGE-CHAR-HMM. This may be explained as follows. After the first stage character-level clusters are formed, the second stage only attempts to merge the similar-looking stroke clusters in order to obtain a reduced stroke set. Since reduction in the number of stroke clusters does not guarantee improvement in recognition [20], one may expect the accuracy of US-FIRST-STAGE-CHAR-HMM to be an upper bound on that of CC-STROKE-HMM. However, it is important to note that while the two accuracies are comparable, the number of states to train in CC-STROKE-HMM is nearly half of that of US-FIRST-STAGE-CHAR-HMM, which translates to significant savings in terms of memory and computation time during recognition. A second, somewhat surprising observation is that the accuracy of US-STROKE-HMM is substantially lower than even SINGLE-HMM. This may be attributed largely to the errors in stroke clustering, and reiterates the importance of using handwriting domain information such as the number of strokes in a character and the sub-collection rule while clustering strokes. Finally, it is not surprising that the performance of US-FIRST-STAGE-CHAR-HMM and CC-STROKE-HMM are significantly better than that of the SINGLE-HMM, due to the reasons mentioned earlier (Section 5.1).

5.7 Summary

We proposed the use of stroke-based modeling for HMM modeling of symbols in Indic scripts. The symbol model features style HMMs connected in parallel. The style HMMs are in turn composed from stroke HMMs that are trained from stroke clusters identified in the data, and shared across character styles. The proposed stroke-based modeling in turn uses a novel approach based on Constrained Stroke Clustering for identifying the optimal set of strokes and styles given the character samples. The technique is completely automatic and script-independent and hence can potentially serve as a generic framework for style identification using stroke clustering. We introduced some basic constraints, inherently present in the handwriting domain, that one can apply to obtain better stroke clusters. In order to cluster strokes under constraints, we adopted the Constrained Complete Link algorithm proposed in the Machine Learning domain. The efficacy of the proposed approach was evaluated for Devanagari characters by measuring its impact on recognition accuracy and the number of states required for HMM modeling. The symbol models built using the proposed approach were found to outperform alternate approaches commonly found in the literature. In the next chapter, we describe how these symbol models are used for the recognition of words.

Chapter 6

Word Recognition using Lexicon-driven and Lexicon-free Approaches

6.1 Introduction

The HMM-based approaches for OHWR presented as part of the literature review indicate a few significant aspects. In the case of Latin script recognition, where the main challenge is segmentation of cursive writing, HMMs are built at the character level and then concatenated to form word-HMMs. The word-HMM implicitly segments the input into characters as a by-product of recognition. On the other hand, for CJK script recognition, HMMs are often built at the stroke level and these models are connected to form a large network that represents different stroke orders and/or imposes syntactic constraints. We believe that both the strategies are relevant for Indic scripts, since both segmentation issues as well as stroke order/number variations are present.

In this chapter, we propose data-driven and script-independent approaches using HMMs, for recognizing handwritten words, which need minimal manual intervention during training. We explore two different recognition strategies: lexicon-driven and lexicon-free. The lexicon-driven technique models each word in the lexicon as a sequence of symbol HMMs according to a standard writing order. On the other hand, our lexicon-free technique uses a novel Bag-of-Symbols representation to deal with the symbol order variation problem and for rapid matching of the lexicon. We also evaluate the performance of our techniques with Devanagari and Tamil word samples, through a series of experiments.

6.2 Symbol Modeling using HMM

The stroke-based modeling approach using constrained stroke clustering (CC-STROKE-HMM) described in the previous chapter was used to model the symbols in Devanagari (Fig. 3.1) as it was shown to be more accurate than using a single HMM per class. However, in the case of Tamil, where each symbol (Fig. 3.2) is typically written in a single stroke, we noticed only a marginal improvement in terms of recognition accuracy when compared to using a single HMM per symbol. Therefore, for Tamil, each symbol is modeled simply using a single HMM.

6.3 Lexicon-driven Word Recognition

In this Section, we describe a lexicon-driven approach for recognition of words, given the symbol models described earlier. In a lexicon-driven approach for word recognition, the lexicon is used to aid the segmentation-recognition process by exploring only valid symbol sequences. First, the input handwritten word is matched against each lexicon word model, considering several segmentation alternatives. The optimum segmentation for a given lexicon word is determined by minimizing the matching cost using approaches such as Dynamic Programming or Viterbi decoding [69, 102, 103]. Then, the words in the lexicon are sorted based on the minimal matching cost and the word with the least cost is declared as the recognition result.

6.3.1 Word and Lexicon Modeling using HMM

When HMMs are used for lexicon-driven word recognition, a word model may be constructed by simply concatenating the constituent symbol HMMs described in the previous section. Even though such a word model may be appropriate for Latin scripts [69, 20], it suffers from a serious limitation in the context of Indic scripts - variability in symbol order. Given the UNICODE representation of a word from the lexicon, one will have to assume a particular order of symbols in order to construct the word-HMM, which may not always be the one used in the input sample.

Based on our knowledge about the script and manual investigation of the collected handwriting samples, we found that people wrote the consonant first followed by the matra, barring a few exceptions. Such an order is also observed in the study conducted by Vaid and Gupta [104] for Devanagari. Therefore, we adopted the phonetic order of the symbols as the 'standard symbol order' while creating word-HMMs from the UNICODE strings. However, we took into

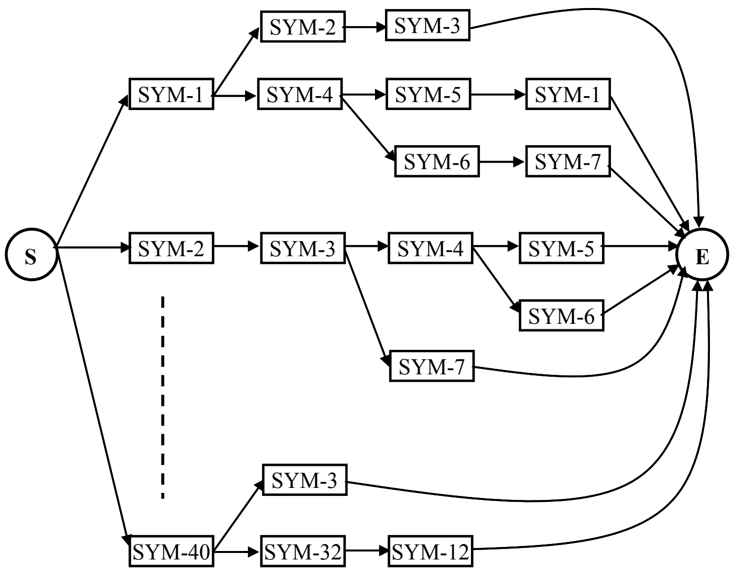


Figure 6.1: Lexicon-driven recognition using prefix-tree representation. Each box represents a symbol HMM. 'S' and 'E' are start and end nodes.

account certain exceptions such as symbols 78, 79 and 80 in Tamil, which are matras that are almost always written before the consonant, and symbol 86 in Devanagari which phonetically comes before the consonant sound but is typically written after. Once the word models were constructed, the lexicon was represented in the form a *prefix-tree* (Fig. 6.1) where the symbol models that are part of the common prefixes are shared across word models [69]. The prefix-tree structure is known to reduce both memory and time requirements during recognition.

Given the features corresponding to the input handwritten word, the Viterbi algorithm [105] was executed on the prefix-tree to determine the path with the maximum likelihood. The sequence of symbols that form the maximum-likelihood path is then considered as the recognized word.

6.3.2 Impact of Non-standard Symbol Orders

The imposition of standard symbol order to model a word may be considered as a major limitation of the lexicon-driven approach. As mentioned earlier, even though the phonetic order of writing is common, there may be situations where one writes the symbols out of order. In our dataset, when we considered word samples where symbols are written one after another with a pen-up in between consecutive symbols, we found that approximately 7% of the word samples in the case of Devanagari and 1% in the case of Tamil were written in a symbol order different from the standard order. The symbol order variations can broadly be classified into two categories:

1. *Symbol order variations within a character* - The symbols (e.g. C and V) that constitute a single character are written in different orders by different writers or even by the same writer at different times. Table 6.1 shows the symbol orders observed for certain characters in our Devanagari dataset. The difference in the spatial and spoken order when small /i/ matra (symbol 49 in Fig. 3.1) is involved has also been studied using a Hindi dictation experiment by Vaid and Gupta [104]. While the matra is written to the left of the consonant, it is pronounced after the consonant in the phonetic sequence. The authors conclude that such a discrepancy incurs a certain processing cost while reading the word.

2. *Symbol order variations across characters* - A symbol that is part of one character is written after writing one or more symbols of other characters in the word. This is similar to the notion of delayed strokes in English such as t-crossing, i-dot and j-dot, and may be called as a *delayed symbol* in the context of Indic scripts. Table 6.2 shows some examples from the Devanagari word dataset where symbol order variations occur across characters. The delayed symbols are typically vowel modifiers (M) such as *chandrabindu* (symbol 60) and *nukta* (symbol 63), which are mostly contained in the upper or lower zones and not part of the body of the word.

It is important to note that it is difficult in practice to learn these variations during training without access to very large datasets of handwritten words. Given the impact of symbol order variability, we also explored a lexicon-free approach, which is described in the next section.

6.4 Lexicon-free Word Recognition

In contrast to the lexicon-driven approach described in the previous section, in a lexicon-free approach, the lexicon is not used to drive the segmentation and recognition of the input word. Instead, the lexicon is used as a post-processing tool, following recognition, to fix errors if any in the recognition result. Our lexicon-free recognition strategy is based on two stages, as described below.

Table 6.1: Within-character symbol order variations observed in the word dataset

Scenario	**Character**	**Standard Writing Order**	**Observed Order(s)**
/i/ matra is present	वि	व + ि	ि + व
	पि	प + ि	ि + प
	रि	र + ि	ि + र
Half-consonant is present	ल्ला	ल् + ल + ा	ल + ल् + ा
	त्य	त् + य	य + त्
Two or more matras are spatially close	फ्रैं	फ + ़ + ⁄ + ै + ं	फ + ⁄ + ़ + ै + ं
			फ + ⁄ + ै + ं + ़
			फ + ै + ं + ⁄ + ़
			फ + ै + ं + ़ + ⁄

6.4.1 Arbitrary Symbol Sequence Recognition using HMM

The first stage of lexicon-free recognition is a word recognizer capable of recognizing any arbitrary symbol sequence. We have adopted the "recurrent HMM" architecture [106, 22] where the symbol models are connected in parallel to each other as shown in Fig. 6.2 and there is a recurrent loop back from the final state to the start state. Therefore, in theory, such a topology can recognize any word without the help of a lexicon. However, in practice, the accuracy of such a recognizer is poor due to absence of context for pruning invalid paths during the Viterbi search.

Table 6.2: Across-character symbol order variations observed in the word dataset

Word	Standard Writing Order	Observed Order
ख़्वाब	ख+़+व+ा+ब	ख+व+ा+ब+़
काग़ज़	क+ा+ग+़+ज+़	क+ा+ग+ज+़+़
साँप	स+ा+ँ+प	स+ा+प+ँ
बॉल	ब+ा+ॅ+ल	ब+ा+ल+ॅ
अंतरिक्ष	अ+ं+त+र+ि+क्ष	अ+त+र+ि+क्ष+ं

The recognized string from the recurrent HMM is matched against the words in the lexicon and the closest match is then declared as the final recognition result. In order to improve recognition, we considered more than one choices from the recurrent HMM.

6.4.2 Bag-of-Symbols Representation

When matching the output word of the recurrent HMM and the words in the lexicon, it is important to take into account the genuine symbol order variations in the writing. Conventional matching schemes based on Edit distance [96] will result in a low matching score when the two words have different symbol orders. Therefore, in order to avoid penalizing genuine symbol order variations, we use a symbol-order-free representation of the word, which we refer to as the *Bag-of-Symbols* representation, along the lines of the popular Bag-of-Words (BoW) model in the information retrieval literature [107], which has also been recently extended for tasks such as Content Based Image Retrieval [108] and object recognition [109]. In the BoW model, a document is represented as a vector where each element in the vector contains the number of times a word occurs in the document. Thus, the representation of a document does not capture the order in which the words occur. Similarly, our BoS representation of a word, contains

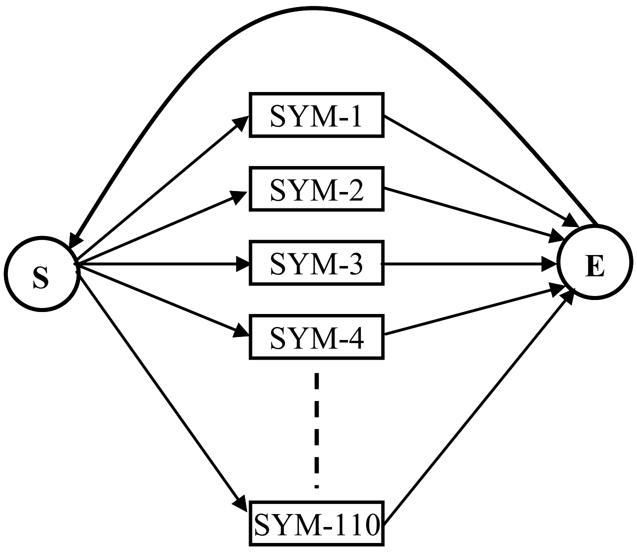


Figure 6.2: Lexicon-free recognition using recurrent HMM.

श्रीमती

Word

104_50_36_27_50

Standard Symbol Order

Symbol ID	*1*	...	*27*	...	*36*	...	*50*	...	*104*	...	*110*
Count	0	...	1	...	1	...	2	...	1	...	0

Bag of Symbols

Figure 6.3: BoS representation

only the counts of symbols in the word, and ignores their order. Figure 6.3 illustrates the BoS representation for a Devanagari word.

Formally, the BoS representation of a given word **x**, is a vector x of N dimensions corresponding to the N symbols in the script. x_i indicates the number of times symbol i is present in the word **x**. The distance between two words when represented as BoS is defined as:

$$Dist(x, y) = \frac{\sum_{i=0}^{N-1} |x_i - y_i| + |\sum_{i=0}^{N-1} x_i - \sum_{i=0}^{N-1} y_i|}{2}$$

The above distance measure ensures that the cost of substitution, deletion or insertion of a symbol, when matching two words, is one.

We studied the feasibility of the BoS representation for Devanagari and Tamil by determining the frequencies of number of words that have the same BoS representation when the symbol order is disregarded. Table 6.3 provides the frequencies for two Devanagari lexicons containing 20K and 90K words each. In the lexicon of 20K words we find that 18,512 (92.56%) words have unique BoS representations and even in a lexicon of 90K words, the maximum number of words having the same BoS representation is only 7 with frequency 2. Similarly in the case of Tamil (Table 6.4), 97.5% of words in the 20K and 95.65% in the 90K have unique BoS representations. Thus the identities of the symbols as captured by the BoS representation can be used for matching the words in the lexicon. Representing the sequence or order of symbols is not valuable except in the rare event of a collision. The BoS representation also supports efficient matching since unlike distance measures such as the Edit [96] which requires $(m \times n)$ computations, where m and n are the lengths of the two strings, two BoS representations can be compared using a maximum of $(m + n)$ computations. Hence, the BoS representation permits one to find potential word matches from the lexicon very rapidly. In the cases where collisions occur, one may employ costlier matching schemes.

6.4.3 Lexicon Matching

In this subsection, we provide a simple algorithm (Algorithm 2) for matching the lexicon entries when words are represented using BoS. The algorithm takes the top R choices obtained as output from the recurrent HMM, and for each choice, finds the top B matching words in the lexicon using the distance measure described in the previous section. A lexicon word is awarded a vote when it is one of the top B lexicon choices and as a result, a lexicon word can acquire a maximum of R votes corresponding to the R choices from the recurrent HMM. Finally, the F word choices with most votes are extracted from the lexicon. Ties between words having equal votes are broken by computing the mean edit distance with the R input choices. Finally, the number of votes for each word is normalized by dividing with the maximum value (R) to obtain a confidence score.

Algorithm 2: Pseudo-code for lexicon matching using BoS

```
Input: $B$, $F$, list $H$ of $R$ word choices from recurrent HMM, and lexicon $L$
Output: Word choices with confidence scores
 1: $Votes \leftarrow 0, Conf \leftarrow 0$
 2: for each word $\mathbf{x}$ in $H$ do
 3:     for each word $\mathbf{y}$ in $L$ do
 4:         $x \leftarrow BoS(\mathbf{x})$
 5:         $y \leftarrow BoS(\mathbf{y})$
 6:         $DistArr[\mathbf{y}] = Dist(x, y)$
 7:     end for
 8:     Sort $DistArr$ in ascending order and set top $B$ corresponding words to
        $ShortListedWords$
 9:     for each word $\mathbf{z}$ in $ShortListedWords$ do
10:         $Votes[\mathbf{z}] = Votes[\mathbf{z}] + 1$
11:     end for
12: end for
13: Sort $Votes$ in descending order and set top $F$ words to $FinalWordChoices$
    /* Break ties, if any, in $FinalWordChoices$ by computing average Edit distance to
    words in $H$ */
14: for each word $\mathbf{c}$ in $FinalWordChoices$ do
15:     $Conf[\mathbf{c}] = Votes[\mathbf{c}]/R$
16: end for
17: return $FinalWordChoices, Conf$
```

Table 6.3: Frequencies of number of words sharing the same BoS representation in the Devanagari lexicon

No. of words	**Frequencies in the lexicon with size**		**Example**
	20K	90K	
1	18512	80540	लक्ष्य
2	605	3714	खेल लेख
3	82	506	जाती जीता ताजी
4	8	103	राम मार मरा रमा
5	-	14	पटेल लपेट लपटे पलेट पलटे
6	-	3	नामक मानक मकान कमान मनका मकना
7	-	2	बाहर बारह बहार बहरा बराह राहब रहाब

6.5 Experimental Evaluation

We evaluated our approach, and the lexicon-driven and lexicon-free strategies described in the previous sections using the Devanagari and Tamil datasets described in Chapter 3. In this section, we describe our evaluation methodology, the experiments carried out, and the main findings.

Table 6.4: Frequencies of number of words sharing the same BoS representation in the Tamil lexicon

No. of words	**Frequencies in the lexicon with size**		**Example**
	20K	90K	
1	19498	86092	கண்டு
2	226	1664	மகள் களம்
3	14	143	சம்பா பாசம் சாபம்
4	2	31	விசா சிவா சாவி வாசி
5	-	3	காலம் கலாம் காமல் மகால் கமால்
6	-	2	தரம் தர்ம தமர் ரதம் மரத் மதர்

6.5.1 Analysis of Writing Styles

Based on manual analysis, we decided to categorize the style of writing into three types, representing progressively fewer constraints and greater complexity for recognition. In a type-1 word sample, there is always a pen-up between any two consecutive symbols in the word and thus may be considered as a 'discrete' form of writing. In a sample of type-2 style, there is at least a pair of symbols in the word which is not separated by a pen-up i.e. the two symbols are written 'cursively'. The type-3 category of word samples has at least one delayed stroke and may contain type-2 writing as well. Therefore, type-3 style may be considered as the most unconstrained form of writing. Figure 6.4 shows samples of the three types from Devanagari. Table 6.5 indicates the number of word samples (excluding the ones with one or more symbols

Table 6.5: Distribution of word samples in the datasets

Style	Devanagari	Tamil
Type-1	9407	4801
Type-2	895	1671
Type-3	1213	35

tagged as bad quality) belonging to each style found in the Devanagari and Tamil datasets. One may note that type-1 style of writing is the most popular form of writing for both Devanagari and Tamil. Delayed strokes rarely occur while writing Tamil and hence the type-3 samples were not considered during evaluation.

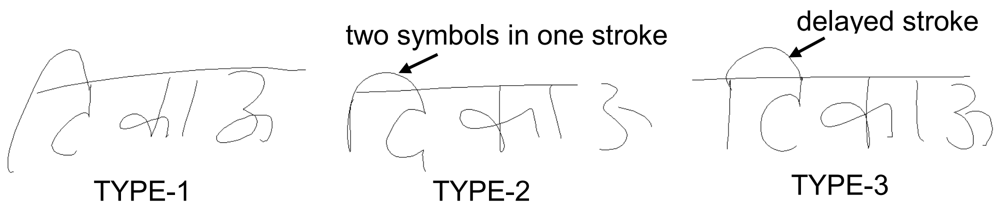


Figure 6.4: Categorization of word samples based on style of writing. Examples from the Devanagari word dataset.

6.5.2 Evaluation Methodology

For the purpose of evaluating the word recognition performance, the dataset was categorized into 10 folds as described in Section 5.6.1. While the first fold of the dataset with type-1 style samples in the test set (hereafter referred to as "validation fold"), was used to learn the recognizer's parameters such as beam width, the recognition accuracy is reported as an average across the other 9 folds in all the following experiments.

Evaluation of the word recognition system was carried out on different lexicon sizes such as 1K, 2K, 5K, 10K and 20K. The lexicons were created using the EMILLE-CIIL text corpus [64] that contains news articles extracted from online news portals. An N-word lexicon contained the N most frequently occurring words in the corpus. Since the words in the corpus were in UNICODE encoding, they had to be converted into sequences of symbol IDs using the standard order as described in Section 6.3.

6.5.3 Lexicon-driven Word Recognition

As described in Section 6.3, in a lexicon-driven approach, the standard symbol order is imposed while modeling the words as HMMs and the lexicon is represented as a prefix-tree. An important parameter that has profound impact on recognition is the Beam Width for Viterbi decoding on the prefix tree. It serves as the central control parameter for pruning the search paths in the HMM network. Higher beam width allows more paths to be explored, whereas lower beam width prunes them aggressively. As mentioned earlier, the validation fold from both Devanagari and Tamil datasets was used for learning the beam width. Beam widths of 300 and 650 were found to be optimal from the standpoint of trading off recognition accuracy and time.

Tables 6.6 and 6.7 present the average recognition accuracies obtained across 9 folds for the two scripts and different styles of writing. In the case of Devanagari, across all lexicon sizes, the accuracies of type-2 and type-3 styles are substantially lower than that of type-1 style. This may be because of insufficient modeling. For instance, the delayed strokes in type-3 and ligatures (connecting parts) in type-2 are not explicitly modeled as done by Brown et al. [20]. Also, in addition to that, the samples belonging to all the three styles may also contain symbol orders different from the standard order. However, one may note that the accuracy of type-1 style of Tamil writing (92.48% for 20K) is substantially higher when compared to that of Devanagari (83.82%). This may be due to lesser symbol order variations in Tamil writing. Overall recognition accuracies of 79.01% and 91.80% are obtained for Devanagari and Tamil respectively with their lexicons containing 20K words. When the 20K-word lexicon was used, the average time taken for recognizing a word sample for both the scripts was close to 400 milliseconds, on a laptop with the Intel Core2 Duo CPU @ 2 GHz and a RAM of 2 GB.

6.5.4 Lexicon-free Word Recognition

The first stage of the lexicon-free strategy is the recognition of the word sample using the recurrent HMM. During training, in addition to the Beam Width, an optimal value for inter-model transition penalty was also determined for the recurrent HMM. The penalty is a constant added to the total log-likelihood during transition from one symbol model to another. Therefore a large positive penalty will result in over-segmentation whereas a large negative value will result in under-segmentation. The two parameters were found using the validation fold. For Devanagari, a beam width of 250 and an insertion penalty of -75 were found to be optimal.

Table 6.6: Average accuracy (%) across 9-folds for Devanagari word recognition

	1K		**2K**		**5K**		**10K**		**20K**	
Style	**Top1**	**Top5**	**Top1**	**Top5**	**Top1**	**Top5**	**Top1**	**Top5**	**Top1**	**Top5**
Lexicon-driven										
TYPE-1	92.01	93.48	90.33	92.56	88.24	91.59	85.97	90.87	83.82	90.22
TYPE-2	68.99	70.87	66.04	70.17	61.91	67.57	60.61	66.27	57.55	64.39
TYPE-3	72.50	74.93	69.45	73.02	65.54	71.19	62.92	68.67	59.53	67.01
All Samples	87.99	89.60	86.06	88.59	83.59	87.39	81.37	86.42	79.01	85.56
Lexicon-free										
TYPE-1	87.06	96.06	86.44	95.26	84.65	93.71	83.03	92.98	81.49	91.92
TYPE-2	78.18	92.57	77.12	91.04	75.47	89.27	73.35	86.44	70.52	83.84
TYPE-3	76.33	92.60	75.46	90.95	71.98	87.73	69.45	84.51	66.32	81.90
All Samples	85.16	95.40	84.48	94.44	82.51	92.69	80.75	91.51	78.92	90.16
Combination of lexicon-driven and lexicon-free										
TYPE-1	95.42	98.70	94.37	98.19	92.40	97.59	90.63	96.95	89.29	96.31
TYPE-2	83.84	97.05	83.49	96.34	82.08	95.52	81.60	94.34	80.19	92.69
TYPE-3	85.47	95.47	84.07	94.43	82.07	92.78	78.24	91.47	76.41	89.73
All Samples	93.38	98.21	92.35	97.62	90.42	96.89	88.53	96.14	87.13	95.29

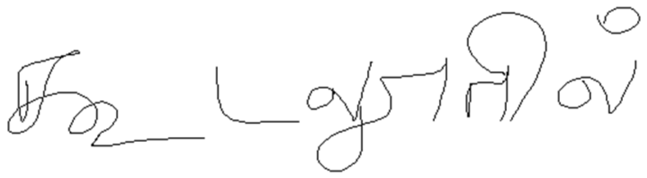
Table 6.7: Average accuracy (%) across 9-folds for Tamil word recognition

	1K		**2K**		**5K**		**10K**		**20K**	
Style	**Top1**	**Top5**	**Top1**	**Top5**	**Top1**	**Top5**	**Top1**	**Top5**	**Top1**	**Top5**
Lexicon-driven										
TYPE-1	96.46	97.96	95.40	97.78	94.53	97.42	93.51	97.18	92.48	96.68
TYPE-2	94.79	97.42	93.47	97.05	92.21	96.42	90.83	95.73	89.82	95.29
All Samples	96.03	97.82	94.90	97.60	93.93	97.16	92.82	96.80	91.80	96.32
Lexicon-free										
TYPE-1	82.88	92.98	82.21	91.94	80.65	89.88	79.39	88.25	77.76	86.73
TYPE-2	82.54	93.84	81.91	93.22	81.16	91.46	80.03	90.45	77.70	89.26
All Samples	82.80	93.21	82.13	92.27	80.78	90.28	79.55	88.82	77.74	87.38
Combination of lexicon-driven and lexicon-free										
TYPE-1	93.90	98.39	93.09	98.11	91.85	97.42	90.53	96.66	88.94	95.87
TYPE-2	93.03	98.74	92.46	98.43	91.58	97.42	89.89	97.17	88.69	96.55
All Samples	93.67	98.48	92.93	98.19	91.79	97.42	90.36	96.79	88.88	96.05

Similarly, in the case of Tamil, the values were found to be 350 and -45 for beam width and insertion penalty respectively. Top 20 word choices from the recurrent HMM were passed to the lexicon matching stage (Algorithm 2) that uses the BoS representation for symbol-order-free ranking of words.

Tables 6.6 and 6.7 show the performance of the lexicon-free approach for Devanagari and Tamil respectively. The overall accuracy across all the styles and with the 20K lexicon is 78.92% and 77.74% for Devanagari and Tamil respectively. In the case of Devanagari (Table 6.6), when compared to the lexicon-driven approach (83.82% for 20K), the accuracy of type-1 style samples (81.49%) is lower. However, the performance with type-2 and type-3 styles is substantially higher and is found to be 70.52% and 66.32% respectively. The higher performance may be attributed to two important factors: (i) In contrast to lexicon-driven HMM, the recurrent HMM is context-independent [22] and attempts to spot best-matching individual symbols in the word sample. As a result, even when a ligature or a delayed stroke is present in the word sample, it may identify the other well-written discrete symbols correctly. (ii) The recognition scheme is independent of the symbol order and hence, unlike the lexicon-driven scheme, it can potentially recognize any symbol order. However, in the case of Tamil (Table 6.7), the performance of the lexicon-free recognizer is found to be substantially lower than the lexicon-driven approach. Our analysis reveals that the Tamil symbol set contains many similar-looking symbols and as a result, the same context-independent symbol-order-free nature of the recurrent HMM, which was advantageous for Devanagari, is not effective for Tamil. Figure 6.5 shows a Tamil word sample which was misrecognized by the lexicon-free approach. One may note that the symbol 23 which is a consonant (Fig. 3.2) is written identical to symbol 73 (matra). This is the most common style of writing the symbol 23 when it is part of the character containing matras 72, 74 or 75. Therefore, the character context is needed to distinguish the two symbols. From the example, one also finds that segmentation errors, e.g. symbol 54 being over-segmented as two different symbols 12 and 4 based on shape similarity, also occur in the absence of context. As a result, the top-20 accuracy of the recurrent HMM, averaged across 9 folds, was found to be only 54.6%. Such errors in many cases are irrecoverable using the lexicon in the second stage.

The average time taken for recognizing a word sample was 430 milliseconds (320 ms for recurrent HMM recognition and 110 ms for BoS-based matching of 20K-word lexicon) for Devanagari and 550 milliseconds (380 ms + 170 ms) for Tamil.



Truth:	கூடலூரில் 54_16_66_23_74_24_72
Recurrent HMM Choice:	கஉடலூ◌ா◌িலீ 12_4_16_66_73_74_24_75
Lexicon-free Choice:	கடலூரில் 12_16_66_23_74_24_72

Figure 6.5: Error in lexicon-free Tamil word recognition

6.5.5 Combination of Lexicon-driven and Lexicon-free Approaches

While the lexicon-driven approach was found to be more accurate for type-1 style samples written in the standard symbol order, the lexicon-free approach was suitable for type-2 and type-3 styles, especially when written in a non-standard symbol order. In order to get the best of both; we also explored the possibility of combining the two recognizers. We adopted a simple parallel combination scheme for combining their results, using top 5 choices from the lexicon-driven recognizer and top 20 choices from the lexicon-free approach. When only one of the recognizers was confident, the combination scheme returned the results of the corresponding recognizer. The lexicon-driven recognizer was assumed to be confident of its result when there was only a single output choice after Viterbi decoding of the input word sample. The other competing paths might have been pruned due to their low scores. Similarly, the lexicon-free recognizer was assumed to be confident of its result if the difference in the confidence scores of the first and second choice was greater than a predefined threshold. On the other hand, when both the recognizers were confident or when both were not, we combined their individual choices using the Borda count [110]. In the case of a tie in Borda count between two words having the same number of votes, the number of votes from the lexicon-free recognizer was considered to break it. Tables 6.6 and 6.7 show the results obtained by the combination for Devanagari and Tamil for various lexicon sizes. In the case of Devanagari, with the 20K-word lexicon, top-1 accuracy of 87.13% and top-5 of 95.29% were obtained. It is interesting to note that there is a substantial improvement in accuracy when compared to lexicon-driven (Fig. 6.6) and lexicon-free approaches alone. The improvement is as high as 22.64% and 16.88% for type-2 and type-3 styles respectively. An overall accuracy improvement of 8.12% was achieved

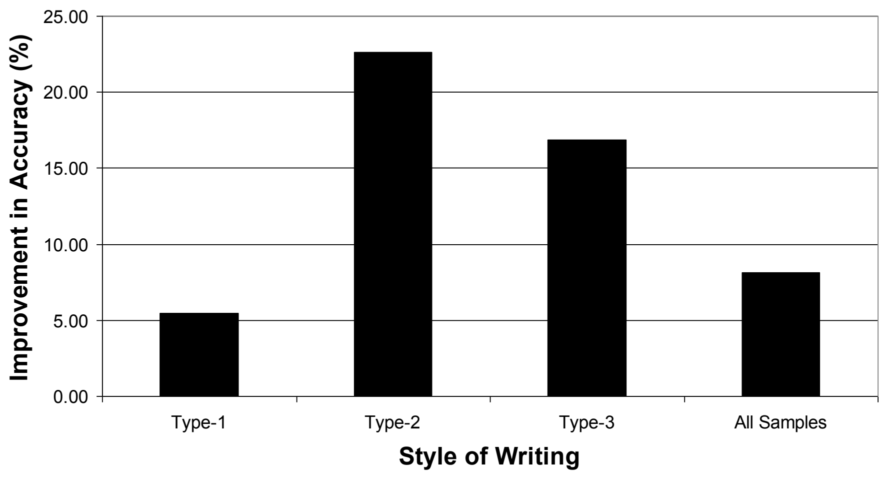


Figure 6.6: Accuracy improvements achieved by the combination over lexicon-driven for Devanagari word recognition with 20K lexicon

across the three styles, which may considered as substantial.

In the case of Tamil, it may not be surprising to note that there is actually a decrease in accuracy due to combination, when compared to the lexicon-driven recognizer. This may be attributed due to the low performance of the lexicon-free recognizer due to the reasons described in the previous section.

6.5.6 Performance with Samples having Symbol Order Variations

Besides computing the overall accuracy as explained in the previous section, the performance of each technique was also evaluated with those type-1 style word samples in Devanagari that had symbol order variations. In total, 592 type-1 samples in the Devanagari dataset had symbol orders different from the standard. Along with lexicon-driven, lexicon-free and their combination, we also computed the accuracy of an *oracle lexicon-driven* recognizer. Instead of using the standard symbol order of a word for the lexicon, the oracle recognizer used the exact symbol order in which the writer had written the word. This is achieved by dynamically adding the actual symbol order of the test word sample, determined from the annotation, to the list of words in the lexicon before recognizing the word sample [111]. Therefore, the performance of the oracle recognizer may be considered as an upper bound on that of the lexicon-driven technique. Table 6.8 presents the results of the techniques. As one may expect, the top-1 accuracies of lexicon-free (77.36%) and the combination (74.83%) are substantially higher than the lexicon-driven (28.38%). There is a marginal decrease in accuracy with the combination when compared to that of lexicon-free. This is mainly due to the false positives from the lexicon-driven recognizer as it misrecognizes certain samples with high confidence. The top-5 accuracy of the combination is comparable to the oracle lexicon-driven recognizer.

When all the type-1 style samples were considered, the accuracy of the combination i.e. 89.29% (Table 6.6) was on par with the Top-1 accuracy of the oracle, which was found to be only 89.23%.

Table 6.8: Average accuracy (%) across 9-folds for type-1 Devanagari word samples having symbol order variations, with 20K lexicon

Approach	**Top1**	**Top5**
Lexicon-driven	28.38	39.53
Lexicon-free	77.36	93.92
Combination	74.83	94.26
Oracle lexicon-driven	92.23	95.44

6.6 Summary

In this chapter, we explored lexicon-driven and lexicon-free strategies for HMM-based word recognition. Our lexicon-free strategy uses a novel Bag-of-Symbols representation for symbol-order-free representation of words. We demonstrated empirically that the proposed approach addresses the problem of symbol order variation in Devanagari, and in combination with the lexicon-driven approach, high recognition accuracies (93.38% and 87.13% with the 1K and 20K lexicons respectively) can be achieved. The results of our investigation also indicate that the lexicon-driven approach is best suited for Tamil (96.03% and 91.8% with the 1K and 20K lexicons respectively) since symbol order variation is not significant in the script, and also because the presence of similar-looking symbols makes the task of recognition difficult without the context of the lexicon. The strategies proposed are data-driven and script-independent and are therefore extensible to other Indic scripts.

In the next chapter, we propose a novel IME for Indic scripts that recognizes handwriting input on small touch surfaces.

Chapter 7

Position-free Handwriting Input for Small Touch Interfaces

In this chapter we address the second part of the problem definition, and propose a handwriting-based text input solution for entering Indic scripts on small touch surfaces.

7.1 Introduction

Many interfaces have been developed for handwriting-based text input on devices that support stylus or touch for interaction. Handwriting has several intrinsic advantages over hard or soft keyboards such as naturalness, suitability for scripts with large character sets, and absence of seek time. These interfaces use *discrete symbol based* or *continuous* handwriting input methods. Examples of discrete symbol based methods include character input interfaces popular on PDAs and stylus equipped mobile phones. These require symbols to be entered one at a time into a specific writing area, and generally use a timeout to indicate the end of a symbol. This considerably slows down the process of writing. Unistroke alphabets such as Graffiti use the end of stroke to signify the end of a character and eliminate the timeout, but require the user to learn special sets of symbols. Continuous input interfaces on the other hand allow words to be written as a continuous stream of strokes (i.e. without explicit indication of character transitions), often support cursive styles, and may allow several words to be entered at once. These require larger digitizer surfaces as found on TabletPCs, external tablets, and electronic paperclip devices. Limited forms are also found on some PDAs. However, continuous input is difficult if not impossible when the writing area on the device is small and the stylus is substituted for by a finger, as shown in Figure 7.1.

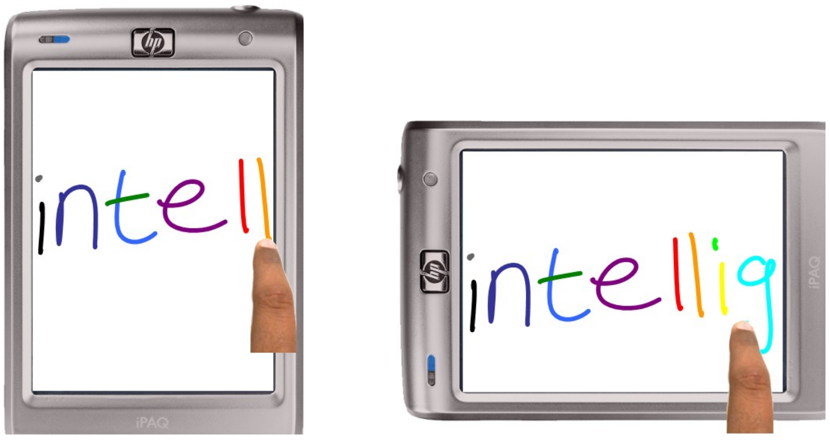


Figure 7.1: Insufficient space to enter the word *intelligent* using a finger: (a) portrait mode (b) landscape mode

Another issue with continuous handwriting input is the attention required during writing. Existing continuous input interfaces have been designed taking into account the conventional writing order of the script (for example, left to right in the case of English), so that the spatial positioning of the characters and words that make up the input is maintained. For example, the characters and words need to be approximately the same size, written next to one another, and aligned with a (imaginary or displayed) baseline. In addition to adhering to these requirements, users also unconsciously tend to read what they have written before sending the ink for recognition. As a result, such input methods require almost continuous user attention while writing, i.e. the user has to look at the writing surface to make sure that the input is "spatially correct" and the position of each unit of writing with respect to its predecessors and/or the baseline is maintained.

Where the writing (digitizing) surface is different from the application display surface (as in the case of an external digitizing tablet connected to a desktop PC), the user has to look at the writing surface while writing, and then at the display to see the results of recognition. This constant switching between the display device and the writing surface after each writing unit (which could be a character, word, or a larger unit, depending on the recognizer used) results in significant cognitive load on the user, and impacts the speed of text entry.

As mentioned in Section 1.2, handwriting recognition is even more relevant in the context of Indic scripts, which have large character sets, making text entry through the QWERTY keyboard overlays (for desktop and notebook PCs) and the keypads or soft keyboards (on mobile devices) extremely cumbersome.

7.2 Prior Work on Continuous Handwriting Input for Small Writing Surfaces

Given the inherent difficulty in supporting natural handwriting input on small writing surfaces, various interfaces have been explored by researchers. In an early effort, Goldberg and Richardson [112] proposed the "Unistroke" alphabet for entering the English letters. The alphabet contained carefully-designed single-stroke symbols for the English letters. Some of these symbols had identical shapes and differed only in the direction of writing. By using single-stroke symbols, Unistroke enabled continuous input while circumventing the problem of letter segmentation. With respect to user interaction, it supported "heads-up" or "eyes-free" writing, and could be used on small devices. However, a major limitation was that the users were burdened with the task of learning these new symbols before they could enjoy its benefits. Graffiti from Palm [113] offers single-stroke symbols that bear closer resemblance to the actual letters, when compared to Unistroke, but continues to suffer from the same limitations. EdgeWrite [114] also specifies a set of single-stroke symbols that are expected to be written by tracing along the edges of a square box (which might be the display surface or touchpad in practice). The sequence in which the corners of the square are visited during writing determines the input character. This is based on the observation that there is high stability in the writing when users gesture along the physical edges, making it particularly suitable for people with motor impairments. Systems such as Unistroke, Graffiti or EdgeWrite may not be practical for Indic scripts owing to the large character set sizes of the latter.

TreadMill Ink proposed by Seni [115] addresses the problem of continuous handwriting input on small devices, by providing a user interface with a virtual writing area that scrolls from right to left as one writes. Apart from recognizing mixed style natural writing, it can also automatically detect the end of a word, thereby eliminating the timeout needed for recognition. One principal shortcoming is that the user has to get accustomed to writing on a scrolling surface. Moreover, being a software solution, it needs an 'active' writing surface capable of providing continuous visual feedback to the user.

Relatively recent efforts by Shimodaira et al. [116] and Tonouchi and Kawamura [117] have explored "overlaid handwriting" for entering Japanese words. This addresses the problem of small real-estate by allowing the users to write normal characters one over another to form the word. However, it assumes that the relative positions between strokes forming a character are maintained by the user even while overwriting.

7.3 Prior Work on Handwriting-based Input Method Editors for Indic Scripts

While recognition of unconstrained continuous handwriting is clearly the end-objective, a number of practical Indic text input solutions (also called Input Method Editors or IMEs) can be obtained by imposing constraints on the writer. One such constraint is that of phonological symbol order: for example, constraining the writer to complete the consonant (C) before the matra (V) while writing a CV character (even when the matra appears to the left of the consonant, and is typically written before). This allows a two-step IME in which the consonant is first written and recognized (and manually corrected if necessary by selecting from an n-best list), and the matra written and recognized in a second step. A second constraint could be to ask the user to unravel consonant clusters or conjuncts into sequences of CH characters using the *halanth* (vowel muting diacritic), as opposed to using the half forms or consonant conjuncts. When used together, these constraints allow recognition of complex characters such as C'C'CV using only isolated symbol recognizers for C and V. Such strategies are used by IMEs such as the Gesture Keyboard [2], Guided Handwriting [118], Compact IME [119] and IndicDasher [120]. The Gesture Keyboard uses a combination of coarse pen position in a grid of base consonants, and recognition of the written matra, to interpret the user's CV input (Fig. 7.2). Guided Handwriting uses a set of basic stroke and substroke shapes together with rules to predict a consonant even as it is being written. Compact IME is proposed for mobile devices where the consonant is written first and followed by selecting the matra from a visual menu arranged based on the actual positions of the matras in a character. Srinivas et al. [120] propose Indic-Dasher, a modification of English Dasher [121] for Indic scripts. While a consonant is entered by simply pointing to it as in the case of English Dasher, in order to enter the matra, the user writes the matra around the consonant currently pointed at. The handwritten matra is then recognized and displayed along with the top N recognition choices.

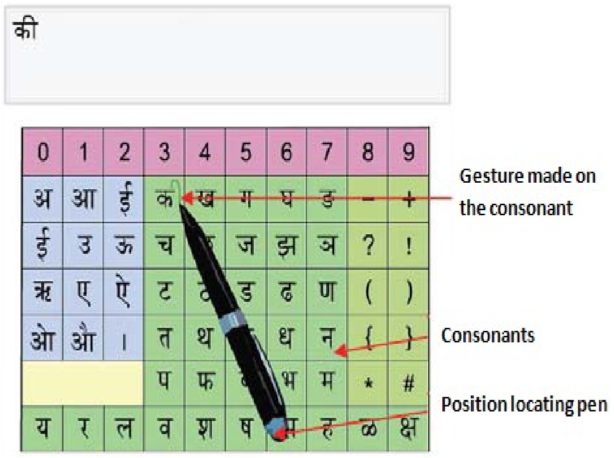


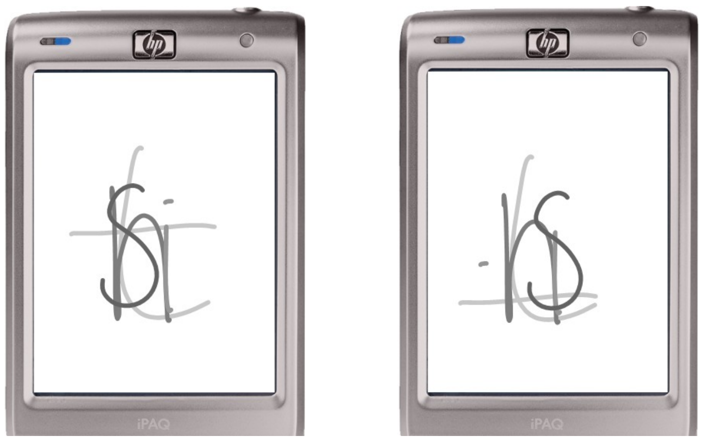
Figure 7.2: Gesture keyboard for entering Devanagari characters (Adapted from Balaji et al. [2])

7.4 FreePad IME: Position-free Handwriting Input

Taking into account the limitations of the existing approaches for continuous handwriting input on small devices (Section 7.2), we propose an IME called FreePad. The central idea of FreePad is that the user can completely disregard the positions and sizes of the input strokes and treat handwriting input simply as a sequence of independently written strokes and still expect it to be recognized. Even the relative positions and sizes of the strokes forming a character may be disregarded by the user. In contrast to systems based on single-stroke alphabets which have predefined symbols, the user can enter characters in the manner (s)he is used to writing. Since the positions and sizes of the strokes are ignored, FreePad is well suited for small devices where the user can "overwrite" the strokes of the word in place.

It is important to note the difference between FreePad and the overlaid handwriting work [116, 117] mentioned in Section 7.2. That effort also addresses the problem of small real-estate by allowing the users to write one character over another to form the word. However, it assumes that the relative positions and sizes of strokes forming a character are maintained by the user. In contrast, FreePad does not impose this constraint on the user. Figure 7.3 shows an example where the word 'this' is written as overlaid handwriting (Fig. 7.3(a)) and using FreePad (Fig. 7.3(b)). One may note that unlike in the case of overlaid handwriting, in FreePad, even the positions of the t-crossing and i-dot need not be maintained[1].

[1]Fig. 7.3(b) shows an extreme case of disregarding stroke positions in FreePad. In practice, users tend to maintain some resemblance of relative size and position especially within a character. However FreePad treats all input uniformly as being devoid of stroke position and size information.



(a) (b)

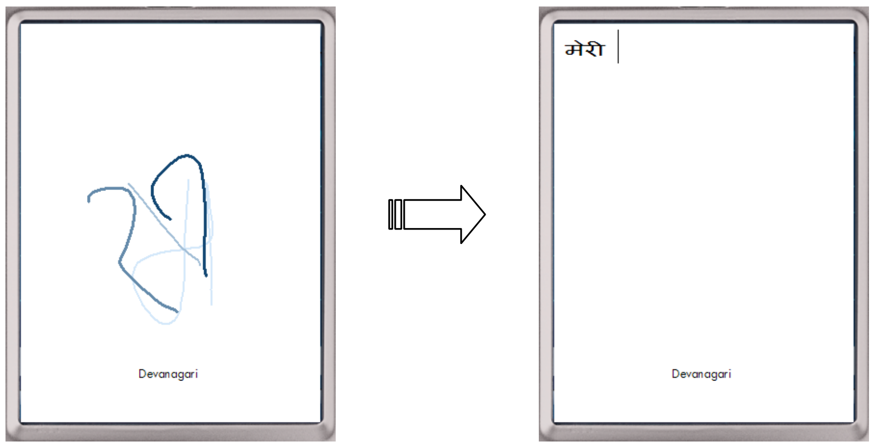
Figure 7.3: Comparison between input recognized by (a) overlaid handwriting (b) FreePad

Disregarding the positions of the strokes even within a character is an advantage, especially in the context of Indic scripts, where one is faced with the challenge of entering complex characters (e.g. CVM or vertically-stacked conjuncts in Telugu) using a finger on a small writing surface (e.g. mobile phone display, laptop touchpad, watch dial). The ability to recognize "position-free" handwriting also has other significant advantages. In addition to enabling "eyes-free" or "heads-up" text input, which was one of the objectives of Unistroke [112], position-free handwriting input is also appropriate in the absence of visual displays to guide writing, and for input using devices such as mice and "wands" which can only capture movement but not absolute (x, y) position.

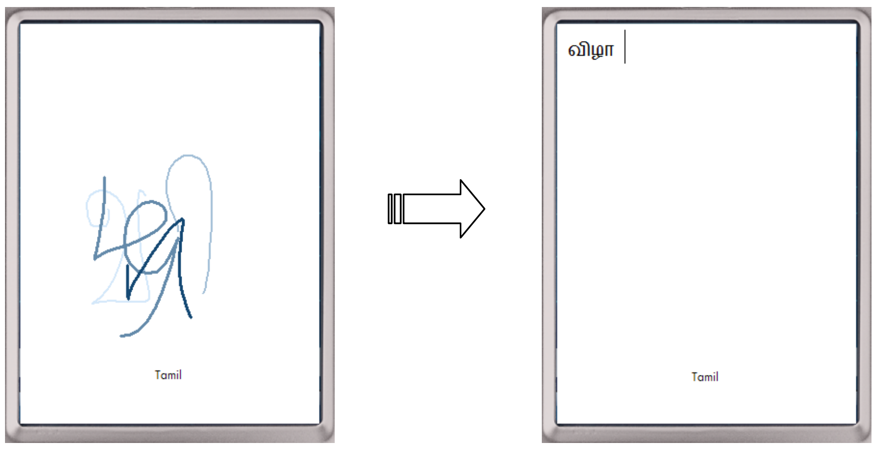
The FreePad IME concept as currently implemented (Fig. 7.4) accepts "position-free" input of handwritten words written in phonological order and discretely style (i.e. with pen lifts between symbols). In contrast to the approaches presented in Section 7.3, which only recognize isolated consonants or matras, it supports continuous input and does not require any explicit indication of symbol transition.

7.5 Consequences of Loss of Position Information

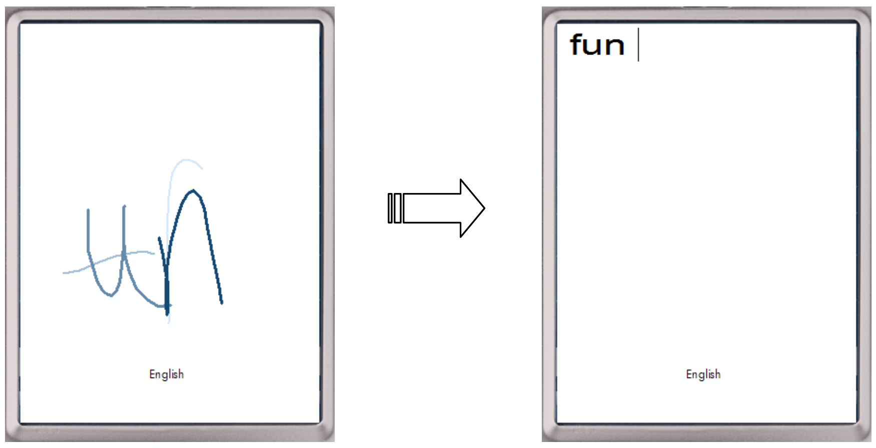
While an IME that can accept position-free handwriting input has several benefits, there are some noteworthy challenges in recognizing such input. For instance, ascenders and descenders, considered to be vital information in some approaches for Latin script word recognition, are



(a)



(b)



(c)

Figure 7.4: FreePad interface for (a) Devanagari, (b) Tamil and (c) English. The last four strokes of the writing are displayed with different shades, and the recognition of the word is triggered using a timeout.

no longer detectable. The effects are even more severe in the case of Indic scripts due to their intrinsic two-dimensional structure. In this section, we attempt to categorize the ambiguities that arise in recognition due to loss of position information.

1. *Ambiguities between multi-stroke symbols* - When relative positions of the strokes *within* a symbol are ignored, certain pairs of symbols become indistinguishable, e.g. symbol 26 (*/Na/*) and symbol 32 (*/pa/*) in Devanagari, shown on the left and right respectively in Fig. 7.5.

Figure 7.5: Ambiguity due to loss of position information within a symbol

2. *Ambiguities between symbols having identical shapes* - These arise due to the loss of position information *between* symbols. These symbols are visually the same but differ only in their positions in the context of the character or word (e.g. hyphen '-' and underscore '_'). Fig. 7.6 shows an example from Devanagari: symbol 54 (*/ai/ matra*) and symbol 62 (*halant*) differ only in their positions with respect to the base consonant (shown as a dotted circle).

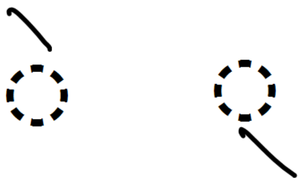


Figure 7.6: Ambiguity due to loss of position information between symbols

3. *Ambiguities between multi-stroke symbols and two or more symbols* - Such ambiguities are a result of loss of position information both *within* and *between* symbols. Fig. 7.7 shows an example from Devanagari where symbol 34 (*/ba/*) is a multi-stroke symbol, and symbol 40 (*/va/*) and symbol 54 (*/ai/ matra*) are two different symbols.

Since FreePad accepts completely position-free input, one can expect all the three forms of ambiguity to be present. In order to deal with these uncertainties, we take advantage of a lexicon which serves as a contextual knowledge source.

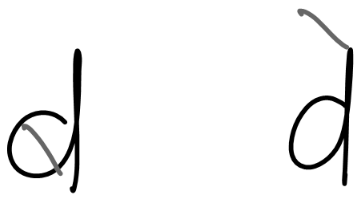


Figure 7.7: Ambiguity due to loss of position information within and between symbols

In the following sections we describe the steps involved in recognition of position-free handwriting, along with the results of our evaluation.

7.6 Recognition of Position-free Handwriting

The recognition of position-free handwriting may be accomplished using an approach similar to that used for conventional writing. In this section we describe the preprocessing and feature extraction steps involved in the recognition of position-free writing. For training the symbol models, and word recognition after feature extraction, the same approaches developed and described earlier for normal handwriting are adopted, and are not repeated here.

7.6.1 Preprocessing

Conventional preprocessing techniques for word recognition (e.g. in Fig. 4.6) typically do not disturb the positions of the strokes while normalizing their sizes. Therefore, such techniques are not suitable for processing position-free handwriting where the absolute positions of the strokes are either unknown or unreliable. We propose *stroke-level preprocessing* in order to deal with the variations in the input. In our approach, irrespective of the position of the stroke on the writing surface, we translate each stroke such that the (x_{min}, y_{min}) point of its bounding box is at the origin. In order to cope with the variability in sizes, each stroke is normalized by scaling the height or width of the bounding box, whichever is larger, to a constant value while maintaining the original aspect ratio of the stroke. Once the entire handwritten word has been preprocessed at the stroke level, each stroke in the temporal order appears to be written at the same location and with similar size as shown in Fig. 7.8. Following position and size normalization, each stroke is also resampled as described in Section 4.3 of Chapter 4.

Even though stroke-level preprocessing removes variations in the positions, it is not suitable when two symbols in the input are written cursively in a single stroke. As one can imagine,

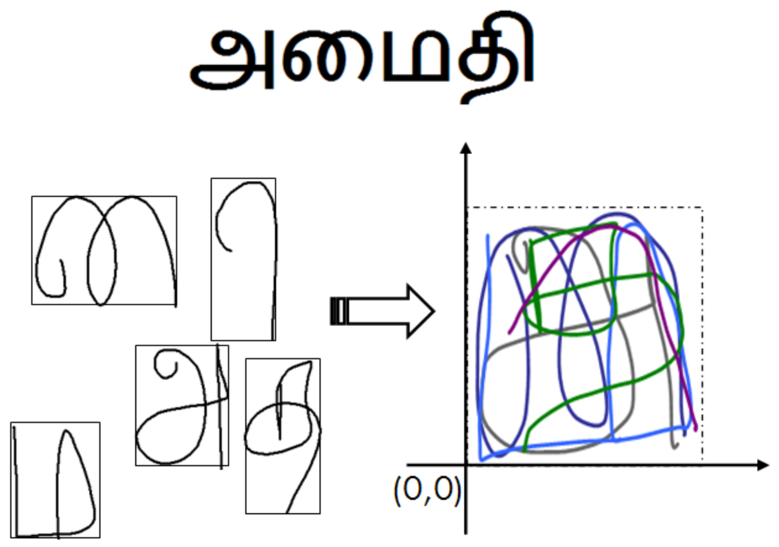


Figure 7.8: Position-free input of a Tamil word before and after stroke-level preprocessing

when such a stroke is normalized to a fixed size, the values of the points (x and y) would change significantly. Therefore, in the current implementation of FreePad, we restrict the input to be of the discrete style (type-1 in Section 6.5.1), wherein no two symbols in the word are written without a pen-up between them. It is important to note that the stroke-level preprocessing does not in any way impact recognition of different symbol writing styles varying in the shape, number, sequence or direction of strokes as long as the style was presented during training of the recognizer (Chapter 5).

7.6.2 Feature Extraction

In addition to the features extracted for normal handwriting described in Section 4.4, we also extract and use the x-value at each point after size normalization as one of the features. In conventional writing, the x-value varies with the position of the symbol within the word and is not directly useful for characterizing the shape of the stroke. However the x-value becomes a meaningful feature here because of the stroke-level preprocessing described earlier.

7.7 Experimental Evaluation

The datasets containing 'normal' handwriting samples (described in Chapter 3), were used for evaluating the performance of position-free handwriting recognition. This was possible because of the stroke-level preprocessing step in the recognizer which ignores the original positions of the strokes irrespective of the input. Thus, a word sample written conventionally on a large surface may be assumed to be no different from position-free handwriting input entered via the IME for the purpose of evaluating the position-free recognition system[2].

The dataset was divided into 10 folds as it was described in Section 5.6.1. For the reasons mentioned earlier, the performance of our position-free word recognizer was evaluated only on the type-1 samples, for both Devanagari and Tamil, and the recognition accuracy was computed as the average of 9 folds excluding the validation fold. The validation fold was used to determine the optimal beam widths and word insertion penalties for the HMM networks. The approaches (lexicon-driven, lexicon-free and combination) proposed earlier for normal handwriting recognition, were evaluated in the context of position-free handwriting. Since the shirorekha stroke(s) do not have any significance in the position-free writing of a Devanagari word and are not expected to be entered by the user, we removed them (using the symbol label) before evaluation.

7.7.1 Performance of Position-free Recognition

Table 7.1 shows the results obtained using the three approaches for position-free Devanagari for different lexicon sizes. As with normal handwriting, the combination of lexicon-driven and lexicon-free recognizers produced the highest accuracy (88.78% for the 20K-word lexicon). Similarly for Tamil (Table 7.2), the combination returns the highest accuracy (93.18% for 20K-word lexicon). The top-5 accuracy for 20K lexicons for both scripts exceeds 96%. From these results, it is clear that FreePad can be effective as an IME. As with other text input methods, the user can be given the option of selecting from alternative recognition choices in the event of misrecognition.

In the next sub-section, we study the relevance of position information by comparing these results with those for normal handwriting.

7.7.2 Relevance of Position Information

The experimental results for position-free handwriting recognition allow interesting comparisons to be made with those for normal handwriting. Some of the specific observations are as

[2]The assumption may not be valid if the writing surface is too small and has an impact on the shapes of the strokes written. Moreover, by using the normal handwriting dataset for evaluating position-free handwriting recognition, we assume that the severity of symbol order variation is the same in both types of input.

Table 7.1: Average accuracy (%) across 9-folds for position-free Devanagari word recognition using Type-1 samples

	1K		**2K**		**5K**		**10K**		**20K**	
Approach	**Top1**	**Top5**	**Top1**	**Top5**	**Top1**	**Top5**	**Top1**	**Top5**	**Top1**	**Top5**
Lexicon-driven	90.82	92.07	89.02	91.01	86.79	90.07	84.32	89.33	81.88	88.44
Lexicon-free	87.30	95.85	86.46	94.91	85.01	93.76	83.55	92.86	81.91	91.59
Combination	94.81	98.39	93.81	97.94	92.39	97.34	90.51	96.74	88.78	96.16

Table 7.2: Average accuracy (%) across 9-folds for position-free Tamil word recognition using Type-1 samples

	1K		**2K**		**5K**		**10K**		**20K**	
Approach	**Top1**	**Top5**	**Top1**	**Top5**	**Top1**	**Top5**	**Top1**	**Top5**	**Top1**	**Top5**
Lexicon-driven	96.03	97.76	94.90	97.52	94.09	97.13	93.46	96.61	92.48	96.42
Lexicon-free	89.16	96.68	88.88	96.50	88.03	95.72	87.73	95.26	86.49	94.57
Combination	96.13	98.78	95.74	98.61	94.98	98.31	94.16	98.13	93.18	97.76

follows.

- In the case of Tamil, the accuracy of the lexicon-free recognizer (86.49% for 20K) is substantially higher than what was obtained for normal handwriting (77.76% for 20K, from Section 6.4) with the same set of type-1 samples. We hypothesize that this improvement may be because of the use of the x feature which may have helped the recurrent HMM avoid the kind of segmentation errors described in Section 6.5.4. As a result, the combination of lexicon-driven and lexicon-free recognizers (93.18% for 20K) performs marginally better for position-free input, compared to the lexicon-driven approach alone (92.48%). This is in contrast to normal writing, where the much poorer performance of the lexicon-free approach adversely impacts the performance of the combination, causing it to perform worse than the lexicon-driven approach.

- In general, for Type-1 input, there is virtually no loss of accuracy due to the loss of position information, as compared to normal handwriting recognition.

From the above observations, the following conclusion can be drawn. When compared to the normal handwriting recognizer which has to preserve the positions of the strokes in order to recognize both discrete and cursive styles of writing, the position-free recognizer designed specifically for discrete-style input can achieve comparable accuracy even in the absence of position information, especially when a lexicon is available.

7.7.3 User Acceptance and Usability

While we have demonstrated the ability to recognize position-free input with acceptable accuracy, another important question concerns the user's ability to change their mental model of writing in order to use FreePad. In order to study this aspect, we conducted a preliminary user study involving twelve participants to determine user reactions to FreePad [122]. In particular, we compared the user experience of using FreePad with T9, a popular text input mechanism for SMS messaging, for the task of composing SMS messages in English. Subjective ratings by the users on a Likert scale of 1(lowest) to 7 (highest) corresponding to three attributes: naturalness, ease of use, and overall experience - were collected. The study subjects rated FreePad higher than T9 on all three counts. While limited in scale and scope, this study suggests that overwriting with a finger is easy and effective to use, and users are able to cope with the change in the model of writing.

7.8 Summary

In this chapter, we proposed a novel handwriting-based text input method for entering Indic scripts on a small device, using a finger. The proposed approach allows the user to "overwrite" a word in place, and offers several advantages such as supporting eyes-free writing. A brief overview of related prior efforts such as Unistroke and Overlaid handwriting, and IMEs for Indic scripts was presented. We also described the stroke-level preprocessing step that is used in FreePad to deal with the unreliable positions of the input strokes. Our results indicate that the accuracy of position-free handwriting recognition is comparable with that of normal handwriting when the input word sample is written discretely, and word context (in the form of lexicon) is available. Preliminary user studies (for English) suggest that FreePad is a good alternative to keypads for text input on small devices.

Chapter 8

Conclusions

8.1 Summary

This thesis addressed the problem of online handwritten word recognition for the Indic family of scripts, which are used by a sixth of the world's population. Specifically, it explored the use of script-independent and data-driven approaches for addressing this problem, so that the approach may easily be extended to other Indic scripts.

In the introductory chapter, having introduced the problem, we briefly discussed the structure of these scripts, and presented the challenges posed by them for online recognition, especially when compared to Latin and Oriental scripts. We presented an overview of our HMM-based approach to online handwritten word recognition, and the key steps of preprocessing and feature extraction, symbol modeling, and word recognition.

An overview of the established approaches for recognition of Latin and Oriental scripts, with an emphasis on HMM-based approaches, was then presented. We also presented an overview of the state of the art in online Indic script recognition. Previously explored approaches for recognition of handwritten words were seen to be either script-dependent or involve significant human intervention during training of the recognizer.

We then described the methodology adopted for creating handwriting word datasets for Devanagari and Tamil. The steps of defining the symbol set, identifying a minimal set of words for data collection using OTS, data cleanup and annotation, along with the various tools used for these tasks were also briefly described. These datasets have been used for training and evaluation of the handwriting recognition algorithms proposed in this work.

The various steps in preprocessing of the input word sample such as size normalization and resampling were then described. In the case of Devanagari, shirorekha stroke(s) are identified as the first step and used for size normalization. The performance of the shirorekha detection algorithm was evaluated on a test set and found to be more than 99% accurate. We then described the features extracted from the preprocessed ink at each point in the trajectory. These are adopted from the literature and include writing angles, aspect, curliness, linearity and slope.

Next, we described our approach to symbol modeling using HMMs. Our symbol model is a parallel-path network where each path corresponds to a style of writing the symbol, and contains a sequence of stroke HMMs. The stroke HMMs are shared across styles of the same or different symbols. While conventional stroke-based modeling approaches use unsupervised or heuristic-based stroke clustering algorithms in order to identify the unique strokes in the dataset, we proposed the use of *constrained stroke clustering*. We introduced some basic constraints, inherently present in the handwriting domain that one can apply to obtain better stroke clusters. We showed through empirical evaluation that Devanagari symbol modeling using writing styles and stroke clusters determined using the proposed approach achieves significantly higher recognition performance and/or reduced number of HMM states, when compared to alternate modeling techniques. The alternate modeling techniques studied included the use of a single HMM per character, character-based modeling using character-level clustering, and stroke-based modeling using unsupervised stroke clustering. In the case of Tamil, where writing style variations at the symbol level were not found to be significant, a single HMM was used to model each symbol.

We then described two different approaches to word recognition: lexicon-driven and lexicon-free. The former represented lexicon words as HMMs by concatenating the corresponding symbol HMMs in a predetermined, "standard" order. From the Devanagari word samples collected, we found that 7% of them are written in a symbol order different from the standard order. The accuracy of the lexicon-driven approach with these samples was found to be very low. Therefore we also proposed a lexicon-free approach that uses a novel *Bag-of-Symbols* representation for symbol-order-free representation of words. We demonstrated empirically that the proposed lexicon-free approach addresses the problem of symbol order variation in Devanagari, and using a Borda count combination with the lexicon-driven approach, high recognition accuracies (93.38% and 87.13% with the 1K and 20K lexicons respectively) can be achieved. The results of our investigation also indicated that the lexicon-driven approach is well suited for Tamil (96.03% and 91.8% with the 1K and 20K lexicons respectively) since the symbol order variation problem is not significant in the script.

Finally, we proposed FreePad, a novel text input method (IME) for handwriting Indic scripts using a finger on small touch surfaces, such as those increasingly common in mobile phones. The proposed solution allows the user to handwrite a word without regard for the relative positions and sizes of strokes, and in particular, overwrite a word in place on a small writing surface. In order to recognize this form of "position-free" writing, we proposed *stroke-level preprocessing* which ignores the original position and size of each stroke and normalizes them to have a fixed location and size. The results of our experiments indicated that the accuracy of recognizing position-free input is comparable to that of the normal handwriting when written discretely, despite disregarding the positions of the strokes. The maximum accuracies obtained for 20K word lexicons were 88.78% for Devanagari and 93.18% for Tamil, when the lexicon-driven and lexicon-free approaches were used in combination. Preliminary user studies for English indicate that FreePad is a viable text input method for mobile devices.

8.2 Contributions

This thesis contributes to the field of OHWR in a number of areas. The main contributions are as follows:

1. *Script-independent and data-driven approach for unconstrained online handwritten word recognition in Indic scripts*: To the best of our knowledge, there is no prior work on recognition of online handwritten words in Indic scripts written in an unconstrained manner. Previous efforts have been script-specific and required manual intervention while training, or assumed constraints on the writer, or both. This thesis addresses this gap by proposing a *script-independent* and *data-driven* approach using HMMs for online handwritten word recognition in Indic scripts that requires very little manual intervention beyond the design of the symbol set and creation of labeled datasets, and is extensible to other Indic scripts. The approach also addresses the two central issues that arise in the context of online Indic scripts: (i) *variations in writing style of individual symbols*, and (ii) *symbol order variations within and across characters*. The approach has been validated in this thesis for two very different and significant Indic scripts - Devanagari and Tamil.

2. *Stroke-based symbol modeling using Constrained Stroke Clustering for discovering writing styles*: In order to deal with different types of writing style variations at the symbol level, a novel approach for symbol modeling is proposed, that is *stroke-based*, completely data-driven, and discovers writing styles automatically from the training data. The approach uses a novel technique based on *constrained stroke clustering* that exploits constraints implicit in the handwriting domain to identify the optimal set of strokes and writing styles in the dataset. Results obtained for Devanagari symbols using the resulting symbol models indicate substantial improvement in recognition accuracy and/or reduction in HMM states, when compared to alternative modeling techniques.

3. *Lexicon-driven approach for word recognition*: The proposed HMM-based lexicon-driven approach for Indic scripts represents the lexicon as a *prefix-tree assuming standard symbol order*, and works well when there is little symbol order variation. For instance, the top-1 and top-5 recognition accuracies on Tamil words for this approach were 91.8% and 96.32% respectively, for a lexicon containing 20K words.

4. *Lexicon-free approach using the BoS representation of words*: In order to deal with the symbol order variations in scripts such as Devanagari, a novel lexicon-free approach was proposed that uses a novel *Bag-of-Symbols (BoS) representation* of words. For word samples with non-standard symbol orders, BoS based lexicon-free recognition achieved significantly higher accuracy (77.36%) when compared to the lexicon-driven approach (28.38%). A *Borda count combination* of the lexicon-driven and lexicon-free approaches performs well in the general case where a mixture of standard and non-standard symbol orders may be expected. For Devanagari, maximum accuracies of 93.38% and 87.13% with the 1K and 20K lexicons respectively were achieved when all the samples were considered.

5. *Enabling handwriting input using a finger on small touch surfaces*: Also proposed in this thesis is a novel text input method (IME) named FreePad that enables "position-free" handwriting input on small touch surfaces by allowing the user to disregard the positions and sizes of the strokes in the input. In the absence of other usable methods, FreePad has considerable potential as a practical solution for enabling text input in Indic scripts into mobile phones.

6. *Datasets and dataset creation tools for Indic scripts*: As part of the research described in this thesis, online handwriting datasets containing symbol-level-annotated word samples of over a hundred writers were created for Devanagari and Tamil. A tool for collecting handwriting data using the DigiMemo device was developed, which is highly suited for large-scale data collection efforts and is freely available as part of the Lipi Toolkit [123].

Given that there is no prior work on Devanagari or Tamil online word recognition, we hope that the recognition performances reported in this work will serve as a benchmark for future efforts.

8.3 Future Research Directions

In this thesis we have explored various facets of online handwritten word recognition. In doing so, we have uncovered a number of promising directions for future research. Some of these are described below.

8.3.1 Symbol Modeling

1. We have used the classical Baum-Welch algorithm for training stroke HMMs. It would be interesting to exploit the constraints identified during the style identification process to train the stroke HMMs in a discriminative manner. Alternatively, approaches that learn subspaces using the given constraints may be investigated. Conditional Random Fields (CRF) may be explored as an alternative to HMMs for modeling strokes and symbols.

2. While we found the optimum number of stroke clusters based on a threshold distance, we would like to explore whether the constraints themselves can be exploited for the purpose. The problem of determining the optimum number of clusters (model selection) based on constraints appears to be relatively unaddressed in the Constrained Clustering literature.

8.3.2 Word Recognition

1. An important problem in the case of Indic script recognition is supporting the different forms in which a character may be written. While our lexicons currently use only the distinct forms of conjuncts (such as symbol 103 in Devanagari), explicitly modeling all the alternate forms (such as 78 followed by 31) may result in better performance.

2. In the case of Devanagari, while the combination of lexicon-driven and lexicon-free recognizers is able to cope with ligatures in the type-2 style and delayed strokes in type-3 to some extent, explicit modeling of such artifacts (as is typically done in the case of Latin script recognition) may further improve performance.

3. The distance function that finds the dissimilarity between two BoS representations currently assigns Boolean scores while matching pairs of symbols. However, distance functions that take shape similarities into account while continuing to be agnostic of symbol-order may improve the overall performance of lexicon shortlisting and matching.

4. Both the lexicon-driven and lexicon-free approaches are limited to using a lexicon for improving accuracy. However, for real-life applications where one may encounter out of vocabulary words, language models at the word level would be important to include to aid recognition.

8.3.3 FreePad

1. While FreePad currently recognizes discrete-style word input, it may be extended to support more cursive styles of writing that do not always have a pen-up between every pair of symbols. This may be possible by using an appropriate set of features (in lieu of the current stroke-level preprocessing) such as writing angles and curvature which capture the shape of the stroke even while being invariant to its size and position.

2. Word or sentence-level language models will be important to further boost the performance of position-free handwriting recognition, especially when the word context (from the lexicon) proves insufficient for resolving ambiguities to loss of position information.

3. The user acceptance and usability of FreePad for Indic scripts needs to be studied along the lines of the study conducted for English.

8.3.4 Extension to other Indic Scripts

While the proposed approach is script-independent by design and has been evaluated with two very distinct scripts - Devanagari and Tamil, its extensibility to other Indic scripts needs to be rigorously evaluated. In particular, Telugu with its vertical stacking of symbols, or Bangla wherein cursive writing is common, would be good candidates to evaluate this approach with.

Bibliography

[1] S. Madhvanath and V. Govindaraju, “The role of holistic paradigms in handwritten word recognition,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 23, no. 2, pp. 149–164, Feb. 2001.

[2] R. Balaji, V. Deepu, S. Madhvanath, and J. Prabhakaran, “Handwritten gesture recognition for gesture keyboard,” in *Proc. 10th International Workshop on Frontiers in Handwriting Recognition (IWFHR’06)*, La Baule, France, Oct. 2006.

[3] C. C. Tappert, C. Y. Suen, and T. Wakahara, “On-line handwriting recognition – A survey,” in *Proc. 9th International Conference on Pattern Recognition (ICPR’88)*, Rome, Italy, Nov. 1988, pp. 1123–1132.

[4] C. Tappert, C. Suen, and T. Wakahara, “State of the art in on-line handwriting recognition,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 8, pp. 787–808, Aug. 1990.

[5] R. Plamondon and S. N. Srihari, “Online and off-line handwriting recognition: A comprehensive survey,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 1, pp. 63–84, Jan. 2000.

[6] C. L. Liu, S. Jaeger, and M. Nakagawa, “Online recognition of Chinese characters: The state-of-the-art,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 2, pp. 198–213, Feb. 2004.

[7] S. Jaeger, Liu, and M. Nakagawa, “The state of the art in Japanese on-line handwriting recognition compared to techniques in western handwriting recognition,” *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 6, no. 2, pp. 75–88, Oct. 2003.

[8] List of languages by number of native speakers. [Online]. Available: http://en.wikipedia.org/wiki/List_of_languages_by_number_of_native_speakers

[9] F. Coulmas, *The Blackwell Encyclopedia of Writing Systems*. Oxford: Blackwell, 1996.

[10] S. P. Mudur, N. Nayak, S. Shanbhag, and R. K. Joshi, "An architecture for the shaping of Indic texts," *Computers & Graphics*, vol. 23, no. 1, pp. 7–24, Feb. 1999.

[11] Tamil script. [Online]. Available: http://en.wikipedia.org/wiki/Tamil_script

[12] G. Seni, R. K. Srihari, and N. Nasrabadi, "Large vocabulary recognition of on-line handwritten cursive words," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 18, no. 7, pp. 757–762, Jul. 1996.

[13] E. Lecolinet and O. Baret, "Cursive word recognition: Methods and strategies," in *Fundamentals in Handwriting Recognition*, S. Impedovo, Ed. New York: Springer-Verlag, 1994, pp. 235–263.

[14] A. Vinciarelli, "A survey on off-line cursive word recognition," *Pattern Recognition*, vol. 35, no. 7, pp. 1433–1446, Jul. 2002.

[15] R. G. Casey and E. Lecolinet, "A survey of methods and strategies in character segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 18, no. 7, pp. 690–706, Jul. 1996.

[16] L. Schomaker and E. Segers, "Finding features used in the human reading of cursive handwriting," *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 2, no. 1, pp. 13–18, Jul. 1999.

[17] T. Steinherz, E. Rivlin, and N. Intrator, "Offline cursive script word recognition – A survey," *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 2, no. 2-3, pp. 90–110, Dec. 1999.

[18] R. K. Powalka, "An algorithm toolbox for on-line cursive script recognition," Ph.D. dissertation, The Nottingham Trent University, Nottingham, May 1995.

[19] T. Starner, J. Makhoul, R. Schwartz, and G. Chou, "On-line cursive handwriting recognition using speech recognition methods," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'94)*, Adelaide, Australia, Apr. 1994, pp. V/125–V/128.

[20] J. Hu, S. G. Lim, and M. K. Brown, "Writer independent on-line handwriting recognition using an HMM approach," *Pattern Recognition*, vol. 33, no. 1, pp. 133–147, Jan. 2000.

[21] L. R. Rabiner and B. H. Juang, "An introduction to Hidden Markov Models," *IEEE Acoustics, Speech & Signal Processing Magazine*, vol. 3, no. 1, pp. 4–16, Jan. 1986.

[22] S.-C. Oh, J.-Y. Ha, and J. H. Kim, "Context dependent search in interconnected Hidden Markov Model for unconstrained handwriting recognition," *Pattern Recognition*, 1995.

[23] P. R. Cavalin, A. de Souza Britto Jr., F. avio Bortolozzi, R. Sabourin, and L. E. S. Oliveira, "An implicit segmentation-based method for recognition of handwritten strings of caracters," in *Proc. 21st Annual ACM Symposium on Applied Computing*, Dijon, France, Apr. 2006, pp. 836–840.

[24] D. Li, A. Biern, and J. Subrahmonia, "HMM topology optimization for handwriting recognition," in *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'01)*, Salt Lake City, USA, May 2001, pp. 1521–1524.

[25] M. Zimmermann and H. Bunke, "Hidden Markov Model length optimization for handwriting recognition systems," in *Proc. 8th International Workshop on Frontiers in Handwriting Recognition (IWFHR'02)*, Ontario, Canada, Aug. 2002, pp. 369–374.

[26] H. Shu, "On-line handwriting recognition using Hidden Markov Models," Master's thesis, Massachusetts Institute of Technology, Massachusetts, USA, Feb. 1997.

[27] K. S. Nathan, H. S. M. Beigi, J. Subrahmonia, G. J. Clary, and H. Maruyama, "Real-time on-line unconstrained handwriting recognition using statistical methods," in *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'95)*, Detroit, USA, May 1995, pp. 2619–2622.

[28] T. Wakahara and K. Okada, "On-line cursive Kanji character recognition using stroke-based affine transformation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 12, pp. 1381–1385, Dec. 1997.

[29] J. Liu, Cham, and M. M. Y. Chang, "Stroke order and stroke number free on-line Chinese character recognition using attributed relational graph matching," in *Proc. 13th International Conference on Pattern Recognition (ICPR'96)*, Vienna, Austria, Aug. 1996, pp. 259–263.

[30] J.-O. Kwon, B. Sin, and J. H. Kim, “Recognition of on-line cursive Korean characters combining statistical and structural methods,” *Pattern Recognition*, vol. 30, no. 8, pp. 1255–1263, Aug. 1997.

[31] B.-K. Sin and J. H. Kim, “Ligature modeling for online cursive script recognition,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 6, pp. 623–633, Jun. 1997.

[32] S. Marukatat and T. Artieres, “Handling spatial information in on-line handwriting recognition,” in *Proc. 9th International Workshop on Frontiers in Handwriting Recognition (IWFHR’04)*, Tokyo, Japan, Oct. 2004, pp. 14–19.

[33] H. J. Kim, S. K. Kim, K. H. Kim, and J. K. Lee, “An HMM-based character recognition network using level building,” *Pattern Recognition*, vol. 30, no. 3, pp. 491–502, Mar. 1997.

[34] H. J. Kim, K. H. Kim, S. K. Kim, and J. K. Lee, “On-line recognition of handwritten Chinese characters based on Hidden Markov Models,” *Pattern Recognition*, vol. 30, no. 9, pp. 1489–1500, Sep. 1997.

[35] B.-K. Sin, J.-Y. Ha, S.-C. Oh, and K. J. H., “Network-based approach to online cursive script recognition,” *IEEE Trans. Syst., Man, Cybern.*, vol. 29, no. 2, pp. 321–328, Apr. 1999.

[36] M. Nakai, N. Akira, H. Shimodaira, and S. Sagayama, “Substroke approach to HMM-based on-line Kanji handwriting recognition,” in *Proc. 6th International Conference on Document Analysis and Recognition (ICDAR’01)*, Seattle, Washington, Sep. 2001, pp. 491–495.

[37] J. Tokuno, N. Inami, S. Matsuda, M. Nakai, H. Shimodaira, and S. Sagayama, “Context-dependent substroke model for HMM-based on-line handwriting recognition,” in *Proc. 8th International Workshop on Frontiers in Handwriting Recognition (IWFHR’02)*, Ontario, Canada, Aug. 2002, pp. 78–83.

[38] T. Hasegawa, H. Yasuda, and T. Matsumoto, “Fast discrete HMM algorithm for on-line handwriting recognition,” in *Proc. 15th International Conference on Pattern Recognition (ICPR’00)*, Barcelona, Spain, Sep. 2000, pp. 535–538.

[39] H. Swethalakshmi, A. Jayaraman, V. S. Chakravarthy, and C. C. Sekhar, “Online handwritten character recognition of Devanagari and Telugu characters using Support Vector Machines,” in *Proc. 10th International Workshop on Frontiers in Handwriting Recognition (IWFHR’06)*, La Baule, France, Oct. 2006.

[40] A. Jayaraman, C. C. Sekhar, and V. S. Chakravarthy, “Modular approach to recognition of strokes in Telugu script,” in *Proc. 9th International Conference on Document Analysis and Recognition (ICDAR’07)*, Curitiba, Brazil, Sep. 2007, pp. 501–505.

[41] K. H. Aparna, V. Subramanian, M. Kasirajan, G. V. Prakash, V. S. Chakravarthy, and S. Madhvanath, “Online handwriting recognition for Tamil,” in *Proc. 9th International Workshop on Frontiers in Handwriting Recognition (IWFHR’04)*, Tokyo, Japan, Oct. 2004, pp. 438–443.

[42] H. Swethalakshmi, “Online handwritten character recognition for Devanagari and Tamil scripts using Support Vector Machines,” Master’s thesis, Indian Institute of Technology, Madras, India, Oct. 2007.

[43] V. Babu, L. Prasanth, R. Sharma, G. Rao, and A. Bharath, “HMM-based online handwriting recognition system for Telugu symbols,” in *Proc. 9th International Conference on Document Analysis and Recognition (ICDAR’07)*, Curitiba, Brazil, Sep. 2007, pp. 63–67.

[44] V. Deepu, S. Madhvanath, and A. G. Ramakrishnan, “Principal Component Analysis for online handwritten character recognition,” in *Proc. 17th International Conference on Pattern Recognition (ICPR’04)*, Cambridge, United Kingdom,, Aug. 2004, pp. 327–330.

[45] N. Joshi, G. Sita, A. G. Ramakrishnan, and S. Madhvanath, “Tamil handwriting recognition using subspace and DTW based classifiers,” in *Proc. 11th International Conference on Neural Information Processing (ICONIP’04)*, Calcutta, India, Nov. 2004, pp. 806–813.

[46] N. Joshi, G. Sita, A. G. Ramakrishnan, and S. Madhvanath, “Comparison of elastic matching algorithms for online Tamil handwritten character recognition,” in *Proc. 9th International Workshop on Frontiers in Handwriting Recognition (IWFHR’04)*, Tokyo, Japan, Oct. 2004, pp. 444–449.

[47] S. Madhvanath and S. M. Lucas, "IWFHR 2006 online Tamil handwritten character recognition competition," in *Proc. 10th International Workshop on Frontiers in Handwriting Recognition (IWFHR'06)*, La Baule, France, Oct. 2006.

[48] N. Joshi, G. Sita, A. G. Ramakrishnan, V. Deepu, and S. Madhvanath, "Machine recognition of online handwritten Devanagari characters," in *Proc. 8th International Conference on Document Analysis and Recognition (ICDAR'05)*, Seoul, Korea, Aug-Sep 2005, pp. 1156–1160.

[49] V. Deepu and S. Madhvanath, "Genetically evolved transformations for rescaling online handwritten characters," in *Proc. IEEE India Annual Conference (INDICON'04)*, Kharagpur, India, Dec. 2004, pp. 262–265.

[50] A. H. Toselli, M. Pastor, and E. Vidal, "On-line handwriting recognition system for Tamil handwritten characters," in *Pattern Recognition and Image Analysis*. Berlin/Heidelberg: Springer, 2007, pp. 370–377.

[51] L. Prasanth, V. J. Babu, R. R. Sharma, G. V. P. Rao, and M. Dinesh, "Elastic matching of online handwritten Tamil and Telugu scripts using local features," in *Proc. 9th International Conference on Document Analysis and Recognition (ICDAR'07)*, Curitiba, Brazil, Sep. 2007, pp. 1028–1032.

[52] S. Sundaram and A. G. Ramakrishnan, "A novel hierarchical classification scheme for online Tamil character recognition," in *Proc. 9th International Conference on Document Analysis and Recognition (ICDAR'07)*, Curitiba, Brazil, Sep. 2007, pp. 1218–1222.

[53] C. S. Sundaresan and S. S. Keerthi, "A study of representations for pen based handwriting recognition of Tamil characters," in *Proc. 5th International Conference on Document Analysis and Recognition (ICDAR'99)*, Bangalore, India, Sep. 1999, pp. 422–425.

[54] P. V. S. Rao and T. M. Ajitha, "Telugu script recognition – A feature based approach," in *Proc. 3rd International Conference on Document Analysis and Recognition (ICDAR'95)*, Montreal, Canada, Aug. 1995, pp. 323–326.

[55] S. Connell, R. Sinha, and A. Jain, "Recognition of unconstrained on-line Devanagari characters," in *Proc. 15th International Conference on Pattern Recognition (ICPR'00)*, Barcelona, Spain, Sep. 2000, pp. 368–371.

[56] A. Ranade and M. Ranade, "Devanagari pen-written character recognition," in *Proc. 9th International Conference on Advanced Computing and Communications (ADCOM'01)*, Bhubaneshwar, India, Dec. 2001.

[57] R. Niels and L. Vuurpijl, "Dynamic Time Warping applied to Tamil character recognition," in *Proc. 8th International Conference on Document Analysis and Recognition (ICDAR'05)*, Seoul, Korea, Aug-Sep 2005, pp. 730–734.

[58] R. S. R. Kunte and R. D. S. Samuel, "On-line character recognition system for handwritten characters/script with bilingual facility employing neural classifiers and wavelet features," in *Proc. International Conference on Knowledge based Computer Systems (KBCS'00)*, Mumbai, India, Dec. 2000, pp. 1–12.

[59] U. Bhattacharya, B. K. Gupta, and S. K. Parui, "Direction code based features for recognition of online handwritten characters of Bangla," in *Proc. 9th International Conference on Document Analysis and Recognition (ICDAR'07)*, Curitiba, Brazil, Sep. 2007, pp. 58–62.

[60] S. Sundaram and A. G. Ramakrishnan, "Two dimensional Principal Component Analysis for online Tamil character recognition," in *Proc. 11th International Conference on Frontiers in Handwriting Recognition (ICFHR'08)*, Montreal, Canada, Aug. 2008, pp. 88–94.

[61] A. Krishna, G. Prabhu, K. Bali, and S. Madhvanath, "Indic scripts based online form filling – A usability exploration," in *Proc. 11th International Conference on Human-Computer Interaction (HCII'05)*, Las Vegas, USA, Jul. 2005.

[62] U. Bhattacharya, A. Nigam, Y. S. Rawat, and S. K. Parui, "An analytic scheme for online handwritten Bangla cursive word recognition," in *Proc. 11th International Conference on Frontiers in Handwriting Recognition (ICFHR'08)*, Montreal, Canada, Aug. 2008, pp. 320–325.

[63] A. S. Bhaskarabhatla and S. Madhvanath, "Experiences in collection of handwriting data for online handwriting recognition in Indic scripts," in *Proc. 4th International Conference on Language Resources and Evaluation (LREC'04)*, Lisbon, Portugal, May 2004, pp. 2223–2226.

[64] The EMILLE/CIIL corpus. [Online]. Available: http://www.elda.org/catalogue/en/text/W0037.html

[65] B. Kalika, A. G. Ramakrishnan, P. P. Talukdar, and N. S. Krishna, "Tools for the development of a Hindi speech synthesis system," in *Proc. 5th ISCA Speech Synthesis Workshop*, Pittsburgh, USA, Jun. 2004, pp. 109–114.

[66] ACECAD DigiMemo A402. [Online]. Available: http://www.acecad.com.tw/dma402.html

[67] I. Guyon, L. Schomaker, R. Plamondon, M. Liberman, and S. Janet, "UNIPEN project of online data exchange and recognizer benchmarks," in *Proc. International Conference on Pattern Recognition (ICPR'94)*, Jerusalem, Israel, Oct. 1994, pp. 29–33.

[68] A. Namboodiri and A. K. Jain, "Online handwritten script recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 1, pp. 124–130, Jan. 2004.

[69] S. Jaeger, S. Manke, J. Reichert, and A. Waibel, "Online handwriting recognition: The NPen++ recognizer," *International Journal on Document Analysis and Recognition*, vol. 3, no. 3, pp. 169–180, Mar. 2001.

[70] C. Viard-Gaudin, P. M. Lallican, P. Binter, and S. Knerr, "The IRESTE on/off (IRONOFF) dual handwriting database," in *Proc. 5th International Conference on Document Analysis and Recognition (ICDAR'99)*, Bangalore, India, Sep. 1999, pp. 455–458.

[71] S. Connell, "Online handwriting recognition using multiple pattern class models," Ph.D. dissertation, Michigan State Univ., Michigan, USA, May 2000.

[72] L. Prevost and M. Milgram, "Non-supervised determination of allograph sub-classes for on-line omni-scriptor handwriting recognition," in *Proc. 5th International Conference on Document Analysis and Recognition (ICDAR'99)*, Bangalore, India, Sep. 1999, pp. 438–441.

[73] J. Lee, J. Kim, and J. Kim, "Data-driven design of HMM topology for on-line handwriting recognition," in *Proc. 7th International Workshop on Frontiers in Handwriting Recognition (IWFHR'00)*, Amsterdam, The Netherlands, Sep. 2000, pp. 107–121.

[74] K. Takahashi, H. Yasuda, and T. Matsumoto, “A fast HMM algorithm for on-line handwritten character recognition,” in *Proc. 4th International Conference on Document Analysis and Recognition (ICDAR’97)*, Ulm, Germany, Aug. 1997, pp. 369–375.

[75] M. P. Perrone and S. Connell, “K-Means clustering for Hidden Markov Models,” in *Proc. 7th International Workshop on Frontiers in Handwriting Recognition (IWFHR’00)*, Amsterdam, The Netherlands, Sep. 2000, pp. 229–238.

[76] M. Nakai, H. Shimodaira, and S. Sagayama, “Generation of hierarchical dictionary for stroke-order free Kanji handwriting recognition based on substroke HMM,” in *Proc. 7th International Conference on Document Analysis and Recognition (ICDAR’03)*, Edinburgh, Scotland,, Aug. 2003, pp. 514–518.

[77] K. Yamasaki, “Automatic prototype stroke generation based on stroke clustering for on-line handwritten Japanese character recognition,” in *Proc. 5th International Conference on Document Analysis and Recognition (ICDAR’99)*, Bangalore, India, Sep. 1999, pp. 673–676.

[78] N. Matic, J. Platt, and T. Wang, “QuickStroke: An incremental on-line Chinese handwriting recognition system,” in *Proc. 16th International Conference on Pattern Recognition (ICPR’02)*, Quebec City, Canada, Aug. 2002, pp. 435–439.

[79] T. Kohonen, “The self-organizing map,” *Proc. IEEE*, vol. 78, no. 9, pp. 1464–1480, Sep. 1990.

[80] A. Biem, “Minimum classification error training for online handwriting recognition,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 7, pp. 1041–1051, Jul. 2006.

[81] C.-L. Liu and M. Nakagawa, “Evaluation of prototype learning algorithms for nearest-neighbor classifier in application to handwritten character recognition,” *Pattern Recognition*, vol. 34, no. 3, pp. 601–615, Mar. 2001.

[82] V. Vuori and J. Laaksonen, “A comparison of techniques for automatic clustering of handwritten characters,” in *Proc. 16th International Conference on Pattern Recognition (ICPR’02)*, Quebec City, Canada, Aug. 2002, pp. 168–171.

[83] S. Connell and A. Jain, "Learning prototypes for on-line handwritten digits," in *Proc. 14th International Conference on Pattern Recognition (ICPR'98)*, Brisbane, Australia, Aug. 1998, pp. 182–184.

[84] K. Chellapilla, P. Simard, and A. Abdulkader, "Allograph based writer adaptation for handwritten character recognition," in *Proc. 10th International Workshop on Frontiers in Handwriting Recognition (IWFHR'10)*, La Baule, France, Oct. 2006.

[85] V. Vuori, "Clustering writing styles with a self-organizing map," in *Proc. 8th International Workshop on Frontiers in Handwriting Recognition (IWFHR'02)*, Ontario, Canada, Aug. 2002, pp. 345–350.

[86] C. Bahlmann and H. Burkhardt, "The writer independent online handwriting recognition system frog on hand and cluster generative statistical dynamic time warping," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 3, pp. 299–310, Mar. 2004.

[87] L. Vuurpijl and L. Schomaker, "Coarse writing-style clustering based on simple stroke-related features," in *Progress in Handwriting Recognition*, A. Downton and S. Impedovo, Eds. London: World Scientific, 1997, pp. 37–44.

[88] L. Vuurpijl and L. Schomaker, "Finding structure in diversity: A hierarchical clustering method for the categorization of allographs in handwriting," in *Proc. 4th International Conference on Document Analysis and Recognition (ICDAR'97)*, Ulm, Germany, Aug. 1997, pp. 387–393.

[89] A. Bharath, V. Deepu, and S. Madhvanath, "An approach to identify unique styles in online handwriting recognition," in *Proc. 8th International Conference on Document Analysis and Recognition (ICDAR'05)*, Seoul, South Korea, Aug-Sep 2005, pp. 775–778.

[90] S. Basu, "Semi-supervised clustering: Probabilistic models, algorithms and experiments," Ph.D. dissertation, Univ. of Texas at Austin, Texas, USA, Aug. 2005.

[91] S. Basu, M. Bilenko, A. Banerjee, and R. Mooney, "Probabilistic semi-supervised clustering with constraints," in *Semi-Supervised Learning*, O. Chapelle, B. Scholkopf, and A. Zien, Eds. Cambridge, USA: MIT Press, 2006, pp. 73–102.

[92] L. Yi, J. Rong, and A. K. Jain, "BoostCluster: Boosting clustering by pairwise constraint," in *Proc. 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07)*, San Jose, USA, Aug. 2007, pp. 450–459.

[93] B. Kulis, S. Basu, I. Dhillon, and R. Mooney, "Semi-supervised graph clustering: A kernel approach," in *Proc. 22nd International Conference on Machine Learning (ICML'05)*, Bonn, Germany, Aug. 2005, pp. 457–464.

[94] K. Wagstaff, C. Cardie, S. Rogers, and S. Schrdl, "Constrained K-means clustering with background knowledge," in *Proc. 18th International Conference on Machine Learning (ICML'01)*, MA, USA, Jun-Jul 2001, pp. 577–584.

[95] S. Basu and I. Davidson, "Clustering under constraints: Theory and practice," in *Tutorial presented at the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'06)*, Philadelphia, USA, Aug. 2006.

[96] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. New York: Wiley, 2001.

[97] S. Salvador and P. Chan, "Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms."

[98] A. Fred and A. Jain, "Combining multiple clusterings using evidence accumulation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 6, pp. 835–850, Jun. 2005.

[99] L. Zelnik-Manor and P. Peronam, "Self-tuning spectral clustering," in *Proc. 18th Annual Conference on Neural Information Processing Systems (NIPS'04)*, Vancouver, Canada, Dec. 2004, pp. 1601–1608.

[100] D. Klein, S. Kamvar, and C. Manning, "From instance-level constraints to space-level constraints: Making the most of prior knowledge in data clustering," in *Proc. 19th International Conference on Machine Learning (ICML'02)*, Sydney, Australia, Jul. 2002, pp. 307–314.

[101] A. Bharath and S. Madhvanath, "A framework based on semi-supervised clustering for discovering unique writing styles," in *Proc. 10th International Conference on Document Analysis and Recognition (ICDAR'09)*, Barcelona, Spain, Jul. 2009, pp. 891–895.

[102] G. Kim and V. Govindaraju, “A lexicon driven approach to handwritten word recognition for real-time applications,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 4, pp. 366–379, Apr. 1997.

[103] A. L. Koerich, R. Sabourin, and C. Y. Suen, “Lexicon-driven HMM decoding for large vocabulary handwriting recognition with multiple character models,” *International Journal on Document Analysis and Recognition*, vol. 6, no. 2, pp. 126–144, Oct. 2003.

[104] J. Vaida and A. Guptab, “Exploring word recognition in a semi-alphabetic script: The case of Devanagari,” *Brain and Language*, vol. 81, no. 1-3, pp. 679–690, Apr. 2002.

[105] L. R. Rabiner, “A tutorial on Hidden Markov Models and selected applications in speech recognition,” *Proc. of IEEE*, vol. 77, no. 2, pp. 257–286, Feb. 1989.

[106] M.-P. Schambach, “Recurrent HMMs and cursive handwriting recognition graphs,” in *Proc. 10th International Conference on Document Analysis and Recognition (ICDAR’09)*, Barcelona, Spain, Jul. 2009, pp. 1146–1150.

[107] G. Salton and M. McGill, *Introduction to Modern Information Retrieval.* New York: McGraw-Hill, 1983.

[108] J. Sivic and A. Zisserman, “Video Google: A text retrieval approach to object matching in videos,” in *Proc. 9th IEEE International Conference on Computer Vision (ICCV’03)*, Nice, France, Oct. 2003, pp. 1470–1477.

[109] L. Fei-fei, R. Fergus, and A. Torralba. Recognizing and learning object categories. [Online]. Available: http://people.csail.mit.edu/torralba/shortCourseRLOC/index.html

[110] T. K. Ho, J. J. Hull, and S. N. Srihari, “Decision combination in multiple classifier systems,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 1, pp. 66–75, Jan. 1994.

[111] A. Bharath and S. Madhvanath, “Hidden Markov Models for online handwritten Tamil word recognition,” in *Proc. 9th International Conference on Document Analysis and Recognition (ICDAR’07)*, Curitiba, Brazil,, Sep. 2007, pp. 506–510.

[112] D. Goldberg and C. Richardson, “Touch-typing with a stylus,” in *Proc. ACM Conference on Human Factors in Computing Systems – INTERCHI*, Amsterdam, The Netherlands, Apr. 1993, pp. 80–87.

[113] I. S. MacKenzie and S. Zhang, “The immediate usability of graffiti,” in *Proc. Graphics Interface*, Kelowna, Canada, May 1997, pp. 129–137.

[114] J. O. Wobbrock, B. A. Myers, and J. A. Kembel, “EdgeWrite: A stylus-based text entry method designed for high accuracy and stability of motion,” in *Proc. ACM Symposium on User Interface Software and Technology (UIST’03)*, Vancouver, British Columbia, Nov. 2003, pp. 61–70.

[115] G. Seni, “TreadMill Ink - Enabling continuous pen input on small devices,” in *Proc. 8th International Workshop on Frontiers in Handwriting Recognition (IWFHR’02)*, Vancouver, British Columbia, Aug. 2002, pp. 215–220.

[116] H. Shimodaira, T. Sudo, M. Nakai, and S. Sagayama, “On-line overlaid-handwriting recognition based on substroke HMMs,” in *Proc. 7th International Conference on Document Analysis and Recognition (ICDAR’03)*, Ontario, Canada, Aug. 2003, pp. 1043–1047.

[117] Y. Tonouchi and A. Kawamura, “Text input system using online overlapped handwriting recognition for mobile devices,” in *Proc. 9th International Conference on Document Analysis and Recognition (ICDAR’07)*, Curitiba, Brazil, Sep. 2007, pp. 754–758.

[118] A. Prasad, A. Prashant, and S. Borgaonkar, “Guided handwriting: Predictive writing input method environment,” HP Labs India, Internal Technical Report, Dec. 2005.

[119] Manish Kumar, “Compact stylus-based input method for Indic scripts,” National Institute of Design, Ahmedabad, India, Diploma Thesis, 2007.

[120] N. K. Srinivas, N. Varghese, and R. K. V. S. Raman, “IndicDasher: A stroke and gesture based input mechanism for Indic scripts,” in *Workshop on Intelligent User Interfaces for Developing Regions (IUI4DR’08)*, Gran Canaria, Spain, Jan. 2008.

[121] D. J. Ward, A. F. Blackwell, and D. J. C. MacKay, “Dasher - a data entry interface using continuous gestures and language models,” in *Proc. 13th Annual ACM Symposium on User Interface Software and Technology (UIST’00)*, San Diego, USA, Nov. 2000, pp. 129–137.

[122] A. Bharath and S. Madhvanath, “FreePad: A novel handwriting-based text input for pen and touch interfaces,” in *Proc. 13th International Conference on Intelligent User Interfaces (IUI’08)*, Gran Canaria, Spain, Jan. 2008, pp. 297–300.

[123] Lipi toolkit. [Online]. Available: http://lipitk.sourceforge.net

Publications and Presentations

Book Chapter

1. Bharath A. and Sriganesh Madhvanath, "Online handwriting recognition for Indic scripts," in *Guide to OCR for Indic scripts*, V. Govindaraju and S. Setlur, Eds. London: Springer, 2009, pp. 209-234.

Journal Papers

1. Bharath A. and Sriganesh Madhvanath, "Constrained stroke clustering for modeling writing styles in online handwriting recognition," Communicated to *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

2. Bharath A. and Sriganesh Madhvanath, "HMM-based lexicon-driven and lexicon-free word recognition for online handwritten Devanagari and Tamil scripts," Communicated to *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Conference Papers

1. Bharath A. and Sriganesh Madhvanath, "A framework based on semi-supervised clustering for discovering unique writing styles," in *Proc. 10th International Conference on Document Analysis and Recognition (ICDAR'09)*, Barcelona, Spain, Jul. 2009, pp. 891-895.

2. Bharath A. and Sriganesh Madhvanath, "Recognition of eyes-free handwriting input for pen and touch Interfaces," in *Proc. 11th International Conference on Frontiers in Handwriting Recognition (ICFHR'08)*, Montreal, Canada, Aug. 2008.

3. Bharath A. and Sriganesh Madhvanath, "FreePad: A novel handwriting-based text input for pen and touch interfaces," in *Proc. 13th International Conference on Intelligent User Interfaces (IUI'08)*, Gran Canaria, Spain, Jan. 2008, pp. 297-300.

4. Bharath A. and Sriganesh Madhvanath, "FreePad: Attention-free handwriting-based text input for pen and touch interfaces," in *Proc. HP TechCon Asia*, Kobe, Japan, Nov. 2007 and *Proc. HP TechCon Global*, Boston, USA, May 2008.

5. Bharath A. and Sriganesh Madhvanath, "Hidden Markov Models for online handwritten Tamil word recognition," in *Proc. 9th International Conference on Document Analysis and Recognition (ICDAR'07)*, Parana, Brazil, Sep. 2007, pp. 506-510.

6. Babu V.J., Prasanth L., Sharma R.R., Rao G.V.P., and Bharath A., "HMM-based online handwriting recognition system for Telugu symbols," in *Proc. 9th International Conference on Document Analysis and Recognition (ICDAR'07)*, Curitiba, Brazil, Sep. 2007.

Patents Filed

1. Handwriting Identification Method, Program and Electronic Device, USPTO, Date of filing: 10 Dec. 2008.

Presentations

1. Online Handwritten Word Recognition for Indic Scripts. Doctoral Symposium in ACM Compute Conference, Bangalore, India, Jan. 2008
2. FreePad: A Novel Handwriting-based Text Input Method for Pen and Touch interfaces. Presented at the DGP Lab, University of Toronto, Toronto, Canada, Aug. 2008. Host: Prof. Ravin Balakrishnan

Candidate's Biography

Bharath A. is a PhD student at Hewlett-Packard Labs India under the BITS - HP Labs PhD Fellowship Programme. Prior to joining the fellowship programme, he did his M.Tech. in Information Technology at the International Institute of Information Technology, Bangalore (IIIT-B) and B.E. in Electronics and Communication Engineering from the University of Madras. His research interests include Pattern Recognition and Machine Learning.

Supervisor's Biography

Dr. Sriganesh Madhvanath is a Senior Research Scientist and the Principal Investigator for the Intuitive Multimodal and Gestural Interaction (IMaGIn) project at HP Labs India. This multidisciplinary project aims to explore intuitive touch and visual gesture-based interaction experiences for personal systems, and create technologies to support them.

Dr. Madhvanath joined HP Labs India in April 2002. At HP Labs, he has led research into handwriting recognition and linguistic resources for Indic scripts, standards such as W3C InkML for platform-neutral representation of digital ink, and pen-based interfaces and solutions relevant to developing nations in spaces such as form filling, text input and collaboration. Some of this work is freely available in the form of two open source toolkits: Lipi Toolkit (lipitk.sourceforge.net) and InkML Toolkit (inkmltk.sourceforge.net).

Prior to joining HP, Dr. Madhvanath was a senior staff engineer with Narus, a startup in Palo Alto, California. Earlier he was Research Staff Member with the Document Analysis and Recognition group at IBM Almaden Research Center, where he researched constraint-driven interpretation of OCR results in the context of systems for forms processing and interpretation of names and addresses for postal and banking applications.

Dr. Madhvanath holds a PhD and MS in Computer Science from the State University of New York at Buffalo and a B.Tech. in Computer Science and Engineering from Indian Institute of Technology, Mumbai. His PhD research was one of the first to investigate the holistic paradigm in offline handwritten word recognition, and apply it to real-world problems such as interpreting handwritten addresses.

Dr. Madhvanath has one granted patent and several others under process. He is also the author of more than 70 publications in international journals, conferences and workshops.

Dr. Madhvanath's research interests are in the general areas of pattern recognition and machine learning. His current research focus is human computer interaction and the design of gestural and multimodal systems.

Dr. Madhvanath is a Senior Member of the IEEE, life member of IAPR and a Member of the ACM. He has also served on the program committees or as a reviewer for various conferences and journals in the areas of handwriting recognition, document analysis and pattern recognition.