USE OF DISTANCE MEASURES IN HANDWRITING ANALYSIS

by

SUNG-HYUK CHA

A dissertation

submitted to the Faculty of the Graduate School

of the State University of New York at Buffalo

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Written under the direction of

Sargur N. Srihari

April, 2001

ABSTRACT OF THE DISSERTATION

Algorithmic analysis of human handwriting has many applications such as in *on-line & off-line handwriting recognition, writer verification*, etc. Each of these tasks involves comparison of different samples of handwriting. To compare two samples of handwriting requires distance measures. In this dissertation, several new and old distance measures appropriate for *handwriting analysis* are given, e.g., element, histogram, probability density function, string, and convex hull distances. Results comparing newly defined histogram and string distance measures with conventional measures are given. We present several theoretical results and describe applications of the methods to the domain of *on-line & off-line character recognition* and *writer verification.*

The theoretical results pertain to *individuality validation.* In classification problems such as *writer, face, finger print* or *speaker identification*, the number of classes is very large or unspecified. To establish the inherent distinctness of the classes, i.e., validate individuality, we transform the many class problem into a dichotomy by using a "distance" between two samples of the same class and those of two different classes. Based on conjectures derived from experimental observations, we present theorems comparing polychotomy in feature domain and dichotomy in distance domain from the view point of tractability vs. accuracy.

The practical application issues include *efficient search, writer identification* and *discovery.* First, fast nearest-neighbor algorithms for distance measures are given. We also discuss designing and analyzing an algorithm for *writer identification* for a known number of writers and its relationship to *handwritten document image indexing and retrieval.* Finally, we present mining a database consisting of writer data and features obtained from a handwriting sample, statistically representative of the US population, for feature evaluation and to determine similarity of a specific group of people.

Committee

Chairman:	Sargur N. Srihari, PhD Distinguished Professor Department of Computer Science and Engineering State University of New York at Buffalo
Members:	Peter D. Scott, PhD Associate Professor Department of Computer Science and Engineering State University of New York at Buffalo
	Ashim Garg, PhD Assistant Professor Department of Computer Science and Engineering State University of New York at Buffalo
External Reader:	Graham Leedham, PhD Associate Professor Division of Computer Engineering Nanyang Technological University
Date Defended:	March 28, 2001

Preface

Preparing for a thesis regarding distances and their practical usages was definitely not an easy task. Thesis would not be complete without a whole lot of knowledge in *Artificial Intelligence* and *Pattern Recognition*. Along with taking many AI related courses, I also perfected materials as a teaching assistant. I would like to delineate some courses that affected and assisted this thesis greatly: *Artificial Intelligence, Computational Vision, advanced techniques in AI, Analysis of Algorithm, Database Systems, Computer Vision and Image Processing, Pattern Recognition, Computational Geometry, Image Analysis, Document Analysis, Machine Learning, Data Mining, Design and Analysis of Experiment,* and *Information Theory*.

Starting from my first semester at UB, I was involved, as a graduate research assistant, in the project called *Handwriting Individuality Validation Study*. I took a part in preparing the proposal to *NIJ (National Institute of Justice)* for the grant in 1998, writing the progress(1999) and final reports(2000), and submitting the continuation proposal (2001). Since it is a collaborated work [91] with many other people, I have not chosen it as my dissertation topic, yet this dissertation discusses a lot about the handwriting individuality. Hence, I would like to give a sketchy outline of the project.

Handwriting is considered to be the talisman of the individual [103]. For hundreds of years handwriting has been used to signify assent in legal documents. Yet to this day there is no definitive work that quantifies the individual uniqueness of handwriting. While such a study has scientific interest it has also become necessary to perform such a study in view of several rulings in US courts (Daubert vs. Merrell

Dow Pharmaceuticals, etc.) that require the presence of accepted scientific studies before presenting related evidence in court. Our group at the Center of Excellence for Document Analysis and Recognition (CEDAR) at the State University of New York at Buffalo has been conducting a study on the individuality of handwriting since mid-1999. Chapter 2 discusses the *dichotomy model* to establish the individuality in handwriting using distance measures and a procedure for comparing handwritten items.

This thesis also presents two important applications: *on-* and *off-line character recognition*. At the heart of research in *Character Recognition* lies the hypothesis that feature sets can be designed to extract certain types of information from the image. Another important issue is pattern matching which exploits the similarity or distance measure between feature patterns. *"The distance is nothing; it is only the first step that is difficult."* [1] Most of chapters 3 through 6 of this document are dedicated to introduce a number of distance measures.

There are over 20 journal, proceedings, book chapters and technical report publications related to this dissertation which can be divided into the six main chapters (two through seven). Albeit all chapters are organized to demonstrate the usefulness of the distance measures in *handwriting analysis*, each chapter is self-explanatory having its own introduction and conclusion. Readers may read some chapters of interest without reading the entire dissertation.

---

[1] Madame du Deffand (1697-1780), Letter to d'Alembert, 7 July 1763

Acknowledgements

I extend my sincere thanks to Dr. Sargur N. Srihari for his help during the development stages of the thesis and giving me an opportunity to work in the highly competent environment at CEDAR (Center of Excellence for Document Analysis and Recognition). I am grateful to Dr. Peter D. Scott and Dr. Ashim Garg for their suggestions and encouragement throughout my graduate studies.

This dissertation has been possible funded by National Institute of Justice (NIJ) in response to the solicitation entitled *Forensic Document Examination Validation Studies*: Award Number 1999-IJ-CX-K010 [105]. I'd like to thank Dr. Richard Rau who is the NIJ program manager.

Thanks are also due to many anonymous reviewers for making helpful comments as many parts of the thesis have been reviewed for publications in various journals and conference proceedings. Especially, I would like to thank the NIJ review panel members for the careful and invaluable critiques.

I also would like to thank Hina Arora and Eugenia Smith for collecting handwriting samples and Pradeep SaganeGowda for scanning the document images. Many thanks are due to summer research assistants, Bang S Jeong, Heybhin Kim, Jihyung Kim, Hyoungjoon Jeon, and Sanghee Lee who assisted greatly for the CEDAR letter database construction. And finally Denise P. Mak for implementing the web-based document image retrieval system.

I would like to thank two fellow graduate students who have supported and assisted me throughout my entire graduate years in UB. They are Sangjik Lee and Srikanth Munirathnam. Many ideas in this dissertation are due to the invaluable discussions with them.

And finally, I should like to thank my parents for support and love.

Dedication

This dissertation is dedicated to my family for their love and affection and CEDAR

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ABT is for Additive Binary Tree.

ANN is for Artificial Neural Network.

CEDAR is for the Center of Excellence for Document Analysis and Recognition.

CDSS is for Contour Direction Sequence String.

DWDP is for Different Writer Document Pair.

EUV is for Experimental Unit Variable.

GSC is for Gradient, Structural, and Concavity.

indels are for insertion and deletion.

KNN is for k-Nearest Neighbor.

MDPA is for Minimum Difference of Pair Assignments.

OCR is for Optical Character Recognition.

OUV is for Observational Unit Variable.

PDF is for Probability Density Function.

QD is for Questioned Document.

SDSS is for Stroke Direction Sequence String.

SPSS is for Stroke Pressure Sequence String.

SWDP is for Same Writer Document Pair.

Chapter 1

Introduction

The objective of *handwritten document analysis* is to recognize the contents in document images and to extract the intended information as a human would. Not surprisingly, it has received a great deal of attention because of its various practical applications [70, 74, 82, 72, 102, 80, 99]. Forensic literature is extensive as it pertains to handwriting [57, 6, 80]. The *analysis of handwriting* crops up in many diverse applications as given in Figure 1.1. While several types of algorithmic analysis can

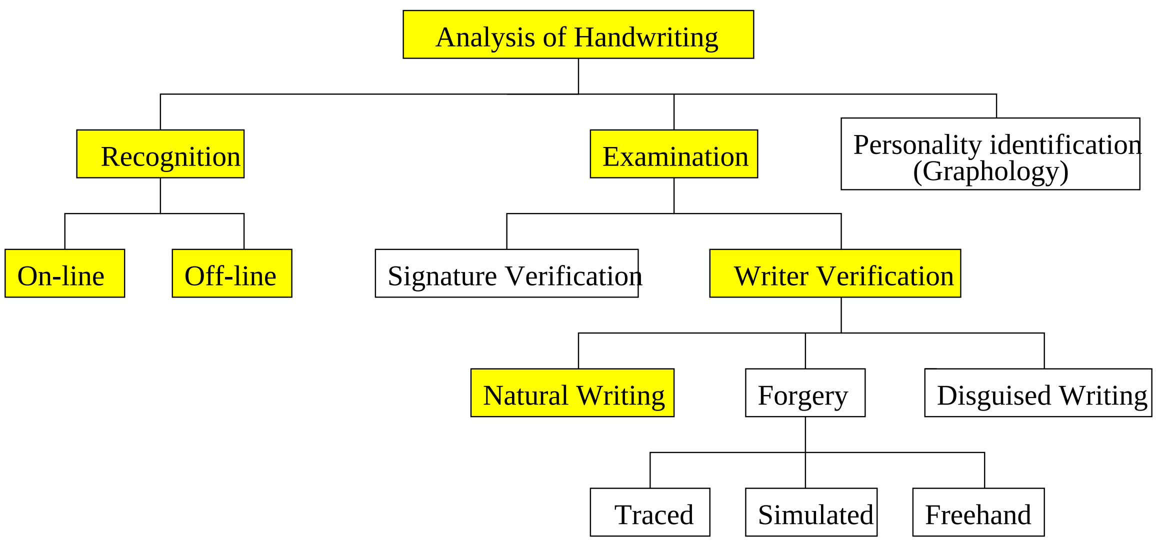


Figure 1.1: Taxonomy of topics in handwriting analysis.

be associated with human handwriting, the path that leads to the subject areas of this dissertation: *on-line* and *off-line handwriting recognition* and *writer verification* based on *natural writing*, is shown in light grey of Figure 1.1. *Handwriting recognition* is the task of transforming a language represented in its spatial (off-line) and temporal (on-line) form of graphical marks into its symbolic representation. *Writer verification* is a process to compare questioned handwriting with samples of handwriting obtained from known sources for the purposes of determining authorship or non-authorship [6]. Each of these applications involves comparison of different samples of handwriting and the algorithmic comparison requires distance measures. In this formalization, these problems fall under the aegis of *pattern classification*.

1.1 Problem Statement

This dissertation is to study the use of distance measures in feature space for the purpose of *handwriting analysis*. Specifically, we deal with following issues: *individuality validation*, *designing distance measures*, *efficient search* and *discovery*.

1.1.1 Individuality Validation

The primary objective of the first part of this dissertation is to determine the scientific validity of individuality in handwriting. In other words, we would like to show that everybody writes differently. Individuality in handwriting can be illustrated as in Fig. 1.2. In this example, eight authors provide three handwriting samples each of the word "referred". As can be seen, the variation within a person's handwriting (*within-author variation*) is less than the variation between the handwriting of two different people (*between-author variation*). Thus, the problem of validating the individuality in handwriting can be modeled and formalized by defining a distance metric between samples of handwritings and finding the optimal threshold value to discriminate the within and between author variations.

Figure 1.2: Variability in handwriting. Eight authors provided three handwriting samples each, showing within and between author variations.

1.1.2 Designing Distance Measures

In classification problems, the classification rates depends significantly on distance measures. We consider the problem of how to design distance measures and evaluate distance measures. Speaking informally, the problem can be that modifying or varying conventional measures to improve the classification rates for various handwriting analysis applications. Another way of stating the problem is showing the various invariant properties of distance measures. Another issue regarding designing distance measures is the integration of heterogeneous features. Distance measures must be sound to measure theory and consider the various types of features. We carefully consider the numerous types of features, measurements made on them, and integrating the multiple type features for the purpose of *handwriting analysis*.

1.1.3 Efficient Search

Classifying an unknown input is a fundamental problem in *Pattern Recognition*. One standard method is finding its nearest neighbors in a reference set. The nearest neighbor classification technique pertains to distance measures. It would be very time consuming if distances are computed feature by feature for all templates in the reference set; this naïve method is $O(nd)$ where n is the number of templates in the reference set and d is the number of features or dimension. We consider the problem of designing the fast nearest neighbor search algorithms.

1.1.4 Discovery

Finally, we consider the problem of mining a database consisting of writer data and features obtained from a handwriting sample, statistically representative of the US population to determine similarity of a specific group of people. The sub-category classification problem is that of discriminating a pattern to all possible sub-categories. In other words, it is to find any trend of pattern in a specific sub-category.

Albeit the data mining issue is not deeply related to the core topic of this dissertation, we include it here because the sub-category classification problem is a vicissitude of pattern classification problem and it provides the attractive fructescence of the handwriting document database collected for this dissertation. Trends in handwriting are invaluable information to handwriting analysts.

1.2 Historical Background

1.2.1 Individuality Validation

The analysis of people's handwriting, and the identification of authors of suspect handwritten documents has great bearing on the criminal justice system. One of the basic foundations on which the entire subject of handwriting identification rests is that each individual's handwriting is consistent and is distinguishable from another individual's handwriting.

Since the writer identification plays an important investigative and forensic role in many types of crime, various automatic writer identification by computer techniques, feature extraction, comparison and performance evaluation methods have been studied (see [80] for the extensive survey). Osborn suggested a statistical basis to handwriting examination by the application of the *Newcomb rule of probability* and Bertillon was the first who used the Bayesian theorem to handwriting examination [57]. Hilton calculated the *odds* by taking the likelihood ratio statistic that is the ratio of the probability calculated on the basis of the similarities, under the assumption of identity, to the probability calculated on the basis of dissimilarities, under the assumption of non-identity [57, 54]. However, relatively little study has been carried out to demonstrate its scientific and statistical validity and reliability as forensic evidence. To identify writers, it is necessary to determine the statistical validity of individuality in handwriting based on measurement of features, quantification, and statistical analysis.

Osborn suggested a statistical basis to handwriting examination by the application of the *Newcomb rule of probability* and Bertillon was the first who used the Bayesian theorem to handwriting examination [57]. Hilton calculated the *odds* by taking the likelihood ratio statistic that is the ratio of the probability calculated on the basis of the similarities, under the assumption of identity, to the probability calculated on the basis of dissimilarities, under the assumption of non-identity [57, 54]. However, relatively little study has been carried out to demonstrate its scientific and statistical validity and reliability as forensic evidence. For this reason, we propose a model to validate the individuality of handwriting.

Consider the multiple class problem where the number of classes is small and one can observe enough instances of each class. To show the individuality of class statistically, one can cluster instances into classes and infer it to the population. It is an easy and valid setup to establish the individuality as long as a substantial number of instances for every class are observable. Now consider the *many class problem* where the number of classes is too large to be observed; n is very large or often the united state population. Most pattern identification problems such as *writer, face, fingerprint* or *speaker identification* fall under the aegis of the many class problem. Most parametric or non-parametric multiple classification techniques are of no use to validate the individuality of classes and the problem is seemingly insurmountable because the number of classes is too large or unspecified. Nonetheless, most of studies use the *writer identification* model which is the many class problem measuring the confusion matrix [61, 60, 49, 55, 50] (see [80] for the extensive survey on the writer identification).

Particularly problematic in this regard is the fact that there exists neither true standard nor universal definition of similarity. Interaction with questioned document is generally personal, subjective and limited to some degree by geography. Given the nature of the assessment, the improvement in it might be realized if the process were less subjective. The FBI formed the Technical Working Group on Forensic Document Examination (TWGDOC) in May 1997 and the importance of standardizing procedures for handwriting comparison was recognized as a primary task [105]. Such procedures must be based on more than community-based agreement. Procedures must be tested statistically in order to demonstrate that following the stated procedures allows analysts to produce correct results with acceptable error rates. This has not yet been done. For this reason, we propose an algorithmic objective approach for the analysis part of the procedure.

1.2.2 Designing Distance Measures

Classifying an unknown input is an important problem in *Pattern Analysis & Recognition*. The fundamental idea underlying pattern recognition using distance measures is based on finding the most or top k similar vectors in a reference set. The k-nearest neighbor or simply *k-nn* has wide acceptance in Character Recognition (see [32, 33] for extensive surveys). There are two important goals in this approach. One is selecting important features from a character image. The other is selecting an appropriate similarity measure. There exist several definitions encountered in various fields such as information retrieval and biological taxonomy [35]. Common definitions include *Euclidean, Minkowski, cosine, dot product, Tanimoto distance*, etc. As illustrated

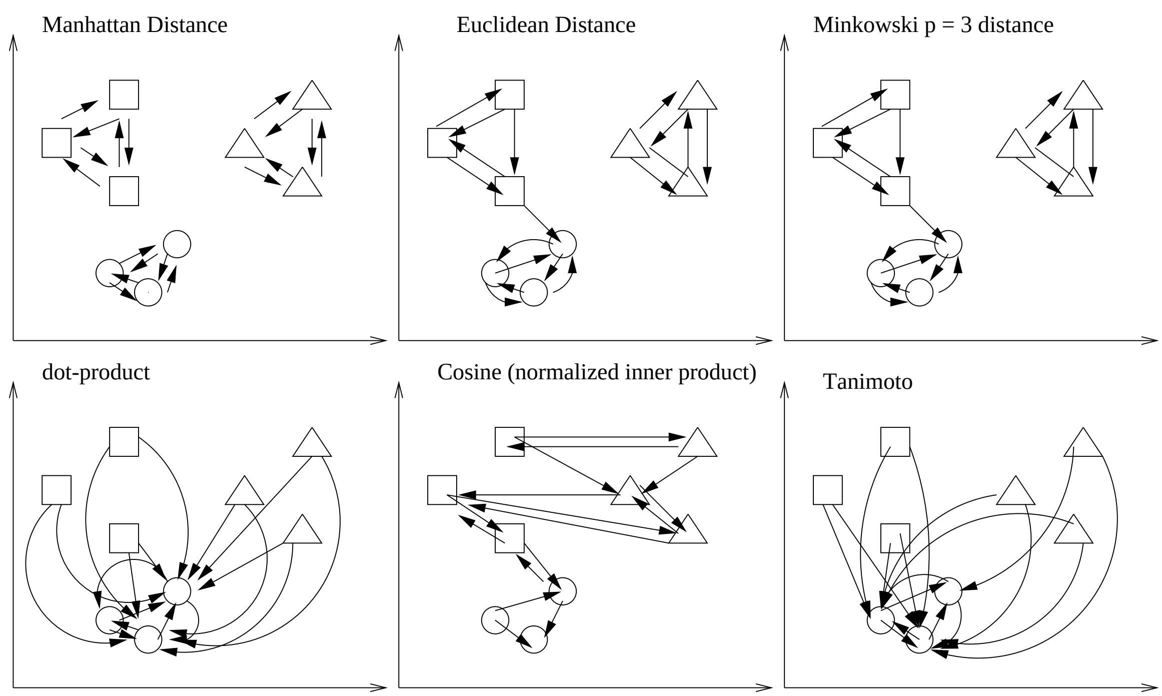


Figure 1.3: *All k-nearest neighbor graph where $k = 2$.*

in Figure 1.3, classification depends largely on distance or similarity measures as neighbors are different depending on distance measures. Therefore, it is important to choose a suitable distance measure. In an attempt to answer this question, this dissertation considers various distance measures and proposes new alternative distance measures considering the data type and measure theory. In addition, this thesis animadverts on some conventional distance measures used for the histogram and string comparisons.

Histogram

A distance measure between two histograms has applications in feature selection, image indexing and retrieval, pattern classification and clustering, etc. In the history of distance between two histograms, there are two methodologies in histogram distance measures: vector and probabilistic. In the vector approach, a histogram is treated as a fixed-dimensional vector. Hence standard vector norms such as *city block, Euclidean* or *intersection* can be used as distance measures. Vector measures between histograms have been used in image indexing and retrieval [43, 78, 84, 101].

The probabilistic approach is based on the fact that a histogram of a measurement provides the basis for an empirical estimate of the probability density function (pdf) [35]. Computing the distance between two pdf's can be regarded as the same as computing the Bayes (or minimum misclassification) probability. This is equivalent to measuring the overlap between two pdf's as the distance. There is much literature regarding the distance between pdf's, an early one being the *Bhattacharyya* distance or *B-distance* measure between statistical populations [58]. The *B-distance*, which is a value between 0 and 1 provides bounds on the Bayes misclassification probability. An approach closely related to the *B-distance* was proposed by Matusita [67, 28]. Kullback and Leibler [63] generalized Shannon's concept of probabilistic uncertainty or *"entropy"* [86] and introduced the *"K-L distance"* [36, 87] measure that is the minimum cross entropy (see [104] for an extensive bibliography on estimation of misclassification).

String

In measuring distance between strings, the approximate string matching algorithm is often used to discriminate between two patterns and it is one of widely studied areas in computer science due to a variety of applications such as genetics and DNA sequence analysis, a spelling corrector, etc [90, 98, 36]. Earlier definition and solution for the traditional *approximate string matching* problem are found in the literature [108, 98] and extensive surveys on various techniques are shown in [51]. It computes the *edit distance*, also known as *Levenshtein distance* that is the minimum number of *indels* (insertions and deletions) and substitutions needed to transform one string to another.

"Find all letters that look like this letter." Such a query has received a great deal of attention in *character recognition* and *handwriting analysis*. This problem has been formalized by defining a distance metric between characters and finding the nearest characters from the reference set. One promising distance measure is using the edit distance between strings after extracting the on-line stroke and off-line contour sequence strings. This approach has been studied by many researchers [66, 77, 14, 15] dating as far back as 1975 [45]. *Fujimoto et al* developed the *OCR* using the idea of *"Nonlinear Elastic Matching"* to read hand-printed alpha-numerics and Fortran programs [45]. The edit distance with cost matrix technique [108] was used to solve the on-line [77] and off-line [66] character recognition problems.

1.2.3 Efficient Search

One straight-forward method is computing feature by feature for all templates in the reference set and it takes is $O(nd)$ where n is the number of templates in the reference set and d is the number of features or dimension. This is very time consuming for users to wait for the output. Hence, there is a wealth of literature regarding computational expenses of the *KNN* problem dating as far back as 1970. Papadimitiou and Bentley showed $O(n^{1/d})$ worst-case algorithm [76] and Friedman, Bentley, and Finkel suggested possible $O(\log n)$ expected time algorithm [44]. There are two main streams of implementing a fast algorithm: lossy and lossless search algorithms. There are three general algorithmic techniques for reducing the computational burden: computing partial distance, pre-structuring, and editing the stored prototypes [36].

Partial distance:

First, the partial distance technique is often called a *sequential decision technique*; decision for match between two vectors can be made before all features in the vector are examined. It requires a predetermined threshold value to reduce computation time.

Pre-structuring:

The most famous method focuses on preprocessing the prototype set into certain well-organized structures for the fast classification processing. Many approaches utilizing *multidimensional search trees* that partition the space appear in the literature [46, 62, 71, 7]. In these approaches, the range of each feature must be large. Otherwise, if features are binary, we achieve little speedup. Furthermore, the dimension of feature space must be low. Quite often in image pattern recognition, each feature is thresholded and binary and the dimension is high.

A different type of preprocessing on the prototypes has been introduced to generate a useful information that helps reduce the overall search time. As a result of the preprocessing, a metric can be built. In a study of utilizing the metric, Vidal *et al* [107] claimed that the approximately constant average time complexity is achieved only by the metric properties. Although it was their claim, what has been shown is that the average number of prototypes necessary for feature by feature comparison is constant [40]. It is $\Omega(d + n)$ on average and even $O(n^2 + nd)$ in the worst case. In some applications, this approach is quite prohibitive as it requires $O(n^2)$ space and the number of templates is often too big.

Editing the stored prototypes:

Another important approach is the prototype reduction method. It reduces the size of prototype set to improve speed with sacrifice of accuracy. The condensed nearest neighbor rule [53] and the reduced nearest neighbor rule [47] are used to select a subset of training samples to be the prototype set. In this approach, we must sacrifice accuracy for speed. Hong *et al* [56] successfully implemented a fast nearest neighbor classifier for the use of Japanese Character Recognition. They combined a non-iterative method for *CNN* and *RNN* and a hierarchical prototype organization method to achieve a great speed-up with a little accuracy drop.

1.2.4 Discovery

Data mining is a very broad area. Only data mining algorithm we study in this dissertation is *Apriori Algorithm* originally designed for the purpose of efficient association rule mining by *Agrawal et al* [3, 2]. The concept of association rules was introduced in 1993 [1] and many researchers have endeavored to improve the performance of algorithms that discover the association rules in large datasets. The *Apriori algorithm* is an efficient association discovery algorithm that filters item sets by incorporating item constraints (support).

1.3 Proposed Model and Solutions

1.3.1 Individuality Validation

Consider the *many class problem* where the number of classes is too large to be observed (n is very large). Most pattern identification problems such as *writer, face,* *fingerprint* or *speaker identification* fall under the aegis of the many class problem. Most parametric or non-parametric multiple classification techniques are of no use and the problem is seemingly insurmountable because the number of classes is too large or unspecified. In this dissertation, the *writer identification* problem is used as an illustrative example. *Writer identification* systems are often developed using a small and finite number of classes drawn from the entire class set as shown in Figure 1.4. However, showing the clusterability in the subset of classes does not provide the validity of individuality of handwriting. Without the validity of individuality, it is meaningless to design a *writer identification* system. A problem that arises with the *writer identification* model is that of statistical inferentiability; the result based on this model cannot be inferred to the entire population. For this reason, this thesis introduces a *dichotomy* model to establish the inherent distinctness of the classes, deferring consideration of *writer identification model* also know as *polychotomy* to another technical report [91].

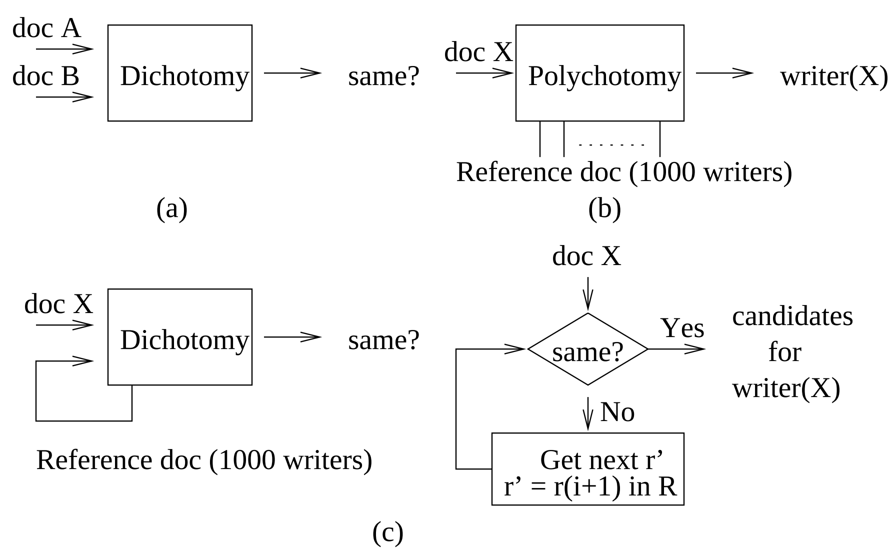


Figure 1.4: (a) Validation of Individuality model (b) Polychotomy model for writer identification (c) Dichotomy model for writer identification.

We use a simple *dichotomy* model that is a classifier that places a pattern in one of only two categories [25, 24] to establish the individuality as shown in Fig. 1.4

(a). We model the problem as a two class classification problem: authorship or non-authorship. Given two handwriting samples, the distance between two documents is first computed. This distance value is used as data to be classified as positive (authorship, inner-variation, within author or identity) or negative (non-authorship, intra-variation, between different authors or non-identity). We use within author distance and between authors distance and subscriptions of the positive ($\oplus$) and negative ($\ominus$) symbols as the nomenclature for all variables of within author distance and between authors distance, respectively. In this model, 96% accuracy performance has been observed with 152 writers with three sample documents per writer, using 5 feature distances.

Particularly problematic in this regard is the fact that there exists neither true standard nor universal definition of comparison. For this reason, we propose an algorithmic objective approach. Using visual information of two or more digitally scanned handwritten items, we show a method to access the authorship confidence that is the probability of errors. Instead of building a costly handwritten item database to support the confidence, *questioned words* are simulated from the *CEDAR letter image database* in order to handle any handwritten items. An *Artificial Neural Network* is trained to verify the authorship using the synthesized words.

One advantage of *dichotomy* model is multiple type feature integration. Features encountered in various pattern recognition problems can be diverse in type. Both continuous and non-continuous features have been studied widely in *pattern recognition* [36], *machine learning* [68] and *feature selection* [65] areas. In Liu and Motoda's version of the hierarchy of feature types [65], only elementary feature types were considered: discrete ordinal and nominal, continuous, and complex. Features observed in real application such as the *writer identification* have much more complicated feature types than these elementary feature types. Various types of features are shown in Fig. 1.5 and we integrate them into one useful for the *writer identification* problem [21].

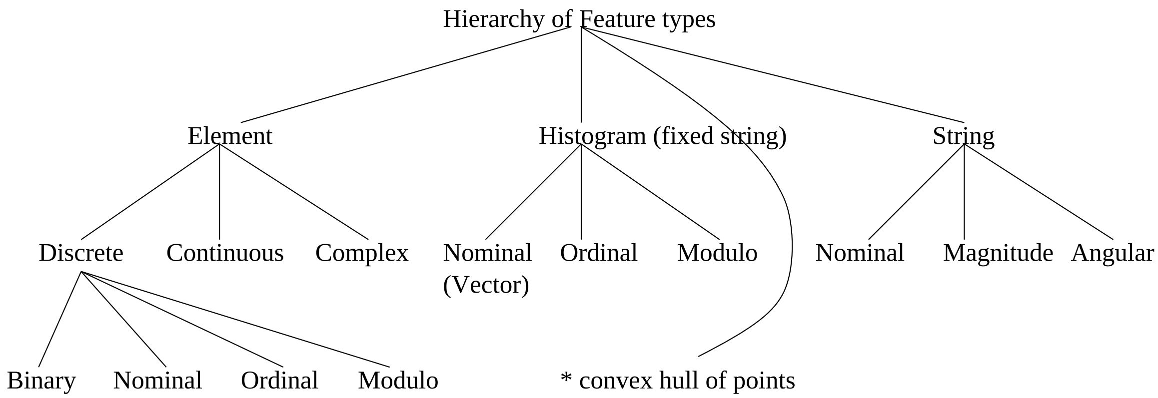


Figure 1.5: *Hierarchy of Feature Types.*

1.3.2 Designing Distance Measures

In the feature distance domain (dichotomy model), all feature distance types are nothing but scalar values and homogeneous regardless of their feature types. In other words, multiple type features are integrated into the feature distance scalar values to solve the writer identification problem. Clearly, the performance depends largely on the distance measure for each homogeneous feature. In this dissertation, we introduce previously and newly defined various distance measures and their algorithms for many feature types: element, convex hull [18], histogram [23, 19, 13, 12], string [16, 14, 15].

Histogram

The viewpoint of regarding the overlap (or intersection) between two histograms as the distance has the disadvantage that it does not take into account the similarity of the non-overlapping parts of the two distributions. For this reason, we present a new definition of the distance for each type of histograms. The new measure uses the notion of the *Minimum Difference of Pair Assignments*. We propose a distance between sets of measurement values as a measure of dissimilarity of two histograms. Three versions of the distance measure, corresponding to whether the type of measurement is nominal, ordinal, and modulo, are given. The advantage of using this distance measure over distance measures found in the clustering and pattern recognition literature are given both with examples and with theoretical justification. While computing a new distance measure has exponential time complexity in general, we show efficient algorithms for computing the distance between two univariate histograms: $\Theta(b)$, $\Theta(b)$ and $O(b^2)$ for each type of measurements, respectively, where b is the number of levels in histograms.

String

String distance measures are useful in both on-line and off-line character recognition for comparing on-line stroke and off-line contour sequence strings. Since stroke and contour string elements are angular in that they represent a circular measurement ($0° \sim 360°$), usual *edit distances* with cost matrix are inadequate for this type of strings. For this reason, we extend *edit distances*, previously defined for nominal type, to handle angular and magnitude types. The newly defined measure utilizes the *"turn"* concept in place of *substitution* for angular string elements and takes local context into account in *indels* (insertions and deletions). The approximate string matching, besides being of interest in itself, provides solutions to *writer verification, on-line* and *off-line character recognition* problems. We also discuss the string concatenation and reverse operations to recognize on-line characters when written unnaturally.

Auxiliary Distance Measures

Gradient, Structural and Concavity features are regarded very important features and GSC classifier gives the best digit recognition performance among other currently used classifiers. In this paper, we present a technique to evaluate similarity measures using the error vs. reject percentage graph and find a new similarity measure for a compound feature: GSC features. Since the optimized similarity measure performs better on a different testing set than the previously used similarity measure, we claim that an improvement in off-line Character Recognition is achieved.

Second, we present a prototypical convex hull discriminant function. As an output, the program gives the geometrical significances of an unknown input to all classes and helps determine its possible class. This technique is particularly useful in the *Writer Identification* problem, in which the number of samples is limited and very small. Convex hulls of all samples in each document in the reference set are computed as one's style of handwriting during a preprocessing. During the query classification process, for all samples in the query document, the average distances to the convex hull of each reference document are computed. The author of the document whose average distance is the smallest or within a certain threshold value is considered as a candidate for possible author of the query document.

1.3.3 Efficient Search

The handwriting classification problem has been formalized by defining a distance metric between two writings and finding all writings which are within dichotomy threshold for every feature. Computing the distance for all templates in the reference set would be very time consuming for users to wait for the output. A threshold value may be used to reduce computation time such as the *sequential decision technique*; decision for match between two vectors can be made before all features in the vector are examined. We present a technique to speed up further than that.

The new algorithm utilizes both partial distance and prestructuring techniques [20, 22]. We reduce computation time by using an *Additive Binary Tree (ABT)* data structure that contains additive information which is frequency information in binary features. The idea behind the *ABT* approach in finding the nearest neighbor is filtration by which the unnecessary computation can be eliminated. It makes this approach unique from the others such as redundancy reduction or metric method. First, take a quick glance at the reference set and select candidates for match. Next, take a harder look only at those candidates selected from previous filtration to select a fewer candidates, and so on. After several filtration, take a complete thorough look only at the final candidates to verify them. All matches whose distance is less than or equal to a threshold are guaranteed to be in all candidate sets.

1.3.4 Discovery

The sub-category classification problem is that of discriminating a pattern to all sub-categories. Not surprisingly, sub-category classification performance estimates are useful information to mine as many researchers are interested in any trend of pattern in specific sub-category. This chapter presents a data mining technique to mine a database consisting of experimental and observational unit variables. Experimental unit variables are those attributes which make sub-categories of the entity, e.g., patient personal information or a person's identity and observational unit variables are features observed to classify the entity, e.g., test results or handwriting styles, etc. Since there are an enormously large number of sub-categories based on the experimental unit variables, we apply the *Apriori algorithm* to select only sub-categories that have enough support among all possible ones in a given database. Those selected sub-categories are then discriminated using observational unit variables as input features to the *Artificial Neural Network (ANN)* classifier. The importance of this paper is twofold. First, we propose an algorithm that quickly selects all sub-categories that have enough both support and classification rate. Second, we successfully applied the proposed algorithm to the field of handwriting analysis. The task is to determine similarity of handwriting style of a specific group of people. Document examiners are interested in trends in the handwriting of specific groups, e.g., (i) does a male write differently from a female? (ii) can we tell the difference in handwriting of age group between 25 and 45 from others?, etc. Subgroups of white males in the age group 15-24 and white females in the age group 45-64 show 87 % correct classification performance.

1.4 Organization

The subsequent chapters of this dissertation is organized as follows. Chapter 2 presents the transformation to dichotomy from polychotomy to validate the individuality in handwriting. The procedure for comparing handwritten items with a measure of confidence is given. Chapter 3 proposes a distance between sets of measurement values as a measure of dissimilarity of two histograms. Three versions of the distance measure, corresponding to whether the type of measurement is nominal, ordinal, and modulo, are given. In Chapter 4, we extend *edit distances* to handle three measurement types: nominal, angular, and magnitude. The newly defined measure utilizes the *"turn"* concept in place of *substitution* for angular string elements and takes local context (momentum) into account in *indels* (insertions and deletions). The approximate string matching, besides being of interest in itself, provides solutions to *writer identification, on-line* and *off-line character recognition* problems. In Chapter 5, two distance measures are discussed: compound feature distance and convex hull distance. First, we present a technique to evaluate similarity measures using the error vs. reject percentage graph and find a new similarity measure for a compound feature: GSC features. Next, we present a prototypical convex hull discriminant function. As an output, the program gives the geometrical significances of an unknown input to all classes and helps determine its possible class. Chapter 6 presents a fast nearest neighbor search algorithm by filtration. *Additive binary tree* data structure is used. We present a technique for quickly eliminating most templates from consideration as possible neighbors. The remaining candidate templates are then evaluated feature by feature against the query vector. Chapter 7 presents a data mining technique to mine a database consisting of experimental and observational unit variables. The sub-category classification problem is that of discriminating a pattern to all sub-categories. Not surprisingly, sub-category classification performance estimates are useful information to mine as many researchers are interested in any trend of pattern in specific sub-category. Finally, Chapter 8 concludes this dissertation with future work.

Chapter 2

Individuality Validation and Procedure to Compare Handwritings

[1] The *Writer Verification* is a process to compare questioned handwriting with samples of handwriting obtained from known sources for the purposes of determining authorship or non-authorship. Since it plays an important investigative and forensic role in many types of crime, it is necessary to determine the statistical validity of individuality in handwriting based on measurement of features, quantification and statistical analysis. It is enigmatic to distinguish every writer because the number of class (writer) is very large or unspecified in classification problems such as *writer, face, finger print* or *speaker identification.* To establish the inherent distinctness of the classes, i.e., validate individuality, we transform the many class problem into a dichotomy by using a "distance" between two samples of the same class and those of two different classes. A measure of confidence is associated with individuality.

Techniques in *pattern recognition* typically require that features be homogeneous. The solution proposed overcomes the non-homogeneity of features as multiple type features are integrated into the feature distance scalar values to solve the *writer verification* problem. Using ten feature distance values, we trained an *artificial neural network* as a *dichotomizer* and obtained 97% overall correctness. In this experiment, $1,000$ people provided three sample handwritings.

---

[1]This chapter contains works published in [25, 21, 17, 24, 26] and is in preparation for journals.

2.1 Introduction

The *Writer Verification problem* is a process to compare questioned handwriting with samples of handwriting obtained from known sources for the purposes of determining authorship or non-authorship. In other words, it is the examination of the design, shape and structure of handwriting to determine the authorship of given handwriting samples. Document examiners or handwriting analysis practitioners find important features to characterize individual handwriting as features are consistent with writers in normal undisguised handwriting [6]. Authorship may be determined due to the following hypothesis that people's handwritings are as distinctly different from one another as their individual natures, as their own finger prints. It is believed that no two people write the exact same thing the exact same way.

Since the *document examination* plays an important investigative and forensic role in many types of crime, various automatic writer identification by computer techniques, feature extraction, comparison and performance evaluation methods have been studied (see [80] for the extensive survey). However, since the seminal ruling in United States v. Starzecpyzel [97], the judicial system has challenged the *Forensic Document Examination*, *FDE* in short, especially handwriting identification, to demonstrate its scientific validity and reliability as forensic evidence. If handwriting identification fails to meet standards for admissibility of scientific evidence [34], its investigative role may continue, but its role in the courts will be further diminished. Therefore, it is necessary to determine the statistical validity of individuality in handwriting based on measurement of features, quantification, and statistical analysis.

Osborn suggested a statistical basis to handwriting examination by the application of the *Newcomb rule of probability* and Bertillon was the first who used the Bayesian theorem to handwriting examination [57]. Hilton calculated the *odds* by taking the likelihood ratio statistic that is the ratio of the probability calculated on the basis of the similarities, under the assumption of identity, to the probability calculated on the basis of dissimilarities, under the assumption of non-identity [57, 54]. However, relatively little study has been carried out to demonstrate its scientific and statistical validity and reliability as forensic evidence. For this reason, we propose a model to validate the individuality of handwriting.

Consider the multiple class problem where the number of classes is small and one can observe enough instances of each class. To show the individuality of class statistically, one can cluster instances into classes and infer it to the population. It is an easy and valid setup to establish the individuality as long as a substantial number of instances for every class are observable. Now consider the *many class problem* where the number of classes is too large to be observed; n is very large or often the united state population. Most pattern identification problems such as *writer, face, fingerprint* or *speaker identification* fall under the aegis of the many class problem. Most parametric or non-parametric multiple classification techniques are of no use to validate the individuality of classes and the problem is seemingly insurmountable because the number of classes is too large or unspecified. Nonetherless, most of studies use the *writer identification* model which is the many class problem measuring the confusion matrix [61, 60, 49, 55, 50] (see [80] for the extensive survery on the writer identification).

To establish the inherent distinctness of the classes, i.e., validate individuality, we transform the many class problem into a dichotomy by using a "distance" between two samples of the same class and those of two different classes. We tackle the problem by defining a distance metric between two writings and finding all writings which are within the threshold for every feature. In this model, one need not observe all classes, yet it allows the classification of patterns. It is a method for measuring the reliability of classification about the entire classes based on information obtained from a small sample of classes drawn from the class population. In this model, two patterns are categorized into one of only two classes; they are either from the same class or from the two different classes. Given two handwriting samples, the distance between two documents is first computed. This distance value is used as data to be classified as positive (authorship, inner-variation, within author or identity) or negative (non-authorship, intra-variation, between different authors or non-identity). We use within author distance and between authors distance throughout the rest of this paper. Also, we use subscriptions of the positive ($\oplus$) and negative ($\ominus$) symbols as the nomenclature for all variables of within author distance and between authors distance, respectively.

Techniques in *pattern recognition* typically require that features be homogeneous [36]. Both continuous and non-continuous features have been studied widely in *pattern recognition* [36], *machine learning* and *feature selection* [65] areas. In Liu and Motoda's version of the hierarchy of feature types [65], only elementary feature types were considered: discrete ordinal and nominal, continuous, and complex. Features observed in real application such as the *writer identification* have much more complicated feature types than these elementary feature types. We integrate various types of features into one useful for the *writer identification* problem. In all, the proposed *dichotomy* model overcomes the non-homogeneity of features as multiple type features are integrated into the feature distance scalar values to solve the *writer verification* problem.

The subsequent sections are constructed as follows. The section 2.2 discusses the dichotomy transformation and the section 2.3 compares it with the polychotomy model. The section 2.4 shows the experimental database of writer, exemplar and features. In section 2.5, the full statistical analysis of the collected database and gives the experimental results using the *Artificial Neural Network* as a *dichotomizer*. Finally, the section 2.7 concludes the paper.

2.2 Dichotomy Transformation

The problem can be viewed as a U.S. population category classification problem, so called *Polychotomizer*. It is stated as follows. There are m writing exemplars of each of n people (n = very large). Given a writing exemplar, x, of an unknown writer, the task is to determine whether x was written by any of the n writers and if so, identify the writer. As the number of classes is enormously large and almost infinite, this problem is seemingly insurmountable. We propose a *dichotomy model* that can handle the *many class problem*. In this section, we show how to transform a large *polychotomy* problem to a simple *dichotomy* problem, a classification problem that places a pattern in one of only two categories.

To illustrate, suppose there are three writers, $\{W_1, W_2, W_3\}$. Each writer provides three handwritten documents and two scalar value features extracted per document. Figure 2.1 (a) shows the plot of documents for every writer. To transform into distance

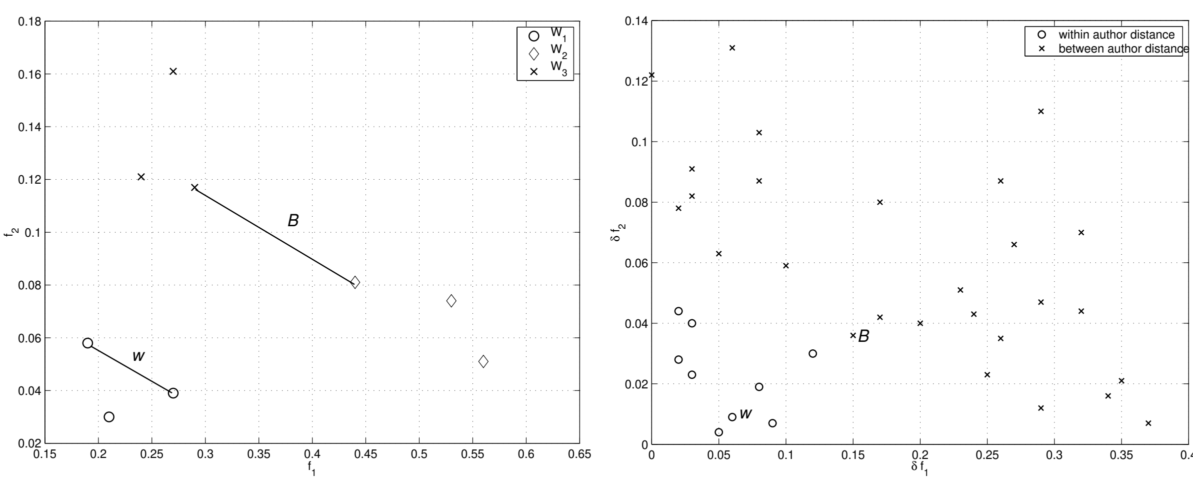


Figure 2.1: Transformation from (a) Feature domain (*polychotomy*) to (b) Feature distance domain (*dichotomy*.

space, we take the vector of distances of every feature between writings by the same writer and categorize it as a *within author distance* denoted by $x_{\oplus}$. The sample of *between author distance* is, on the other hand, obtained by measuring the distance between two different person's handwritings and is denoted by $x_\ominus$. Let d_{ij} denote i'th writer's j'th document.

$$\vec{x_\oplus} = \vec{\delta}(d_{ij} - d_{ik}) \text{ where } i = 1 \text{ to } n, j, k = 1 \text{ to } m \text{ and } j \neq k \tag{2.1}$$

$$\vec{x_\ominus} = \vec{\delta}(d_{ij} - d_{kl}) \text{ where } i, k = 1 \text{ to } n, i \neq k \text{ and } j, l = 1 \text{ to } m \tag{2.2}$$

where n is the number of writers, m is the number of handwritten documents per person, δ is the distance measure between two document feature values. Figure 2.1 (b) represents the transformed plot. The feature space domain is transformed to the feature distance space domain. A *within author distance*, W and a *between author distance*, B in feature domain in Figure 2.1 (a) correspond to the points W and B in the feature distance domain in Figure 2.1 (b), respectively. There are only two categories: *within author distance* and *between author distance* in feature distance domain.

Let $n_\oplus = |x_\oplus|$ and $n_\ominus = |x_\ominus|$, the sizes of within and between author distance classes, accordingly.

Fact 2.2.1 *If n people provide m writings, there are $n_\oplus = \binom{m}{2} \times n$ positive data, $n_\ominus = m \times m \times \frac{n(n-1)}{2}$ negative data and $\binom{mn}{2}$ data in total.*

Proof: $n_\oplus = \binom{m}{2} \times n$ is straight-forward. To count the negative data, we can enumerate them as $m \times (m \times (n-1)) + m \times (m \times (n-2)) + \cdots + m \times (m \times 1)$. For the first author, there are $m \times (n-1)$ number of other writer's writing data and he has three writing data. For the second author, there are $m \times (n-2)$ number of other writer's writing data that are not counted yet. Therefore, $n_\ominus = m \times m \times \sum_{i=1}^{n-1} i$. Now, $n_\oplus + n_\ominus$ must be $\binom{mn}{2}$.

$\binom{mn}{2} = \frac{(mn)!}{(mn-2)!2} = \frac{(mn)(mn-1)}{2} = \frac{m(m-1)}{2}n + m^2\frac{n(n-1)}{2} = n_\oplus + n_\ominus$ ∎

In our data collection, 1000 people (statistically representative U.S. population) provide exactly three samples. Hence, there are $n_\oplus = 3000, n_\ominus = 4,495,500$ and $4,498,500$ data in total.

Most statistical testing requires the assumption that observed data be statistically independent. The distance data is not statistically independent: one obvious reason being the triangle inequality of three distance data of the same person. This caveat should not be ignored. One immediate solution is to choose randomly a smaller sample from a large sample obviating the triangle inequality. One can partition $n_{\oplus} = 3000$ data into disjoint subsets of 500 guaranteeing no triangle inequality.

In the *dichotomy* model, we state the problem as follows; given two randomly selected handwritten documents, the *writer verification* problem is to determine whether the two documents were written by the same person with two types of confusion error probabilities. Figure 2.2 depicts the whole process of the *writer verification* using the *dichotomy transformation*. Let f_i^j be the i'th feature of j'th document. First, features

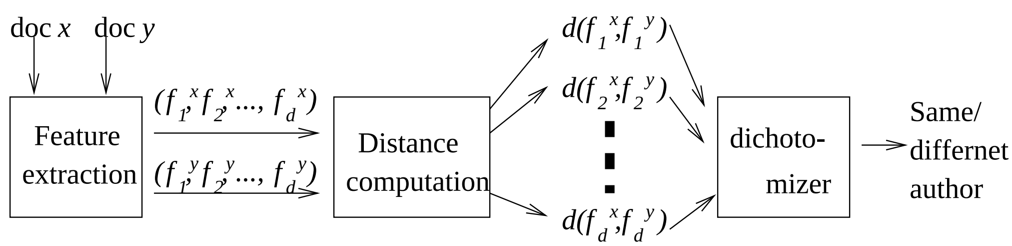


Figure 2.2: Writer Verification Process and dichotomy transformation

are extracted from both document x and y: $\{f_1^x, f_2^x, \cdots, f_d^x\}$ and $\{f_1^y, f_2^y, \cdots, f_d^y\}$. And then, each feature distance is computed: $\{\delta(f_1^x, f_1^y),\ \delta(f_2^x, f_2^y),\ \cdots,\ \delta(f_d^x, f_d^y)\}$. The *dichotomizer* takes this feature distance vector as an input and outputs the authorship.

A good descriptive way to represent the relationship between two populations (classes) is calculating overlaps between two distributions. Figure 2.3 illustrates the two distributions assuming that they are normal. Although this assumption is invalid, we use it to describe the behavior of two population figuratively without loss of generality. The *type I error*, α occurs when the same author's documents are identified as different authors and the *type II error*, β occurs when the two document written by two different writers are identified as the same writer as shown in Figure 2.3.

$$\alpha = Pr(\text{dichotomizer}(d_{ij}, d_{kl}) \geq T | i = k) \quad (2.3)$$

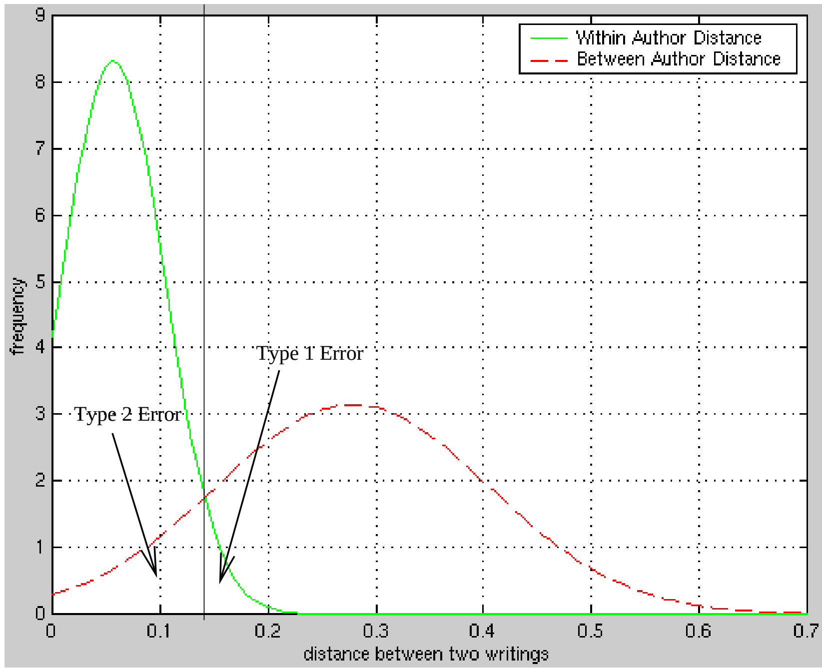
$$\beta = Pr(\text{dichotomizer}(d_{ij}, d_{kl}) < T | i \neq k) \quad (2.4)$$

Let $\hat{X}$ denote the distance x position where two distributions intersect. As shown in Figure 2.3, *type 1 error* is the right side area of positive distributions where the decision bound $T = \hat{X}$. Suppose one must make a crisp decision and choose the intersection as a classification bound. Then the type one error means that the probability of error that one classifies two writings as different authors even though they are written by a same person. The *type 2 error* is the left side area of negative distributions meaning the probability of error that one classifies two writings as a same author even though they are written by two different writers.

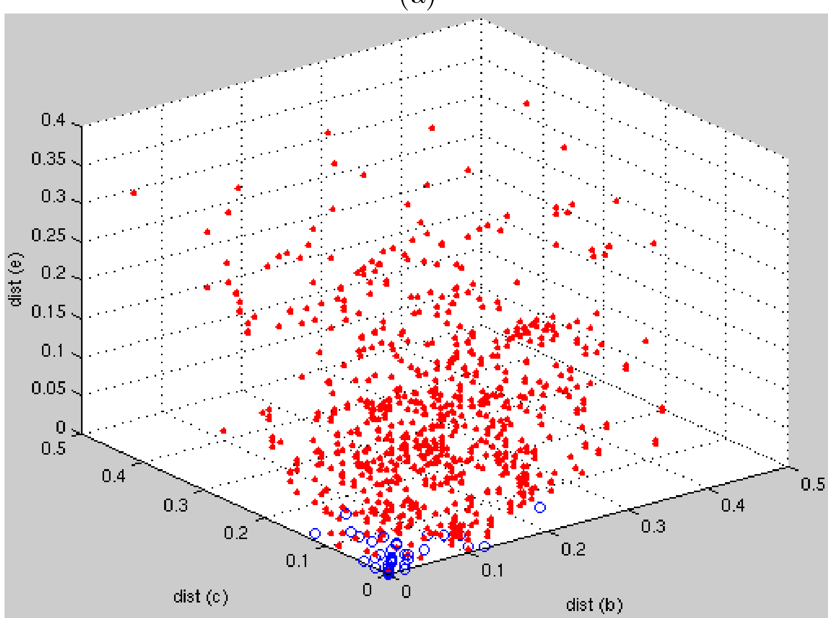
As is apparent from Figure 2.3, the within author distance distribution is clustered toward to the origin whereas the between author distance distribution is scattered away from the origin. Utilizing the fact that the within author distance is smaller, we design the *dichotomizer* that determines the decision boundary between within and between author distances.

2.3 Comparison: Polychotomy vs. Dichotomy

In this section, we discuss the advantage and disadvantage of the proposed *dichotomy* model and compares it with the *polychotomy* model. One immediate and obvious drawback of the *dichotomy* model is that it may fail to detect the difference between w_1 and w_2 who do not differ greatly even though they are detectably different geometrically in the feature space as shown in Figure 2.4. It would be desirable if all distances between the same class (writer) in feature domain belong to the within class distance class in feature distance domain. Similarly, we would like all distances between two different classes in feature domain belong to the between class distance



(a)



(b)

Figure 2.3: (a) Type I and II errors (b) 3-D Space Distribution.

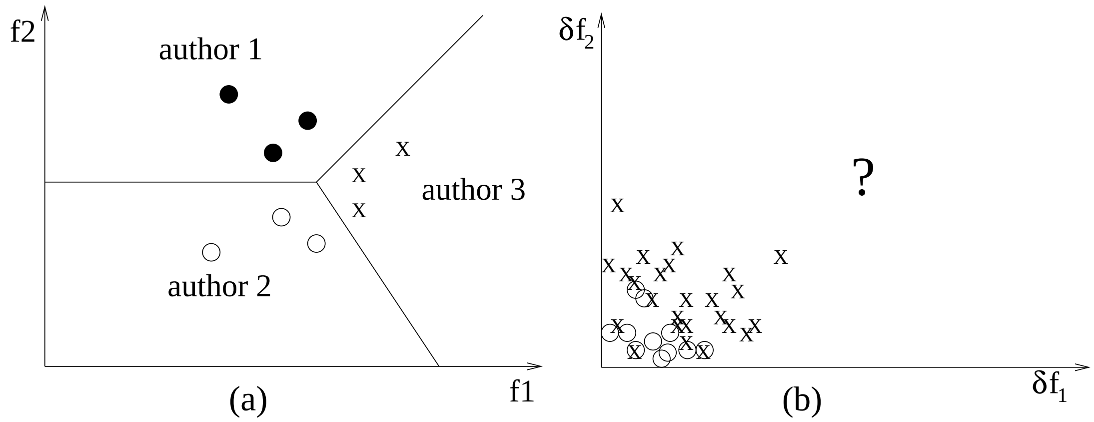


Figure 2.4: Comparison between (a) Feature domain (*polychotomy*) and (b) Feature distance domain (*dichotomy*).

class in feature distance domain. Unfortunately, this is not always the case; perfectly clustered class in feature domain may not be perfectly dichotomized in feature distance domain. The comparison in the *dichotomy* model is relative to a population and is crucially affected by the choice and diversity of the population. The broader the spread of the feature distributions among members of the population, the less we learn about detecting real differences between individuals who do not differ greatly. However, our experimental results show that these extreme cases are very rare.

Moreover, the objective of the paper is to validate the individuality of handwriting statistically but not to detect the difference of particular instances. We are attempting to infer the individuality of entire US population based on the individuality of the sample of 1000 writers. The *dichotomy* model is a sound and valid *inferential statistics*. The definition of *inferential statistics* is to measure the reliability of individuality about the entire population based on information obtained from a sample drawn from the population. We explain the justification of the *dichotomy* model using the *inferential statistics*.

Suppose that we use the *polychotomy* model to validate the individuality of handwriting. In this model, a population is all writings of a particular writer and thus there are US population number of populations. To draw the conclusion, one must draw samples from every writer, which is impossible. In our experimental design, we have only $1,000$ writers (classes/populations). Drawing statistical inferential conclusion is invalid because there are unseen populations. For example, consider a multiple classification problem of English alphabets and we are about to validate the individuality of alphabets. If one observes some instances of alphabets $\{A, B, C\}$ only, then drawing the conclusion that all alphabets are individual is invalid because there are indistinguishable handwritten italic I and l. Without knowing geometrical distribution of unseen classes (populations, writers), one cannot draw the statistical inference; true error of the entire population cannot be inferred from the error estimate of the sample population of 1000 because there are unseen classes (the rest of the US population).

Figure 2.5 (a)-(c) illustrates this issue. Suppose there are only 6 writers in the universe (a) and we observe some writings of the writer 1,4 and 5 only (b) because we assume that observing all writers is very hard. One can successfully learn the real and all differences among writers. However, the learned polychotomizer is not suitable to the rest of the classes as shown in (c).

Transforming the US population-class classification problem to a two class problem of authorship and non-authorship (dichotomizer model) helps us overcome this issue as shown in Figure 2.5 (d)-(f). (d) and (e) are the dichotomy transformed plots of (b) and (c), respectively. There are only two populations. We can acquire enough instances of each class or population. Since every new instance would also map onto these 2 classes, the distribution of the sample population can be used to infer the distribution of the entire population. Although we could do better on detecting real differences between individuals who do not differ greatly in the *polychotomy* model, the statistical inference is of primary interest and the *dichotomy* model is a sound and valid inferential statistics whereas the *polychotomy* is not. As we shall see in the later section 2.5, as borne out by our training and testing results, only 3% of the

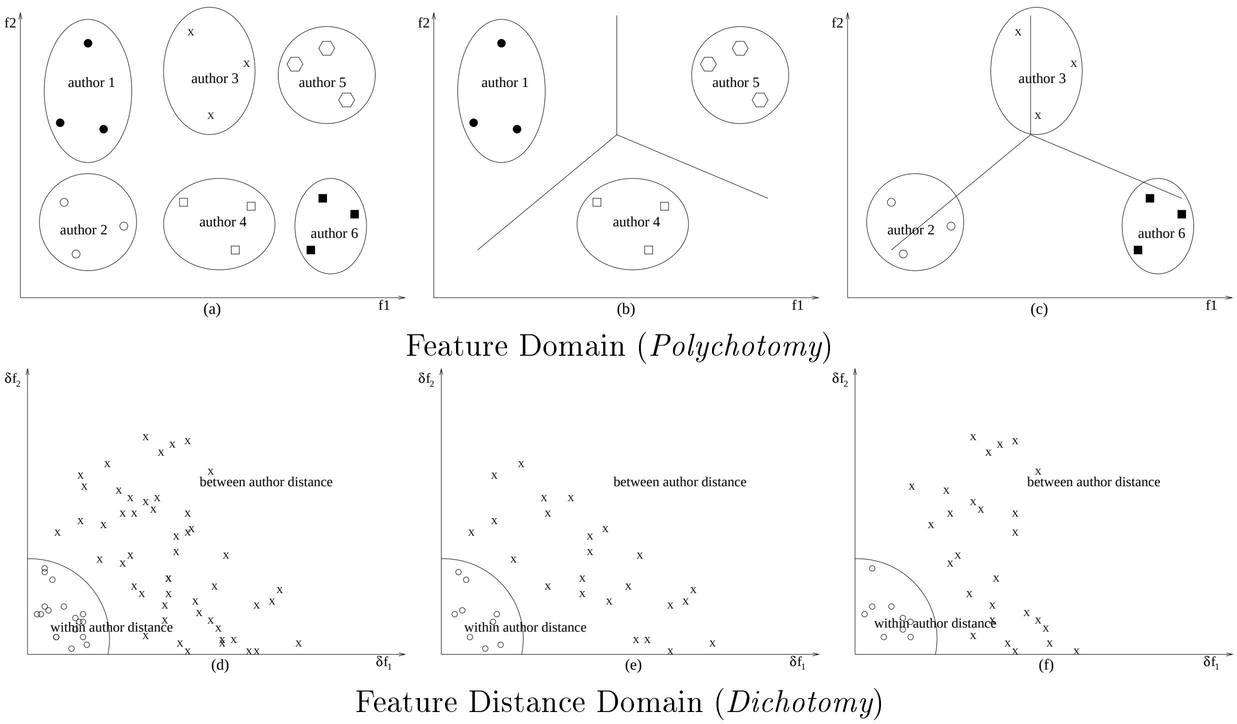


Figure 2.5: Statistical Inference in *Polychotomy* and *Dichotomy* (a) Entire classes in feature domain (b) partial classes and a classifier in feature domain (c) the rest of classes (d) Entire population in feature distance domain (e) a sample representative to the population in feature distance domain (f) another sample representative to the population in feature distance domain.

data was mis-classified. Since mis-classification error can be attributed to a number of factors such as feature selection, "masking", etc, "masking" might have occurred in an even smaller percentage. In all, inferring the error probability of the entire population through the dichotomizer model is more useful than, detecting real differences between individuals who not differ greatly for the sample population through the polychotomizer model, and not being able to infer those results for the entire population.

We have a trade-off between tractability and accuracy. Since sampling a sufficiently large sample from each individual person is intractable, we may wish to transform feature domain to the feature distance domain where we can get large samples for both classes. By the transformation, the problem becomes a tractable inferential statistic problem but we might get the lesser accuracy. If the number of classes is quite small that we can draw every sample for each class, then one may use the polychotomizer to validate the individuality of classes. However, one cannot preclude the *dichotomizer* even in the small multiple class classification problem. The *polychotomizer* may be better if the features are in a vector form of homogeneous scalar values. Techniques in *pattern recognition* typically require that features be homogeneous. The solution proposed overcomes the non-homogeneity of features as feature distances are nothing but scalar values. Hence, the proposed *dichotomy* model servers a two-bird-one-stone solution.

2.4 Experimental Database

In this section, we introduce the *CEDAR letter*, discusses its completeness and then describe the specification of the *CEDAR letter image database* consisting of writer data and features obtained from a handwriting sample, statistically representative of the US population.

2.4.1 CEDAR Letter

From *Nov 10, 1999*
Jim Elder
829 Loop Street, Apt 300
Allentown, New York 14707

To
Dr. Bob Grant
602 Queensberry Parkway
Omar, West Virginia 25638

We were referred to you by Xena Cohen at the University Medical Center. This is regarding my friend, Kate Zack.

It all started around six months ago while attending the "Rubeq" Jazz Concert. Organizing such an event is no picnic, and as President of the Alumni Association, a co-sponsor of the event, Kate was overworked. But she enjoyed her job, and did what was required of her with great zeal and enthusiasm.

However, the extra hours affected her health; halfway through the show she passed out. We rushed her to the hospital, and several questions, x-rays and blood tests later, were told it was just exhaustion.

Kate's been in very bad health since. Could you kindly take a look at the results and give us your opinion?
Thank you!

Jim

Figure 2.6: CEDAR Letter.

The *CEDAR Letter*, as shown in Figure 2.6, is concise (it has just 156 words), easy to understand and *complete*. It's *complete* in that, each alphabet occurs in the beginning of a word as a capital and a small letter, and as a small letter in the middle and end of a word. In addition, it also contains punctuation, numerals, interesting letter and numeral combinations (ff, tt, oo, 00) and a general document structure that would allow us to extract document level features such as word and line spacing, line skew etc. Forensic literature refers to many such documents - the *"London Letter"* [75], the *"Dear Sam Letter"* to name a few. But none of them are *complete* in the sense of the CEDAR Letter as follows. All capitals must appear in the letter and it is desirable to have all small letters in the beginning, middle and terminal positions of the word. We score the letter according to these constraints:

$$\text{score(letter x)} = \frac{104 - \text{Number of 0's}}{104} \tag{2.5}$$

The CEDAR letter scores 99% whereas the London letter scores 76%. the cedar letter has only 1 zero entry that is a word that ends with a letter *"j"*. Since there is no common English word that ends with the letter *"j"*, the cedar letter excludes this entry.

The table in Figure 2.7 shows counts of each letter in each position. The second row is the appearance count of the capital letter A through Z in the beginning of the word. The fourth to sixth rows are the appearance counts of the small letter a through z in the beginning, middle and terminal positions, respectively. Figure 2.7 shows the comparison between the CEDAR and London letters. Each table has 104 entries and we would like each entries to be non-zero. The London letter has 25 zero entries whereas the cedar letter has only 1 zero entry that is a word that ends with a letter *"j"*. The completeness of the *CEDAR letter* allows the better document analysis and examination.

2.4.2 Specification of Database

Each subject provides three exemplars of the CEDAR Letter and they are digitally scanned and stored in the image database. Thus, the *CEDAR letter* database consists of two entities: writer data and document feature data. First, there are seven attributes of writers collected through the questionnaire data-sheet. They are:

	A	B	C	D	E	F	G	H	I	J	K	L	M
Init	1	2	2	3	3	1	1	1	1	1	1	4	2

	a	b	c	d	e	f	g	h	i	j	k	l	m
Init	13	4	0	0	1	2	4	2	1	1	0	1	0
Mid	7	1	2	5	26	1	1	7	13	0	0	10	3
Ter	2	0	2	15	13	0	1	0	0	0	1	2	0

	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Init	1	1	1	1	2	2	2	1	1	1	1	1	1

	n	o	p	q	r	s	t	u	v	w	x	y	z
Init	13	4	0	0	1	2	4	2	1	1	0	1	0
Mid	17	20	3	0	15	7	8	9	2	2	2	1	1
Ter	6	2	0	1	7	10	7	0	1	0	0	2	1

(a) London Letter alphabet frequency counts: score = 76%.

	A	B	C	D	E	F	G	H	I	J	K	L	M
Init	4	2	4	1	1	1	1	1	1	2	3	1	1

	a	b	c	d	e	f	g	h	i	j	k	l	m
Init	17	4	1	1	6	1	2	9	4	2	1	2	2
Mid	33	2	8	6	59	4	5	20	32	1	3	14	3
Ter	5	2	1	21	20	3	3	5	1	0	3	5	2

	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Init	1	2	2	1	1	1	2	1	1	3	1	1	1

	n	o	p	q	r	s	t	u	v	w	x	y	z
Init	1	6	2	1	5	8	14	1	1	8	1	3	1
Mid	35	36	4	1	30	19	25	18	7	5	2	2	2
Ter	7	5	1	1	12	15	17	2	1	2	1	8	1

(b) CEDAR Letter alphabet frequency counts: score = 99%.

Figure 2.7: The CEDAR and London Letter: A Comparison.

- **Gender:** Male/Female
- **Age:**
  - under 15 years
  - 15 through 24 years
  - 25 through 44 years
  - 45 through 64 years
  - 65 through 84 years
  - 85 years and older
- **Handedness:** Left/Right
- **Highest level of education:** High school graduate/ higher
- **Country of Primary Education:** If USA, which state
- **Ethnicity:**
  - Hispanic
  - White (not Hispanic)
  - Black (not Hispanic)
  - Asian and Pacific Islander (not Hispanic)
  - American Indian, Eskimo, Aleut
- **Country of Birth:** USA/Foreign

We built a database that is "representative" of the US Population. This has been achieved by basing our sample distribution on the US census data (1996 Projections) [73]. There are 510 female and 490 male population distributions and 36% of white ethnicity group, and so forth. The database contains handwriting samples of 1,000 distinct writers.

We asked each participant to copy out the *CEDAR letter* three times in his or her most natural handwriting. Thus the relationship between the writer entity and the document entity is one to many ($m = 3$) as shown in Figure 2.8. In this data collection, provided uniform writing materials are used: plain unruled sheets and a medium black *Bic* round stic pen.

2.4.3 Feature Extraction

Encouraged by the recent success in off-line handwriting recognition and handwritten address interpretation [92], we utilize the similar features for the individuality validation. Albeit there are numerous features in line, word, character and spacing features, we give some document level computational features only in this paper.

The darkness value is the threshold value that separates the character parts and background parts of the document image. A digital image is a rectangular array of picture elements called *pixels* and each pixel has a darkness value between 0 and 255. A histogram is built and it has two peaks. One is due to dark handwritten characters and the other is due to the bright background. The valley between two peaks is the grey level threshold. We use the darkness value, grey level threshold value as an indicator of pen pressure. We will refer it (a). Another document level feature is (b) the number of blobs that is the number of connected components in the document image. A blob is also known as an exterior contour. This feature is related to intra-word and inter-word connections. Those writers who connects characters or words have fewer number of blobs while those who do not connect have lots of blobs. A similar feature is (c) the number of holes that is the number of closed loops. A hole is often called an interior contour or a lake. This feature gives the tendency of making loops while writing. The (d) average stroke width feature is computed by measuring

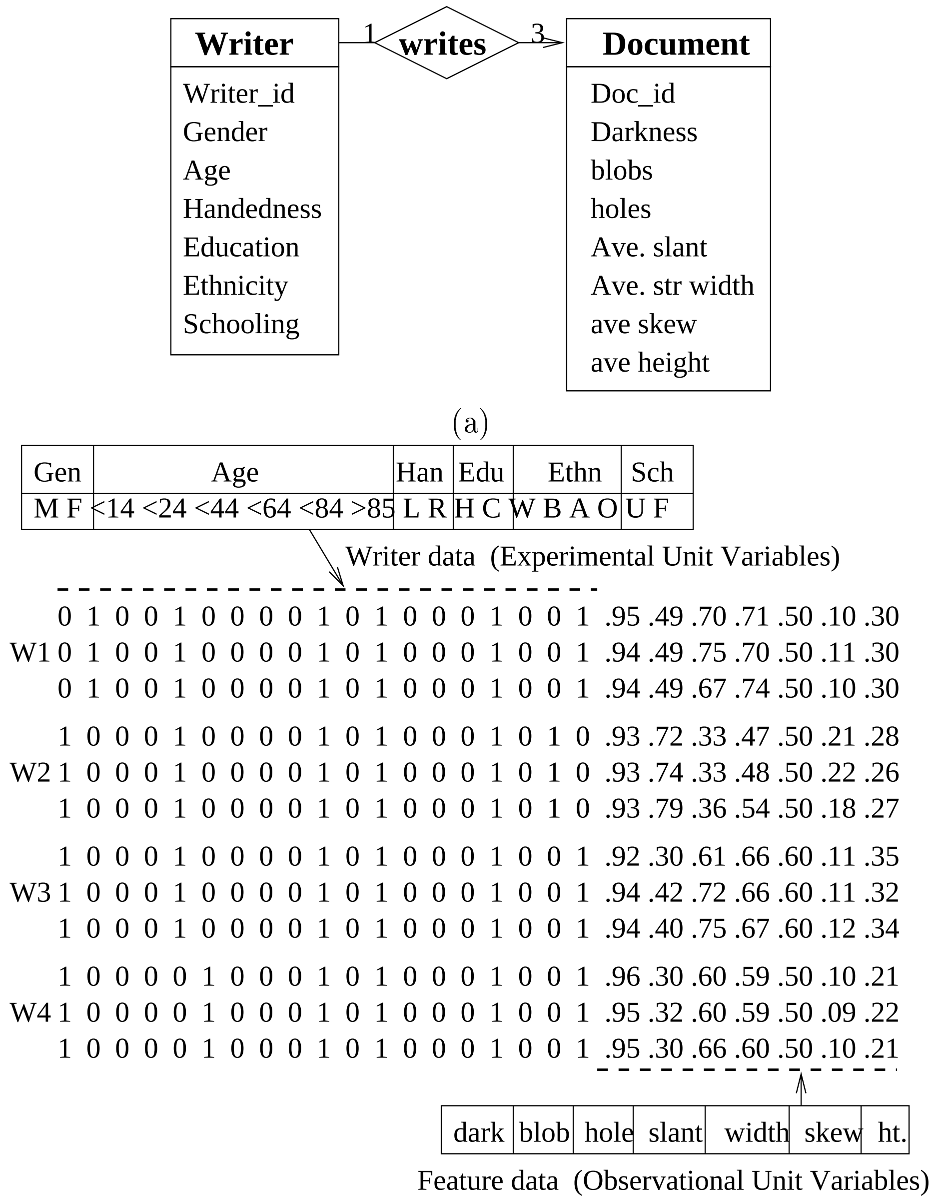


Figure 2.8: CEDAR letter Database (a) Entity and Relationship Diagram (b) Sample entries.

the highest frequency of width per line. We compute the (e) slant, (f) skew and (g) average height of character features.

As discussed earlier, types of features can be various: nominal, linear, angular, strings, histograms, etc. Full discussion and pointers to literature on various features, their distance measures and the multiple feature integration for writer identification can be found in [21].

2.5 Analysis

In this section, the full statistical analysis of the collected database. First, we use a parametric dichotomizer to gain some intuition on the effectiveness of the features. Next, we give the experimental results using the *Artificial Neural Network* as a *dichotomizer*.

2.5.1 Parametric Dichotomizer

First, we are concerned with *univariate parametric dichotomizer* to gain some intuition on the effectiveness of the particular feature. Means and standard deviations of the feature distances of two populations are estimated using the samples. We assume that the distribution is normal while it may be more related to non-central χ^2 distribution. Figure 2.9 shows all distributions for each feature. Table 2.1 shows

Table 2.1: Evaluating Features by overlaps

	δ_A	δ_B	δ_C	δ_D	δ_E	δ_F	δ_{ABCE}
$\hat{X}$	0.0172	0.1029	0.0825	0.0317	0.0576	1.7300	0.1407
Type 1 error	9.0%	6.94%	5.0%	24.54%	0.81%	3.0%	3.84%
Type 2 error	38.6%	27.3%	26.0%	51.4%	15.7%	27.0%	14.0%
Rem.	Good	Good	Good	Bad	Best	Good	

the intersection positions, $\hat{X}$'s and proportion of each error for each feature. Since we randomly select an equal number of points in the two classes, the two classes are

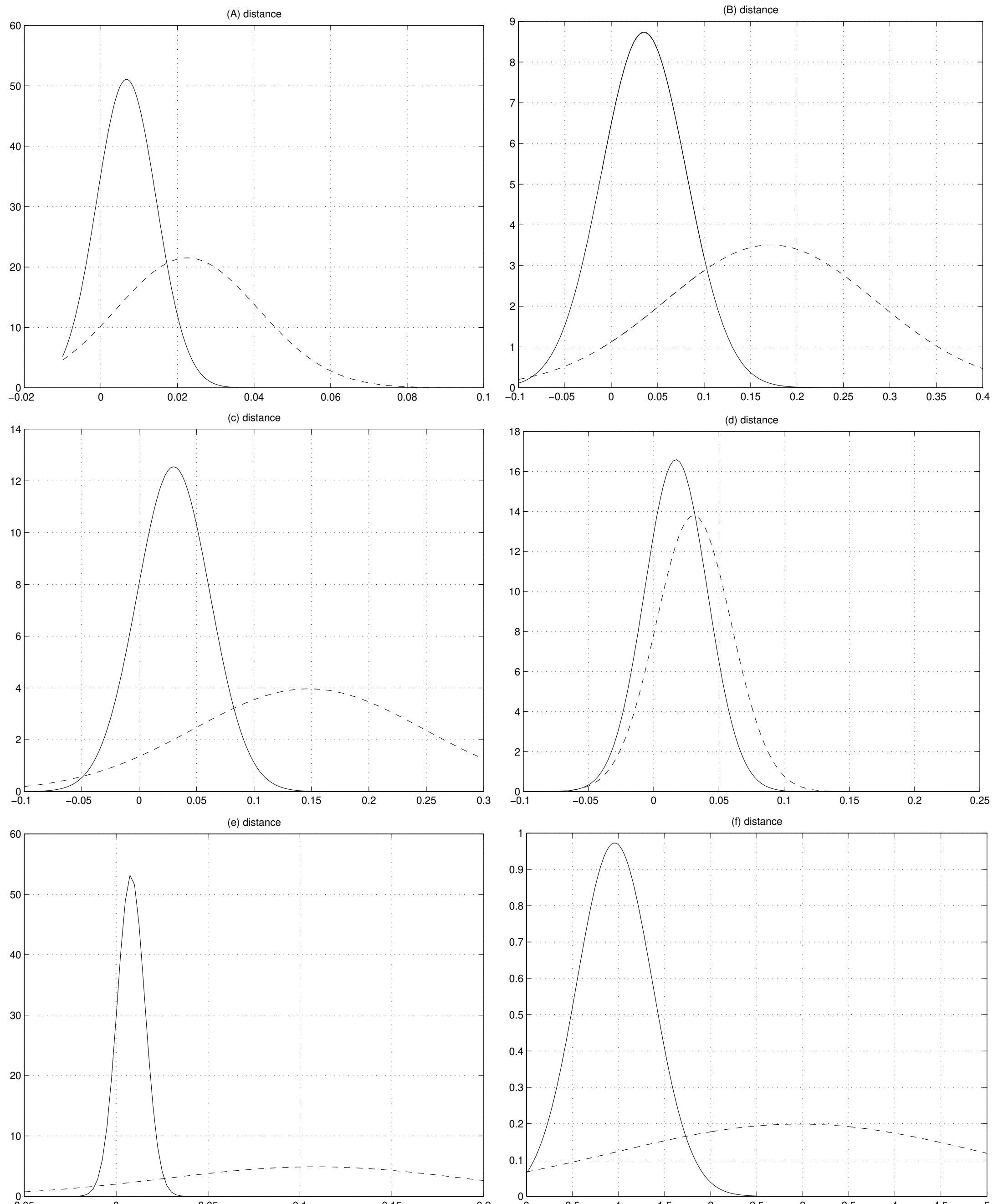


Figure 2.9: Positive and Negative Sample Distributions for each feature.

equally likely, and thus, the point of intersection is in fact the *Bayes discrimination point* that minimizes the sum of the *Type I* and *Type II* errors [36]. Note that feature (E) is an excellent feature whereas feature (D), the average stroke width is a bad one. Note that the last column δ_{ABCE} in Table 2.1 is not the full multivariate results but the univariate overlaps of the Euclidean distance of multiple features.

Another novel way to handle multiple features is to get the distance value for each feature and produce the multi-dimensional vector distances. Figure 2.3 (b) illustrates three dimensional distance values, $\{\delta_b, \delta_c, \delta_e\}$. Similar to the one-dimensional case, the within author distances tend to cluster toward the origin while the between authors distances tend to be apart from the origin. Various multivariate analysis [36, 69] may be applicable yet we use the *artificial neural network* as it requires no assumption on distribution.

2.5.2 Dichotomizer: Artificial Neural Network

We use the *Artificial Neural Network (ANN)* because it is equivalent to multivariate statistical analysis. There is a wealth of literature regarding a close relationship between neural networks and the techniques of statistical analysis, especially multivariate statistical analysis, which involves many variables [29, 36]. Samples of both class are divided into 6 groups of 500 in size. One pair set is used as a training set and the other set is used as a validation set. The rest of them are used as testing sets.

Using ten feature distance values, we trained an *artificial neural network* using the *back-propagation algorithm.* As discussed earlier, we generate all d features from the documents to be compared and then take the distance values between two documents x and y for each feature
$\{d(f_1^x, f_1^y), d(f_2^x, f_2^y), \cdots, d(f_d^x, f_d^y)\}$. These distance values are fed into the *Artificial Neural Network.* Figure 2.2 provides an overview of the entire process using the

artificial neural network as a *dichotomizer*. For training, the output value of 1 is given if the two documents $d_{i,j}$ and $d_{k,l}$ were written by a same writer and 0, otherwise. When the output value of the *ANN* is above 0.5, we classify it as the *"identity"* that two documents were written by the same writer and otherwise, they were written by two different writers. The distance vectors are divided into training, validating and testing sets.

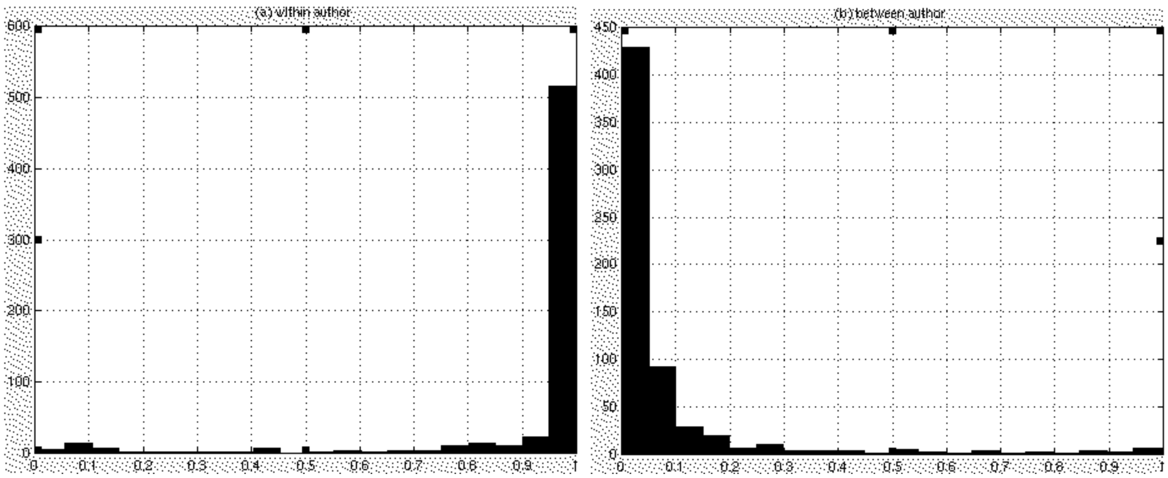


Figure 2.10: Decision Histogram on the testing set: (a) Within author distribution (Identity) (b) Between author distribution (Non-Identity).

Figure 2.10 shows the decision histogram on the testing set. The *within author distance* data, also know as *identity* tends to be voluminous toward the value 1 while the *between author distance* data or *non-identity* tends to be voluminous toward the value 0 as plotted in Figure 2.10 (a) and (b), respectively.

Table 2.2 shows the results when different number of feature distances are used. It is observed that the higher number of features the better the dichotomizer is.

Table 2.2: Experimental results vs. the number of features.

no. of features	5	9	10
Type I error	5.3%	4.6%	3.5%
Type II error	5.2%	3.5%	2.1%
Accuracy	95%	96%	97%

We refer the *dichotomizer* to denote the whole process in Figure 2.2 that takes two document images as input and authorship as output. Instead of the crisp decision, the fuzzy decision can be adopted utilizing the probability information given in Figure 2.10. When new two documents are given, the fuzzy dichotomizer computes the probabilities as stated in equations 2.3 and 2.4 by substituting the threshold value T with its *ANN* output value.

Furthermore, the higher accuracy can be achieved by allowing the rejects. The threshold T in the equations 2.3 and 2.4 are replaced with $T_1 < T$ and $T_2 > T$, respectively.

$$\begin{aligned} \alpha &= Pr(\text{dichotomizer}(d^x, d^y) < T_1 | \text{author}(x) = \text{author}(y)) \\ \beta &= Pr(\text{dichotomizer}(d^x, d^y) \geq T_2 | \text{author}(x) \neq \text{author}(y)) \end{aligned}$$

Any output between T_1 and T_2 is rejected and classified as *"I don't know."* We achieved 99.0% accuracy with 1.45% *Type I* error and 0.39% *Type II* error by allowing rejects of 15.9%.

2.5.3 Estimating Error Probability

Estimating the error probability is one of the simplest problems in statistical inference [110, 37] and performance evaluation [68]. The error probability estimation for the *writer verification* problem is depicted in Figure 2.11. A large number of writers ($w = 1,000$) are chosen as subjects for the experiment and they provided 3 handwritten documents each. There are two populations of interest and they are the *same writer document pair (SWDP)* and *different writer document pair (DWDP)*. A sample of *SWDP* is obtained from pairing two documents written by a same writer. A sample of *DWDP* is obtained from pairing two documents written by two different writers. A *writer verification* system is designed using the training and validation sample sets. The error probability is measured using the testing sample sets. There are two errors

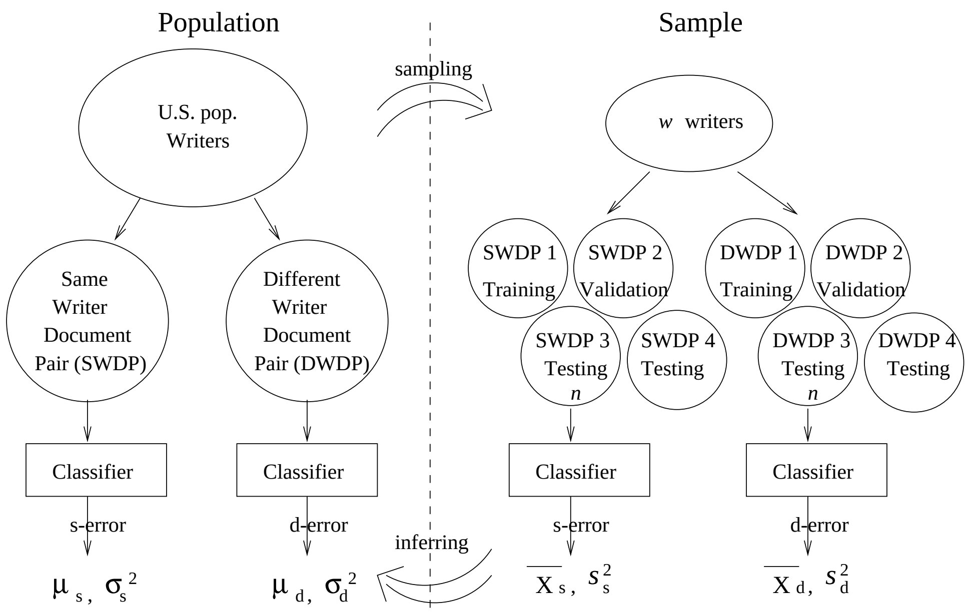


Figure 2.11: Error Evaluation Experimental Setup.

for each population. The *s-error* is the error probability that the *writer verification* system classifies two handwritten documents as a member of *SWDP* although they were written by two different writers. The*d-error* is, on the other hands, the error probability that the system classifies two handwritten documents as a member of *DWDP* even though they were written by one writer.

Estimating Error Mean and the Confidence Interval

Sample error means are denoted as $\bar{X}_s$ and $\bar{X}_d$ for the same writer document pair set and the different writer document pair set, respectively. They are often known as point estimates of population error means, μ_s and μ_d. We designed a system that is trained with training sets *SWDP 1* and *DWDP 1* with two validation sets *SWDP 2* and *DWDP 2*. The *writer verification* system is tested with the testing sets *SWDP 3* and *DWDP 3*. Each sample size is $n = 553$. The experimental results are $\bar{X}_s = 0.0669$ and $\bar{X}_d = 0.0325$.

In addition to the point estimates, we are interested in confidence intervals for

μ_s and μ_d. They are intervals within which we have reason to believe that the true population means, μ_s and μ_d amy lie assuming they are normal. The formula for the $1-\alpha$ level confidence interval for μ_s is:

$$\begin{aligned} \text{confidence interval for } \mu_s &\approx \bar{X}_s \pm t[1-\alpha/2; n-1]\sqrt{s_s^2/n} && (2.6) \\ &\approx \bar{X}_s \pm z[1-\alpha/2]\sqrt{\frac{\bar{X}_s(1-\bar{X}_s)}{n}} && (2.7) \end{aligned}$$

One can use either *Student's t* distribution or the normal table to compute the confidence interval because n is n is quite large. Although the population variance σ_s^2 is unknown [37], one can assume that $\sigma_s^2 \approx s_s^2$ for a large n. Thus the normal table is often used in evaluating performance [68]. We use both and choose one that gives the tighter bound for the sake of higher precision. Variances are $s_s^2 = 0.0624$ and $s_d^2 = 0.0315$. In both cases, $np(1-p) > 5$ and thus one can use either normal table or t-table. The difference is a tot. We have the confidence intervals: $0.046 < \mu_s < 0.088$ and $0.017 < \mu_d < 0.047$.

The normality assumption on the error distribution is a sine-qua-non in the analysis. The error probability follows a binomial distribution. A *Binomial distribution* gives the probability of observing r errors in a sample of n independent instances. A discrete binomial distribution function is then

$$P(r) = \frac{n!}{r!(n-r)!}p^r(1-p)^{n-r} \tag{2.8}$$

The expected, or mean value of X is

$$E[X] = np \tag{2.9}$$

The variance of X is

$$Var(X) = np(1-p) \tag{2.10}$$

For sufficiently large values of n the binomial distribution is closely approximated by a Normal distribution with the same mean and variance.

$$\text{s-error ND}(\mu_s, \frac{\sigma_s^2}{n}) \text{ and d-error ND}(\mu_d, \frac{\sigma_d^2}{n})$$

Most statisticians recommend using the Normal approximation only when $np(1-p) \geq 5$ [68].

From the experiments, we claim that $\mu_s = 0.0669$ and $\mu_d = 0.0325$. Given the results, we can perform the tests of hypotheses on the means.

$$H_0^1 : \mu_s = 0.0669, \qquad H_A^1 : \mu_s \neq 0.0669 \tag{2.11}$$

$$H_0^2 : \mu_d = 0.0325, \qquad H_A^2 : \mu_d \neq 0.0325 \tag{2.12}$$

We would like to validate the hypotheses using the other test sets. From *SWDP 4* and *DWDP 4*, we obtain new sample means, $\bar{X}_s = 0.0651$ and $\bar{X}_d = 0.038$. From the equation (2.7), we obtain the critical regions for the mean. The acceptance region for μ_s is $0.0445 < \mu_s < 0.0856$. Since the hypothesis H_0^1 states that $\mu_s = 0.0669$ and it is within the acceptance regions, we accept the null hypothesis. Similarly, the acceptance region for μ_d is $0.022 < \mu_d < 0.0539$ and we also accept the null hypothesis H_0^2.

Estimating Error Variance and the Confidence Interval

The point estimates for error variances can be derived from the variance in the binomial distribution case in equation 2.10 by substituting $r/n = \bar{X}_s$ for p.

$$\sigma_s^2 = \frac{Var(X)}{n} = \frac{np(1-p)}{n} = \bar{X}_s(1 - \bar{X}_s) \tag{2.13}$$

We have $s_s^2 = 0.0624$ and $s_d^2 = 0.0315$ from *SWDP 3* and *DWDP 3*.

In addition to the error means and their confidence intervals, we are also interested in accessing the error variances and their confidence intervals. The χ^2 distributions enable us to state confidence intervals for σ_s^2 and σ_d^2. Using the point estimate for the variance, we can calculate the confidence interval with $n-1$ degrees of freedom:

$$0 < \sigma_s^2 < \frac{(n-1)s_s^2}{\chi^2[\alpha; n-1]} \tag{2.14}$$

One of the reasons that we calculate the confidence intervals is that they are more informative than the point estimators. Moreover, they allow the hypothesis testings using another sample sets [37, 110]. The 95% confidence intervals are $0 < \sigma_s^2 < 0.0767$ and $0 < \sigma_d^2 < 0.0387$. In repeated sampling, the intervals cover σ_s^2 and σ_d^2 95% of the time.

We set up the hypotheses for variances.

$$H_0^3 : \sigma_s^2 = 0.0624, \qquad H_A^3 : \sigma_s^2 \neq 0.0624 \tag{2.15}$$

$$H_0^4 : \sigma_d^2 = 0.0315, \qquad H_A^4 : \sigma_d^2 \neq 0.0315 \tag{2.16}$$

From *SWDP 4* and *DWDP 4*, we obtain new sample variance point estimates ($s_s^2 = 0.0609$ and $s_d^2 = 0.0365$) and their confidence interval or acceptance regions are $0 < \sigma_s^2 < 0.0748$ and $0 < \sigma_d^2 < 0.0449$. Clearly, both values do not fall under their critical regions. Hence, we accept both null hypotheses H_0^3 and H_0^4.

2.5.4 Error Equality Test for Two Populations

In previous section, we considered the performance of the *writer verification* on the test data drawn from subjects that is representative of the US population. A question arises whether the performance of the system would be the same if the subjects are selected from a specific group of people. For example, subjects are chosen only from the white ethnicity, women and age group between 25 and 45.

We wish to study and compare a general population and a selected population. By studying two populations, one can determine whether the specific group of writers affects the performance of the system. In this section, we use the standard inference technique for two population proportions given in [110]. First, it uses the confidence interval for the difference between two population means and then for the difference between two population variances.

The experimental setup for these tests are illustrated in Figure 2.12. We designate

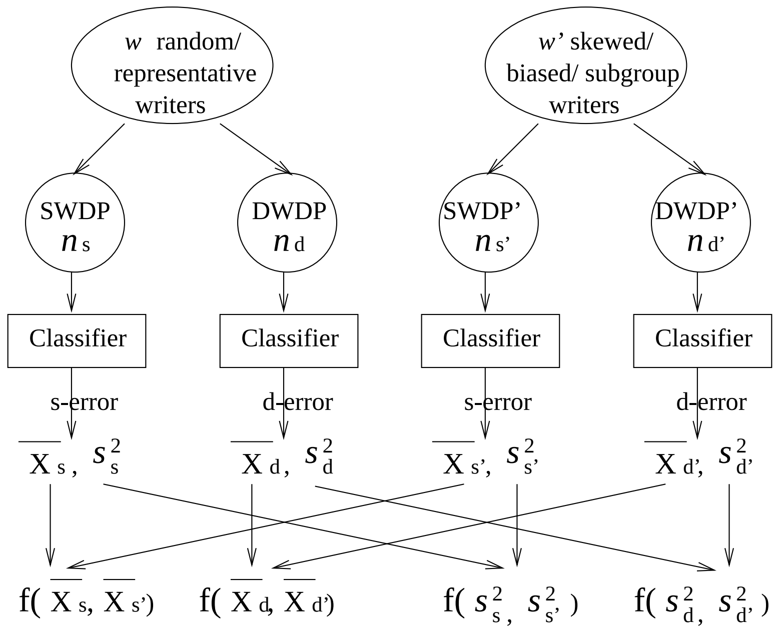


Figure 2.12: *Hypothesis Testing for two populations*

the two population *SWDP* means μ_s and $\mu_{s'}$ and the variance for the two populations $\sigma^2_{s^e}$.

The ultimate goal of this experiment is to determine whether there exists an effect on the system performance for a certain group of writers. We wish to state that there is no effect by accepting following four hypotheses.

$$H_0^1 : \mu_s = \mu_{s'}, \qquad H_A^1 : \mu_s \neq \mu_{s'} \tag{2.17}$$

$$H_0^2 : \sigma_s^2 = \sigma_{s'}^2, \qquad H_A^2 : \sigma_s^2 \neq \sigma_{s'}^2 \tag{2.18}$$

$$H_0^3 : \mu_d = \mu_{d'}, \qquad H_A^3 : \mu_d \neq \mu_{d'} \tag{2.19}$$

$$H_0^4 : \sigma_d^2 = \sigma_{d'}^2, \qquad H_A^4 : \sigma_d^2 \neq \sigma_{d'}^2 \tag{2.20}$$

We show the equality test for two population means using t distribution to validate the hypotheses H_0^1 and H_0^3 and then the equality test for two population variances using F distribution to validate the hypotheses H_0^2 and H_0^4.

Equality Testing for Two Population Means

Before embarking on testing, we must pool the estimates of the variances assuming that both populations have equal variances. Let $\bar{S}^2_{s^e}$ denote the pooled estimate of the variances $\bar{S}^2_s$ and $\bar{S}^2_{s'}$ and $\bar{S}^2_{de}$ denote the pooled estimate of the variances $\bar{S}^2_d$ and $\bar{S}^2_{d'}$ The pooled estimate is necessary because we wish not to combine two samples into one large set of data from which a single variance is calculated. The pooled estimate is instead used by the following equation (2.21).

$$S^2_{s^e} = \frac{(n_s - 1)S^2_s + (n_{s'} - 1)S^2_{s'}}{n_s + n_{s'} - 2} \tag{2.21}$$

The pooled estimate of the variance is simply a weighted mean of the sample variances with weights proportional to the degrees of freedom of the s^2_s and s^2_d. n_s and $n_{s'}$ are the sizes of *SWDP* and *SWDP'*. As the variance is computed using the binomial data, another way to compute the pooled estimate of variances is:

$$S^2_{s^e} = \left(\frac{r_s + r_{s'}}{n_s + n_{s'}}\right)\left(1 - \frac{r_s + r_{s'}}{n_s + n_{s'}}\right) \tag{2.22}$$

where $\bar{X}_s = r_s/n_s$ and $\bar{X}_{s'} = r_{s'}/n_{s'}$. r_s is the number of misclassified instances.

We now wish to make inferences concerning the means of the two populations; that is, for the population mean of the performance on the US representative subjects' writings and the population mean of the performance on a specific group of subjects' writings. In making inferences about the means, the null hypothesis is usually set to be equal and in our case, we wish to accept the null hypothesis to validate that there is no effect or reject the null hypothesis to establish that the performance of the *writer verification* is different on a specific group of people's handwriting. The two sample means, $\bar{X}_s$ and $\bar{X}_{s'}$, are assumed to be approximately normally distributed with means μ_s and $\mu_{s'}$, respectively, and variances $\sigma^2_{s^e}/n_s$ and $\sigma^2_{s^e}/n_{s'}$, respectively:

$$\bar{X}_i \qquad \mathrm{ND}(\mu_i, \frac{\sigma^2_{s^e}}{n_i}) \text{ for } i = s, s' \tag{2.23}$$

Now we can test whether a difference in means exists. By rewriting the hypothesis in equation (2.17), the null hypothesis is $H_0^1 : \mu_s - \mu_{s'} = 0$ and the alternative hypothesis $H_A^1 : \mu_s - \mu_{s'} \neq 0$. This is done by a *t distribution* with $n_s + n_{s'} - 2$ degrees of freedom:

$$t = \frac{(\bar{X}_s - \bar{X}_{s'}) - (\mu_s - \mu_{s'})}{\sqrt{S_{s^e}^2(\frac{1}{n_s} + \frac{1}{n_{s'}})}} \tag{2.24}$$

As stated earlier, we use *t distribution* instead of *z distribution* because the two population variances σ_s^2 and $\sigma_{s'}^2$ are unknown.

Since $t[.999, 901] = 3.107$, the computed $t < -3.107$ which is in the critical region. Therefore, we reject the null hypothesis and we conclude that the positive data mean is smaller than that of negative data. We can say that the given feature is a good feature to distinguish the positive and negative data.

Equality Testing for Two Population Variances

The previous equality testing for two population means is performed assuming that the two population variances are equal. Thus, before testing the mean equality or pooling the two sample variances s_s^2 and $s_{s'}^2$, we need to test the equality for two population variances. Otherwise, the normality assumption (2.23) is invalid. The null hypotheses and alternative hypotheses are given in equation (2.18) and (2.20), and they can be tested using the *F distribution*.

$$F = \frac{s_s^2}{s_{s'}^2} \tag{2.25}$$

When the level of significance is .05, the critical region for the hypothesis, $H_0^2 : \sigma_s = \sigma_{s'}$ are $0 < F < F[.025; n_s - 1, n_{s'} - 1]$ and $F > F[.975; n_s - 1, n_{s'} - 1]$. Since the computed value does not fall in the critical regions, we decide that the null hypothesis is correct.

The variance equality test may be redundant in the binomial data since the variance is computed based on the error probability.

2.5.5 Error Equality Test for Multiple Populations

The previous section considered the case where there are two populations. The number of populations are often greater than two. Consider the ethnicity parameter with values of {Black, Hispanic, and White} and we would like to study the performance differences among ethnicity groups. We tackle the problem using the previous equality testing for two population methods several times exhaustively. The Figure 2.13

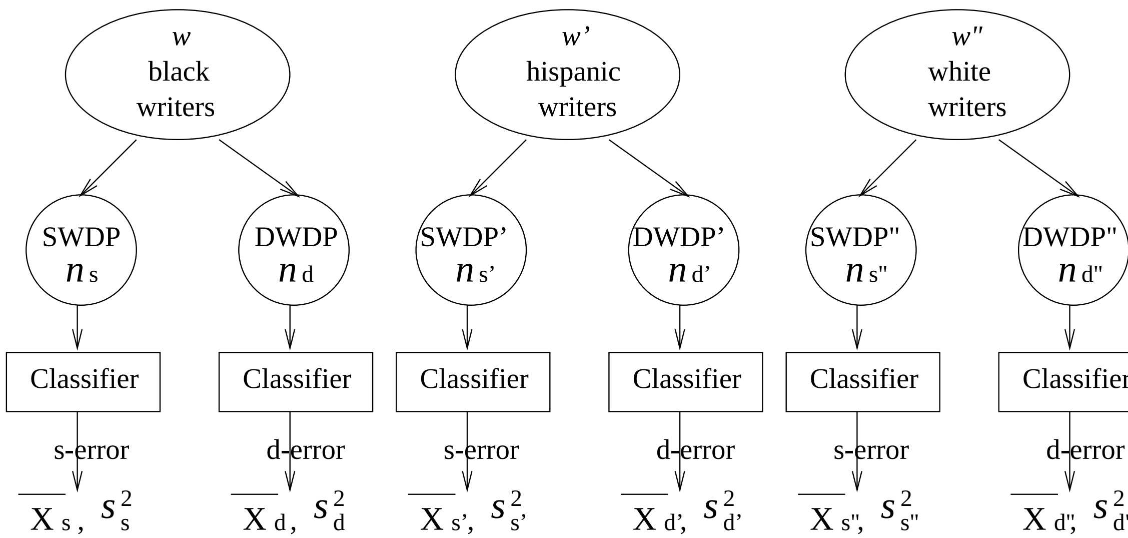


Figure 2.13: *Analysis of Variance for multiple populations*

shows the multiple category variate case. We are interested in testing the following hypotheses.

$$H_0^1 : \mu_s = \mu_{s'} = \mu_{s''}, \quad H_A^1 : \neg H_0^1 \tag{2.26}$$

$$H_0^2 : \sigma_s^2 = \sigma_{s'}^2 = \sigma_{s''}^2, \quad H_A^2 : \neg H_0^2 \tag{2.27}$$

$$H_0^3 : \mu_d = \mu_{d'} = \mu_{d''}, \quad H_A^3 : \neg H_0^3 \tag{2.28}$$

$$H_0^4 : \sigma_d^2 = \sigma_{d'}^2 = \sigma_{d''}^2, \quad H_A^4 : \neg H_0^4 \tag{2.29}$$

If we accept the hypotheses, we can conclude that the *writer verification* system is robust and performance is consistent regardless of whether the test data are from any of three ethnicity groups.

As mentioned earlier, there are three assumptions: i) independent random samples, ii) normal distributions, and iii) the variances of the three populations are equal. The samples are drawn independently and one can test the equality of variances introduced in the previous section. If the variances are equal, the data are normal as they follow the binomial distribution.

Suppose there are m populations to be studied. To accomplish the testings, first do the two population variance equality test for every pair of populations. There are $\binom{m}{2}$ pairs of populations. If the variances are equal, we can test the equality between means.

2.6 Procedure for Comparing Handwritten Items

In previous sections discussing the work on the validation of individuality of handwriting [25], we asked $1,000$ individuals to write the *"CEDAR letter"* three times to determine the statistical validity of individuality in handwriting based on measurement of features, quantification and statistical analysis. A measure of confidence is associated with individuality. Using ten feature distance values, we trained an *artificial neural network* and obtained 97% overall correctness. Various features and their distance measures are given in [21]. Unfortunately, the features used in this experiment do not always occur in the real world writer identification problem. Features are subject to change in case by case.

Consider a comparison between a handwritten item from a ransom note and suspects' exemplars of the same handwritten item. In an attempt to find the authorship confidence of the questioned documents, one can ask $1,000$ or a substantially large number of people that is representative to US population or criminals and do the same analysis as in [25]. Instead of *CEDAR letter database*, the given handwritten item database is constructed to support the authorship confidence. This experiment gives *type I* and *type II* errors helping the decision of the authorship. This is a very costly and time consuming process.

To this end, we propose a simulated method to access the authorship confidence of handwritten items. Instead of constructing the costly handwritten item database, simulated handwritten item database is built from the *CEDAR letter database*. We provide a procedure for determining whether two handwritten items were written by the same person. The suggested procedure is for digitally scanned handwritten items using visual information only.

2.6.1 Procedure

In this section, we first outline the procedure and then explain each step with the example of the real ransom note from *JonBenet Ramsey* murder case as shown in Figure 2.14. [2] To access the authorship confidence, we use the *dichotomy model* [25] as a framework. Only difference, the most important issue of this paper, is how the support image database is simulated.

Outline of Procedure

1. Questioned document item collection

   (a) Extract texts of interest (TOI) from QD.

   (b) Suspects write the *TOI* as well as *CEDAR letter*.

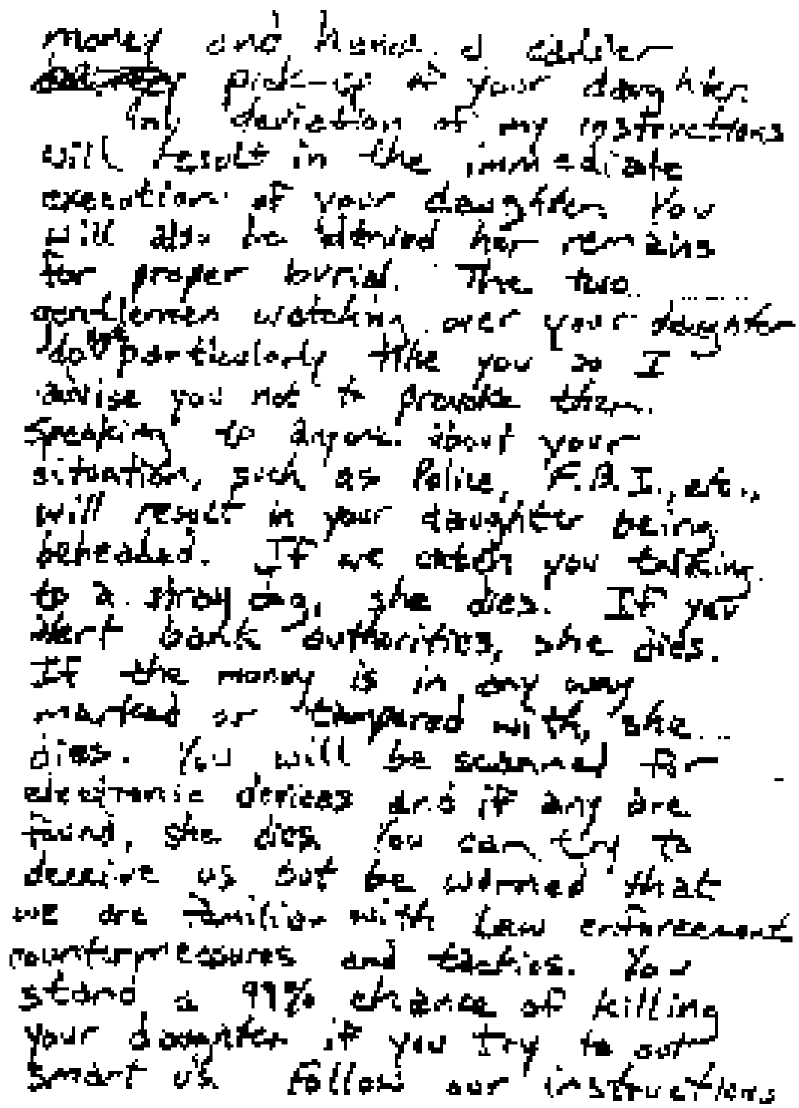
   (c) Scan the QD, suspects' *TOI* and *CEDAR letter*.

2. Simulated *TOI* image DB construction

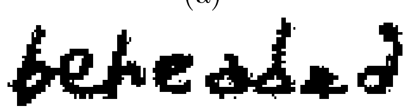
   (a) simulate words in QD using common substrings in CEDAR letter.

---

[2] a copy of the ransom note recovered in the JonBenet Ramsey murder case was released by District Attorney Alex Hunter. The image was obtained with permission from the URL "http://www.thesmokinggun.com/archive/ransom2.html".


money and hence a earlier
~~delivery~~ pick-up of your daughter.
Any deviation of my instructions
will result in the immediate
execution of your daughter. You
will also be denied her remains
for proper burial. The two
gentlemen watching over your daughter
do particularly like you so I
advise you not to provoke them.
Speaking to anyone about your
situation, such as Police, F.B.I., etc.,
will result in your daughter being
beheaded. If we catch you talking
to a stray dog, she dies. If you
alert bank authorities, she dies.
If the money is in any way
marked or tampered with, she
dies. You will be scanned for
electronic devices and if any are
found, she dies. You can try to
deceive us but be warned that
we are familiar with Law enforcement
countermeasures and tactics. You
stand a 99% chance of killing
your daughter if you try to out
smart us. Follow our instructions

(a)



(b)

Figure 2.14: (a) scanned QD, a ransom note (b) Extracted *TOI*, *“beheaded”*.

   (b) (Word spotting) find selected words in the scanned *CEDAR letter* images for every writer.

   (c) (TOI segmentation) segment the word out of the *CEDAR letter* images and then characters of interest out of the word images.

3. Finding the confidence of authorship

   (a) Generate the character level features.

   (b) Compute the feature distances.

   (c) Train an *ANN* using the simulated feature distances as input.

   (d) Build the *same* and *different writer* histograms from simulated data.

   (e) Find the *ANN* output value *(T)* for the distances between the *TOI* in QD and suspects' *TOI* and find error probabilities.

   (f) Validate the result.

Description of Procedure

The first step to compare QD with suspects' writing is to extract texts of interest, *TOI* in short, from the questioned document. Possible *TOI*'s are i) words that appear in the *CEDAR letter* ii) words that can be obtained from suspects' known writing exemplars, or an unusual word such as *"beheaded"* as shown in Figure 2.14 (b). All suspects are asked to copy the *TOI* as well as the *CEDAR letter*. The reason that suspects copy the *CEDAR letter* is for the purpose of the validation in the step 3 (f).

The next step is the simulated *TOI* image DB construction. synthesized *TOI* are generated from the *CEDAR letter* text. To do so requires the text retrieval using the *string matching* technique [98]. For example, to simulate the *TOI*, *"beheaded"*, we first find all words that have the longest *prefix-sub-string* in *CEDAR letter* text. There is only one word {*been*} that have the same first two characters *"be??????"*.

Next, we find all words that have the longest *suffix-sub-string* in the *CEDAR letter* text and they are {*referred, started, overworked, enjoyed, required, affected, passed, rushed*}. They all have the same last two characters *"?????ed"*. Amongst, we choose *"referred"*. For the remaining parts, we repeat searching the longest *sub-string* that are neither prefix nor suffix. We choose {*Co'he'n, Me'd'ic'a'l*}.

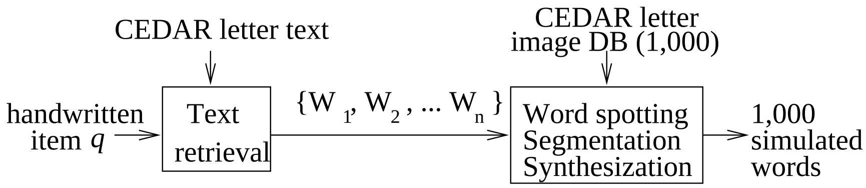


Figure 2.15: Simulated word *TOI* database construction

After words are selected to generate the *TOI*, *"beheaded"*, those words need to be spotted and segmented from the *CEDAR letter* document images. All characters to be synthesized to generate the *TOI* are also segmented from the words. Although there are automatic systems for spotting and segmentation, we use an image manipulating tool to manually extract characters. This is because the performances of current automatic systems are not yet as perfect as humans. This step is depicted in Figure 2.15 and sample synthesized *TOI*'s are shown in Figure 2.16.

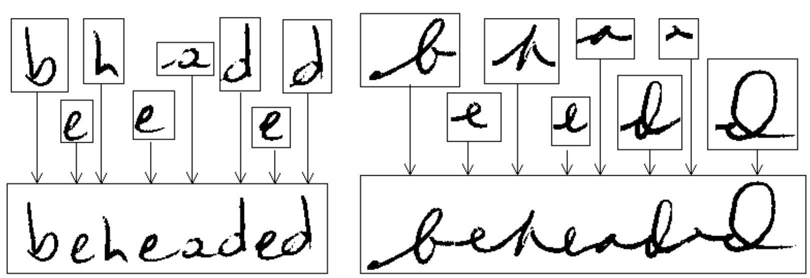


Figure 2.16: Synthesized *TOI*, beheaded

Finally, we find the authorship confidence of the *TOI*. Suppose the *TOI* from the handwritten item x has m character level features, then the *TOI* is represented in the vector of $(f_1^x, f_2^x, \cdots, f_m^x)$. We use only the character level features here as word level features cannot be synthesized from the *CEDAR letter* database. If the *TOI* happen to be the exact word that appear in the *CEDAR letter*, we use the word level features. Those words that frequently appear in both common *English* and *CEDAR letter* are {*From, To, We, were, to, you, at, the, This, is, It, all, an, no, and* }, etc.

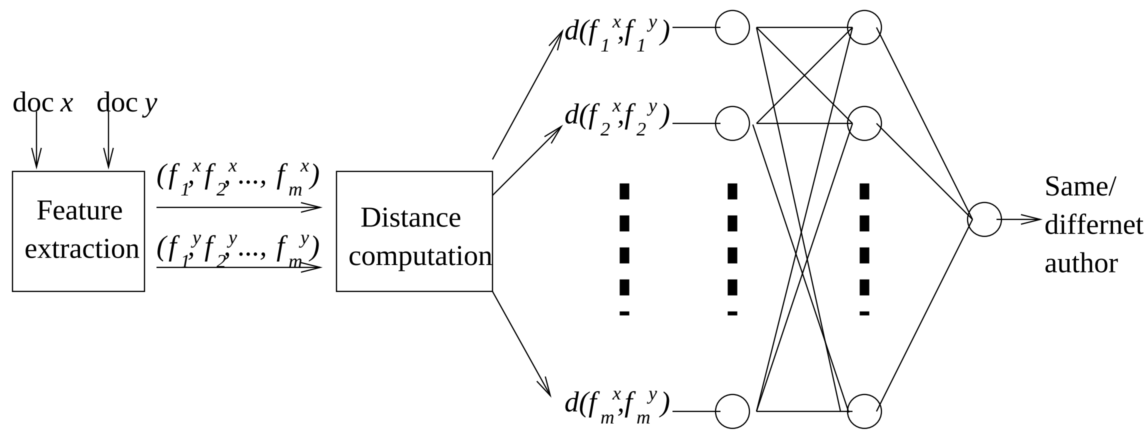


Figure 2.17: Artificial Neural Network

We generate all m features from the synthesized *TOI* and then take the distance values between two documents x and y for each feature $\{d(f_1^x, f_1^y),\ d(f_2^x, f_2^y),\ \cdots,\ d(f_m^x, f_m^y)\}$. These distance values are fed into the *Artificial Neural Network*. Figure 2.17 provides an overview of the entire process of the *artificial neural network*. For training, the output value of 1 is given if the two simulated handwritten items x and y were written by a same writer and 0, otherwise. We use the *Artificial Neural Network (ANN)* because it is equivalent to a sound multivariate statistical analysis [36]. When the output value of the *ANN* is above 0.5, we classify it as the *"identity"* that two documents were written by the same writer and otherwise, they were written by two different writers. The distance vectors are divided into training, validating and testing sets. Figure 2.10 shows the decision histogram on the testing set. The *within author distance* data, also know as *identity* tends to be voluminous toward the value 1 while the *between author distance* data or *non-identity* tends to be voluminous toward the value 0 as plotted in Figure 2.10 (a) and (b), respectively.

Now compute the distance vector between the *TOI* from the QD, q and that from the suspect, s: $\{d(f_1^q, f_1^s), d(f_2^q, f_2^s), \cdots, d(f_m^q, f_m^s)\}$. When these values are fed into the *ANN* as an input vector, the *ANN* returns an output value. We call the output value as the query bar, T as depicted in Figure 2.18. The query bar gives *Type I*, α

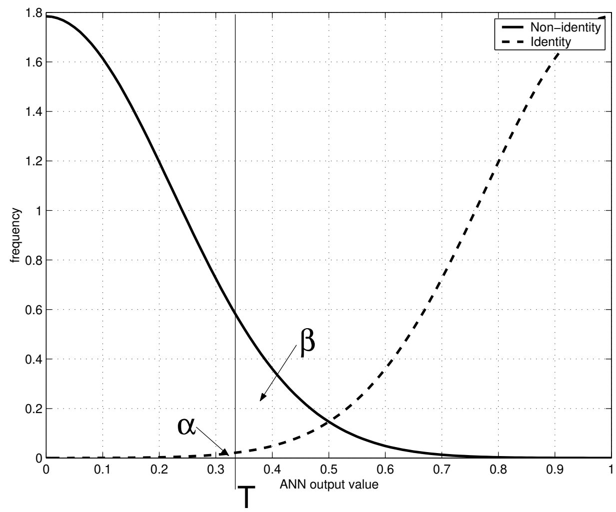


Figure 2.18: Simulated plot to illustrate T, α & β errors

and II, β errors.

$$\begin{aligned}\alpha &= Pr(\text{ANN}(d^x, d^y) < T | \text{author}(x) = \text{author}(y)) \\ \beta &= Pr(\text{ANN}(d^x, d^y) \geq T | \text{author}(x) \neq \text{author}(y))\end{aligned}$$

The *ANN* denotes the whole process in Figure 2.17 that takes two document images as input and authorship as output. The *type I error*, α is the probability of others having similar handwriting attributes as the writer. Conversely, the *type II error*, β is the probability of how differently one writer may write.

Finally, we determine whether the query bar and error probabilities are valid by taking the distance vector between the suspects' *TOI*'s and the synthesized *TOI* from suspects' CEDAR letter. If the similarity between them is low, the confidence is low as well and thus reject the result. If the output of ANN with this input values is identified as non-authorship, then reject the test, otherwise, accept the result. This is necessary to prevent the unnatural simulation of the *TOI*. Hence, this final step measures the acceptability of the synthesized *TOI*'s.

2.7 Conclusion

In this chapter, we showed that the multiple category classification problem can be viewed as a two-categories problem by defining the distance and taking those values as positive and negative data. This paradigm shift from the polychotomizer to the dichotomizer makes the writer identification that is a hard U.S. population multiple class problem very simple. We compared the proposed *dichotomy* model in feature distance domain with the *polychotomy* model in feature domain from the view point of tractability and accuracy. We designed an experiment to show the individuality of handwriting by collecting samples from people that is representative of the US population. Given two randomly selected handwritten documents, we can determine whether the two documents were written by the same person or not. Our performance is 97%.

One advantage of the dichotomy model working on distribution of distances is that many standard geometrical and statistical techniques can be used as the distance data is nothing but scalar values in feature distance domain whereas the feature data type varies in feature domain. Thus, it helps to overcome the non-homogeneity of features. Techniques in *pattern recognition* typically require that features be homogeneous. While it is hard to design a polychotomizer due to non-homogeneity of features, the dichotomizer simplifies the design by mapping the features to homogeneous scalar values in the distance domain.

The work reported in this paper is applicable to the area of *Forensic Document Examination.* We have shown a method to access the authorship confidence of handwritten items utilizing the *CEDAR letter* database. It is a procedure for determining whether or not two or more digitally scanned handwritten items were written by the same person. Thanks to the completeness of the *CEDAR letter* database, it is a panacea for the analysis for any handwritten item by synthesization.

Albeit the simulating *TOI* saves enormous time to build the database to support the authorship confidence, it still takes a while to spot and segment the word and character out of the document unless the questioned *TOI* is already segmented and stored in the image database. The automatic line, word and character segmentation problem is one of open problems.

Standardizing procedures require a community-based effort. In order for the proposed procedure to be recognized as a part of the standard forensic document examination protocol, it is necessary to provide the validation that it uniquely identifies the writer using further statistical experimental design and protocol evaluation techniques.

Chapter 3

On Measuring Distance between Histograms

[1] A histogram representation of a sample set of a population with respect to a measurement represents the frequency of quantized values of that measurement among the samples. Finding the distance, or similarity, between two histograms is an important issue in pattern classification and clustering [36, 87, 28]. A number of measures for computing the distance have been proposed and used.

There are two methodologies in histogram distance measures: vector and probabilistic. In the vector approach, a histogram is treated as a fixed-dimensional vector. Hence standard vector norms such as *city block, Euclidean* or *intersection* can be used as distance measures. Vector measures between histograms have been used in image indexing and retrieval [43, 78, 84, 101].

The probabilistic approach is based on the fact that a histogram of a measurement provides the basis for an empirical estimate of the probability density function (pdf) [35]. Computing the distance between two pdf's can be regarded as the same as computing the Bayes (or minimum misclassification) probability. This is equivalent to measuring the overlap between two pdf's as the distance. There is much literature regarding the distance between pdf's, an early one being the *Bhattacharyya* distance or *B-distance* measure between statistical populations [58]. The *B-distance*, which is a value between 0 and 1 provides bounds on the Bayes misclassification probability. An approach closely related to the *B-distance* was proposed by Matusita [67, 28].

[1] This chapter contains works published in [11, 12, 13, 19] and is under review in pattern recognition journal [23].

Kullback and Leibler [63] generalized Shannon's concept of probabilistic uncertainty or *"entropy"* [86] and introduced the *"K-L distance"* [36, 87] measure that is the minimum cross entropy (see [104] for an extensive bibliography on estimation of misclassification).

The viewpoint of regarding the overlap (or intersection) between two histograms as the distance has the disadvantage that it does not take into account the similarity of the non-overlapping parts of the two distributions. For this reason, we present a new definition of the distance for each type of histograms. The new measure uses the notion of the *Minimum Difference of Pair Assignments*. We also describe the algorithms to compute the distance for each type.

The subsequent sections are constructed as follows. In section 3.1, histograms are defined with respect to three measurement types. In section 3.2, we define a new definition of distance measure. In section 3.3, we examine conventional definitions of distance between two histograms and give examples that show the inadequacy of them when they are used for certain types of measurements. Section 3.4 is dedicated to the description and analysis of each algorithm to compute the distance between each type of histograms. In section 3.5, we address the character similarity problem with the proposed measure after extracting the *Gradient Direction* histograms. Finally, we conclude with the emphasis that the proposed distance measure is a suitable and culminating distance measure between two histograms and other conventional definitions are inadequate for ordinal or modulo type histograms.

3.1 Histogram Definition

We will use the following notations and symbols. Let x be a measurement, or feature, which can have one of b values contained in the set, $X = \{x_0, x_1, \cdots, x_{b-1}\}$. Consider a set of n samples whose measurements of the value of x are: $A = \{a_1, a_2, \cdots, a_n\}$ where $a_j \in X$. The histogram of the set A along measurement x is $H(x, A)$ which is an ordered list (or b-dimensional vector) consisting of the number of occurrences of the discrete values of x among the a_i. As we are interested only in comparing the histograms of the same measurement x, $H(A)$ will be used in place of $H(x, A)$ without loss of generality. If $H_i(A)$, $0 \leq i \leq b-1$, denotes the number of samples of A that have value x_i, then $H(A) = [H_0(A), H_1(A), \cdots, H_{b-1}(A)]$, where

$$H_i(A) = \sum_{j=1}^{n} c_{ij} \text{ where } c_{ij} = \begin{cases} 1, & \text{if } a_j = x_i \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

If $P_i(A)$ denotes the probability of samples in the i-th value or bin, then $P_i(A) = H_i(A)/n$.

As illustrated in Fig. 3.1, a histogram, $H(A)$ is shown for a set of samples for $n = 10$ and $b = 8$, with $A = \{1, 0, 7, 6, 0, 0, 2, 6, 6, 0\}$, $H(A) = [4, 1, 1, 0, 0, 0, 3, 1]$ and $P(A) = [.4, .1, .1, 0, 0, 0, .3, .1]$

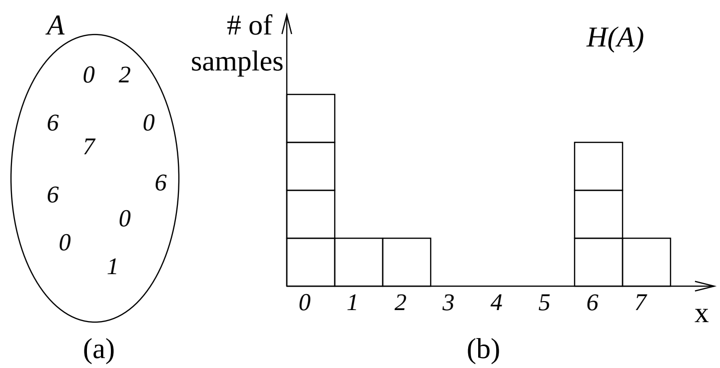


Figure 3.1: *(a) Measurements corresponding to a set of samples A and (b) its histogram H(A)*

If the ordering of the elements in sample set A is unimportant, then $H(A)$ is a lossless representation of A in that A can be reconstructed from $H(A)$.

3.1.1 Types of Measurements

We consider three types of measurements: nominal, ordinal and modulo. According to the measurements, we consider three types of histograms. In a nominal measurement, each value of the measurement is named, e.g., the make of an automobile can take values such as *GM, Ford, Toyota, Hyundai*, etc. An example of a nominal type histogram is one that consists of the numbers of each automobile make in a parking lot. In an ordinal measurement, the values are ordered, e.g., the weight of an automobile can be quantized into ten integer values between 0 and 9 tons. Most measurements are of the ordinal type, e.g., year, height, width or weight of automobiles or grey scale level in grey images. Finally, in modulo measurement, measurement values form a ring due to the arithmetic modulo operation, e.g., the compass direction of an automobile that can take eight values, N, NE, E, SE, S, SW, W, NW, form a ring under the operation of changing direction by 45°. The modulo type histograms are obtained along the angular values such as directions or *"hue"* in color images.

3.1.2 Permutability of Levels

The measurement values are called *levels* when they are used in a histogram to index the sample values, e.g., grey level. In a nominal type histogram, the levels can be of any order and permuted freely as there is no particular ordering among themselves. In contrast, there exists an ordering among the levels in both ordinal and modulo type histograms.

In finding the distance between two histograms of nominal type measurements, the ordering of levels should not affect the outcome as long as the two histograms maintain the same ordering in their levels. For instance, the distance between two histograms of automobile make should not change whether the ordering is *GM, Ford, Toyota, Hyundai* or it is *Ford, Hyundai, GM, Toyota*. This *shuffling invariance* property is satisfied by the existing methods of distance measure, such as *city block, Euclidean, intersection, Bhattacharyya, Matusita* and *K-L distances*, because they are sums of individual distances of each level and due to the commutative law, the distances do not change when levels are permuted among themselves.

On the contrary, the *shuffling invariance* property is not desirable in the distance between the histograms of ordinal or modulo type measurements. Levels cannot be permuted by definition of ordering in levels. Consider the following histograms of ordinal measurement type where $b = 8$, the range $= [0, 7]$ and $n = 5$

$$\begin{aligned} H(D) &= [0 \;\; 5 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0] \\ H(E) &= [0 \;\; 0 \;\; 5 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0] \\ H(F) &= [0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 5] \end{aligned}$$

The distance between $H(D)$ and $H(E)$ must be smaller than that between $H(D)$ and $H(F)$ if histograms are ordinal in measurement type whereas they are the same in nominal measurement type. We will present the universal definition of distance that satisfies both *shuffling invariance* and *shuffling dependence* properties for nominal and other type measurements, respectively.

3.1.3 Difference between quantized measurement levels

Given a set of samples together with values of measurements (or attributes) made on the samples, and where the measurement is quantized into a discrete set of values (levels), a histogram represents the frequency of each discrete measurement. Corresponding to three types of measurements: nominal, ordinal and modulo, we define three measures of difference between two measurement levels, $x, x' \in X$ as follows:

$$\text{nominal:} \quad d_{\text{nom}}(x, x') = \begin{cases} 0, & \text{if } x = x' \\ 1, & \text{otherwise.} \end{cases} \tag{3.2}$$

$$\text{ordinal:} \quad d_{\text{ord}}(x, x') = |x - x'|. \tag{3.3}$$

$$\text{modulo:} \quad d_{\text{mod}}(x, x') = \begin{cases} |x - x'|, & \text{if } |x - x'| \leq \frac{b}{2} \\ b - |x - x'|, & \text{otherwise.} \end{cases} \tag{3.4}$$

The distance value between two nominal measurement sample values is either match or mismatch as shown in Eq.(3.2) and thus levels are permutable. Levels are totally ordered and non-permutable in ordinal measurement sample values. The distance between two ordinal measurement values is the absolute difference between them as shown in Eq.(3.3). Finally, levels form a ring and non-permutable in modulo measurement sample values. The distance between them is the interior difference as shown in Eq.(3.4). For example, for an angular measurement between 0° to 360° , $d_{\text{mod}}(350°, 1°) = 11° \neq 349°$.

The three measures in Eqs.(3.2),(3.3) and (3.4) satisfy the following necessary properties of a metric:

$$d(x, x) = 0 \text{ reflexivity} \tag{3.5}$$

$$d(x, x') \geq 0 \text{ non-negativity} \tag{3.6}$$

$$d(x, x') = d(x', x) \text{ commutativity} \tag{3.7}$$

$$d(x, x'') \leq d(x, x') + d(x', x'') \text{ triangle ineq.} \tag{3.8}$$

Since they are straight-forward facts, we omit the proofs except for the triangle inequality of d_{mod}.

Fact 3.1.1 *d_{mod} has triangle inequality property: $d_{mod}(x, x'') \leq d_{mod}(x, x') + d_{mod}(x', x'')$.*

Proof: Let θ_1 be the interior angle between x and x'' and θ_2 and θ_3 be interior angles between x and x' and between x' and x'', respectively. There are four cases as shown in Fig. 3.2. Case (a) is such that x' lies between x and x''. Since $\theta_1 = \theta_2 + \theta_3$,

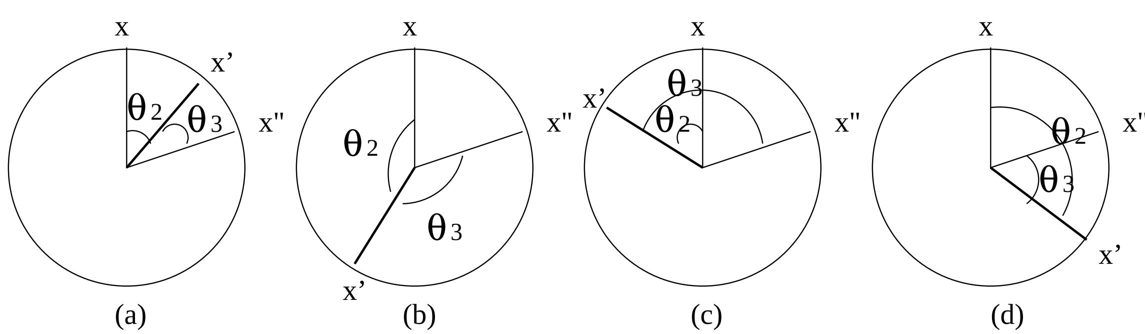


Figure 3.2: *4 cases of 3 modulo measurement values*

$d_{\text{mod}}(x, x'') = d_{\text{mod}}(x, x') + d_{\text{mod}}(x', x'')$. Case (b) is such that $\theta_2 + \theta_3$ is the exterior angle between x and x''. As an exterior angle is always greater than or equal to their interior angle, $\theta_1 \leq \theta_2 + \theta_3$. Both cases (c) and (d) are such that either θ_2 or θ_3 covers θ_1. There is no other case. Clearly, $\theta_1 \leq \theta_2 + \theta_3$. Therefore, $d_{\text{mod}}(x, x'') \leq d_{\text{mod}}(x, x') + d_{\text{mod}}(x', x'')$. ∎

3.2 A New Distance Measure

The distance between any two histograms can be expressed in terms of the distances of sample measurement values. Given two sets of n samples, A and B, we consider the problem as one of finding the minimum difference of pair assignments between two sets. The problem is to determine the best one-to-one assignment between two sets such that the sum of all differences between two individual samples in a pair is minimized. Given n samples $a_i \in A$ and n samples $b_j \in B$, we define the *Minimum difference of pair assignments* as:

$$\text{Minimum difference of pair assignments: } D(A, B) = \min_{A,B}\left(\sum_{i,j=0}^{n-1} d(a_i, b_j)\right) \quad (3.9)$$

where D and d are designated as D_{nom} and d_{nom}, D_{ord} and d_{ord}, and D_{mod} and d_{mod} for nominal, ordinal and modulo measurements, respectively. $\sum$ is a usual arithmetic summation in all three cases. The more similar the two histograms are, the smaller the value $D(A, B)$ is. We are interested only in the value $D(A, B)$ rather than the assignments.

As $H(A)$ is a lossless representation of A, we define the new distance measure between histograms, $D(H(A), H(B)) = D(A, B)$ given in Eq.(3.9). Also, we shall use $D(A, B)$ as a short form of the distance between two histograms, $D(H(A), H(B))$. First, we need to show that the proposed measure is indeed a metric so that it can be useful as a distance measure.

3.2.1 Metric Property

We show that the new distance measure, $D(A, B)$ satisfies conditions for being a metric: non-negativity, reflexivity, commutativity and triangle inequality. Since $d(a_i, b_j)$ is a metric, it follows that $D(A, B)$ is also a metric that can be used to compare two histograms.

Fact 3.2.1 *$D(A, B)$ has non-negativity property: $D(A, B) \geq 0$.*

Proof: $D(A, B)$ is nothing but the sum of $d(a_i, b_j)$ and each $d(a_i, b_j)$ has non-negativity property. Therefore, $D(A, B)$ also has the non-negativity by definition. ∎

Fact 3.2.2 *$D(A, A)$ has reflexivity property: $D(A, A) = 0$.*

Proof: Since $d(a_i, a_i) = 0$ is true, $\sum_{i=1}^{n} d(a_i, a_i) = 0$. Due to the non-negativity, 0 is the minimum bound of the measure: $D(A, A) \geq 0$. Therefore, $D(A, A) = \sum_{i=1}^{n} d(a_i, a_i) = 0$ by definition. ∎

Fact 3.2.3 *$D(A, B)$ has commutativity property: $D(A, B) = D(B, A)$.*

Proof: Let $\mathrm{comm}(\sum_{i=1}^{n} d(a_i, b_i)) = \sum_{i=1}^{n} d(b_i, a_i)$ and vice versa. Due to the commutativity in $d(a_i, b_i)$, $\sum_{i=1}^{n} d(a_i, b_i) = \mathrm{comm}(\sum_{i=1}^{n} d(a_i, b_i))$. Now suppose $D(A, B) \neq D(B, A)$ and $D(A, B) = \sum_{i=1}^{n} d(a_i, b_i)$. This means that $D(B, A) \neq \sum_{i=1}^{n} d(b_i, a_i)$. Let the assignment for $D(B, A)$ be α. Then $\alpha < \sum_{i=1}^{n} d(b_i, a_i)$. Since $\alpha = \mathrm{comm}(\alpha)$, $\mathrm{comm}(\alpha) < \sum_{i=1}^{n} d(a_i, b_i)$. It contradicts that $D(A, B)$ is the minimum difference of pair assignments. Therefore, $D(A, B) = D(B, A)$ by contradiction. ∎

Fact 3.2.4 *D satisfies the triangle inequality property: $D(A, C) \leq D(A, B) + D(B, C)$.*

Proof: Let the assignments $a_i \rightarrow b_i$ be the assignments of $D(A, B)$. Let $b_i \rightarrow c_i$ be the assignments of $D(B, C)$. Then a_i is assigned to c_i by $a_i \rightarrow b_i \rightarrow c_i$. However, $\sum_{i=1}^{n} d(a_i, c_i)$ is not necessarily the minimum and it may not be $D(A, C)$. Thus,

$D(A,C) \leq \sum_{i=1}^{n} d(a_i, c_i)$. Now from Eq.(3.8), the following equation can be drawn: $\sum_{i=1}^{n} d(a_i, c_i) \leq \sum_{i=1}^{n} d(a_i, b_i) + \sum_{i=1}^{n} d(b_i, c_i)$. $D(A,C) \leq \sum_{i=1}^{n} d(a_i, c_i) \leq D(A,B) + D(B,C)$. Therefore, $D(A,C) \leq D(A,B) + D(B,C)$. ∎

3.2.2 Univariate Case: Example

Consider three sets of sample measurements with $b = 8$ and $n = 10$ as follows:

$$\begin{aligned} A &= \{0,0,0,0,1,2,6,6,6,7\} \\ B &= \{0,1,1,1,1,2,6,6,6,7\} \\ C &= \{0,0,1,2,6,6,6,7,7,7\} \end{aligned}$$

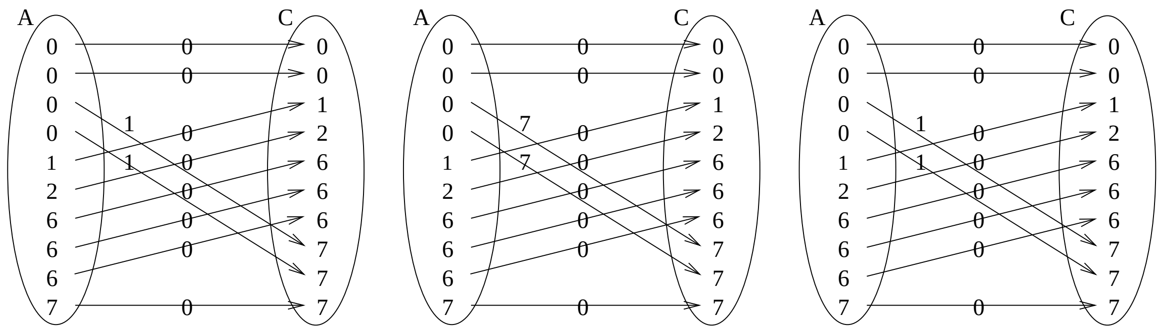
The corresponding three univariate histograms are

$$\begin{aligned} H(A) &= [4,1,1,0,0,0,3,1] \\ H(B) &= [1,4,1,0,0,0,3,1] \\ H(C) &= [2,1,1,0,0,0,3,3]. \end{aligned}$$

We will use these three univariate histograms throughout the rest of this paper. Fig. 3.3 illustrates the minimum difference of pair assignments where $D_{\text{nom}}(A,C) = 2$, $D_{\text{ord}}(A,C) = 14$ and $D_{\text{mod}}(A,C) = 2$.

3.2.3 Normalization

The numbers of collected samples for different classes are not always the same in practice. Thus, we provide a general definition for histogram distributions with arbitrary sample size. Let $N = \text{CM}(n_A, n_B)$ be the *common multiple* of n_A and n_B where n_A and n_B are the numbers of samples in integer in set A and B. One common multiple



(a) Nominal: D nom[A,C] = 2 (b) Ordinal: D ord [A,C] = 14 (c) Modulo: Dmod[A,C] = 2

Figure 3.3: *Distances between $H(A)$ and $H(C)$.*

is $N = n_A \times n_B$. Now we can obtain the new histograms $H^N(A)$ and $H^N(B)$ with the same size of samples by following equations (3.10) and (3.11) on each level.

$$H_l^N(A) = n_B H_l(A) \tag{3.10}$$

$$H_l^N(B) = n_A H_l(B) \tag{3.11}$$

The normalized distance is defined as follows:

$$\begin{aligned} D^N(H(A), H(B)) &= \frac{D(H^N(A), H^N(B))}{N} \\ &= \frac{\sum_{i=0}^{b-1} |\sum_{j=0}^{i}(H_j^N(A) - H_j^N(B))|}{N} \end{aligned} \tag{3.12}$$

Output values in Eq.(3.12) are real numbers while those in Eq.(3.27) are integer values. Eq.(3.12) is the general and normalized form of Eq.(3.27) as all metric properties are preserved.

Lemma 3.2.5 *Let N_1 and N_2 be two common multiples. The normalized distances by any common multiple are the same.*

$$\frac{D(H^{N_1}(A), H^{N_1}(B))}{N_1} = \frac{D(H^{N_2}(A), H^{N_2}(B))}{N_2}$$

Proof: Consider the least common multiple, $N_0 = \mathrm{LCM}(n_A, n_B)$. Then all other common multiples, α are $N_\alpha = cN_0$ where c is a positive integer. The histogram

$H^{N_\alpha}(A)$ can be viewed as $H^{N_\alpha/c}(A)$. The minimum distance of $D(H^{N_\alpha}(A), H^{N_\alpha}(B))$ is equivalent to $c \times D(H^{N_\alpha/c}(A), H^{N_\alpha/c}(B))$, because we can consider c samples as one unit.

$$\begin{aligned} D^{N_\alpha}(H(A), H(B)) &= \frac{D(H^{N_\alpha}(A), H^{N_\alpha}(B))}{N_\alpha} \\ &= \frac{c \times D(H^{N_0}(A), H^{N_0}(B))}{N_\alpha} \\ &= \frac{D(H^{N_0}(A), H^{N_0}(B))}{N_0} \\ &= D^{N_0}(H(A), H(B)) \end{aligned}$$

∎

In order to show the triangle inequality property, consider multiple histograms, $H(A)$, $H(B)$ and $H(C)$ with different sizes.

Theorem 3.2.6 $D^{N_{AB}}(H(A), H(B)) \leq D^{N_{AC}}(H(A), H(C)) + D^{N_{CB}}(H(C), H(B))$

Proof: The matrix of normalized distances by individual two histograms is equivalent to the matrix of distances of $H^N(A)$, $H^N(B)$ and $H^N(C)$
where $N = \mathrm{CM}(n_A, n_B, n_C)$ by the lemma 3.2.5. Therefore,
$D^N(H(A), H(B)) \leq D^N(H(A), H(C)) + D^N(H(C), H(B))$ By definition of distance, $D^N(H(A), H(B))$ is the minimum distance between two histograms. Suppose $D^N(H(A), H(B)) > D^N(H(A), H(C)) + D^N(H(C), H(B))$, then $D^N(H(A), H(B))$ is not the minimum distance and it contradicts the definition. Therefore, the normalized distance holds the triangle inequality. ∎

3.2.4 Multivariate Case: Generalization

Although we considered histograms based on a single measurement variable, Eq.(3.9) is still valid for histograms based on several measurement variables. Only difference is how $d(a_i, b_j)$ is defined. For example, if all variables are ordinal, the *Euclidean* norm in vector space, denoted by $||\cdot||$, is often used for the difference between quantized measurement levels:

$$d_{\text{ord}}(a_i, b_j) = ||a_i - b_j|| \tag{3.13}$$

Various distance measures [36] such as *Minkowski* or *Tanimoto* can be used in place of the *Euclidean* norm. However, designing the algorithms to compute the distance between multivariate histograms is non-trivial because there are $n!$ number of possible assignments and because of high dimensionality. We focus on efficient algorithms for the special case that is the univariate histogram in Section 3.4 since it is commonly encountered in problems such as histogram based image indexing.

3.3 Conventional definitions

There are several definitions of distance (or similarity) between histograms, based on vectors, probabilities, and clusters. Ten such distances defined in the literature are given below and denoted as D_1-D_{10}. Their inadequacy when they are used to compute the distance between certain type histograms is considered.

3.3.1 List of definitions

A histogram is treated as a b-dimensional vector, and hence the standard vector norms can be used as distances between two histograms, as follows:

$$\underline{\text{City block } (L_1\text{-norm})}:\ D_1(A,B) = \sum_{i=0}^{b-1} |H_i(A) - H_i(B)| \tag{3.14}$$

$$\underline{\text{Euclidean } (L_2\text{-norm})}:\ D_2(A,B) = \sqrt{\sum_{i=0}^{b-1} (H_i(A) - H_i(B))^2} \tag{3.15}$$

Another approach is a normalized similarity measure, $S(A,B)$ based on the intersection between two histograms [101]:

$$\underline{\text{Intersection}}:\ S(A,B) = \frac{\sum_{i=0}^{b-1} \min(H_i(A), H_i(B))}{\sum_{i=0}^{b-1} H_i(A)}$$

$$= \frac{1}{n}\sum_{i=0}^{b-1}\min(H_i(A), H_i(B)) \tag{3.16}$$

The intersection (3.16) of two histograms is the same as Bayes P_e, the minimum misclassification (or error) probability, which is computed as the overlap between two PDF's, $P(A)$ and $P(B)$ [35]. To compute this as a distance measure, we will convert $S(A, B)$ using the inverse operation: $n \times (1 - S(A, B))$:

$$\underline{\text{Non-Intersection}}:\ D_3(A, B) = n - \sum_{i=0}^{b-1}\min(H_i(A), H_i(B)) \tag{3.17}$$

Measures D_1-D_3 are widely used for histogram based image indexing and retrieval [78, 101].

The following lemma states that distance measures D_1 and D_3 are closely related when the size of two sets are equal. It suggests an alternative algorithm for $D_{\text{nom}}(A, B)$ later in section 3.4.1.

Lemma 3.3.1 $D_1 = 2 \times D_3$, *provided* $|A| = |B| = n$.

Proof: Since for two integers $H_i(A)$ and $H_i(B)$,
$\min(H_i(A), H_i(B)) = \frac{H_i(A)+H_i(B)-|H_i(A)-H_i(B)|}{2}$,
it follows that $\sum_{i=0}^{b-1}\min(H_i(A), H_i(B)) = \frac{2n-\sum_{i=0}^{b-1}|H_i(A)-H_i(B)|}{2}$. By rearranging the equation, $\sum_{i=0}^{b-1}|H_i(A)-H_i(B)| = 2n-2\sum_{i=0}^{b-1}\min(H_i(A), H_i(B))$. Thus, $D_1 = 2\times D_3$. ∎

Discrete versions of distance between probability density functions are also useful as distances between histograms as follows:

$$\underline{\text{K-L distance}}:\ D_4(A, B) = \sum_{i=0}^{b-1} P_i(B)\log\frac{P_i(B)}{P_i(A)} \tag{3.18}$$

$$\underline{\text{Bhattacharyya distance}}:\ D_5(A, B) = -\log\sum_{i=0}^{b-1}\sqrt{P_i(A)P_i(B)} \tag{3.19}$$

$$\underline{\text{Matusita distance}}:\ D_6(A, B) = \sqrt{\sum_{i=0}^{b-1}(\sqrt{P_i(A)} - \sqrt{P_i(B)})^2} \tag{3.20}$$

Note that *K-L* distance is not a true metric, rather it is the relative entropy.

Distance measures between clusters [35] can be regarded as distances between histograms as follows:

$$
\begin{aligned}
\underline{\text{Nearest-neighbor}}:\ D_7(A,B) &= \min_{a_i \in A, b_j \in B} d(a_i, b_j) && (3.21)\\
\underline{\text{Furthest-neighbor}}:\ D_8(A,B) &= \max_{a_i \in A, b_j \in B} d(a_i, b_j) && (3.22)\\
\underline{\text{Mean distance}}:\ D_9(A,B) &= d(m_A, m_B), && (3.23)\\
&\quad \text{where } m_A \text{ and } m_B \text{ are means of } A \text{ and } B\\
\underline{\text{Average distance}}:\ D_{10}(A,B) &= \frac{1}{n^2} \sum_{a_i \in A} \sum_{b_j \in B} d(a_i, b_j) && (3.24)
\end{aligned}
$$

where $d(a_i, b_j)$ can be defined according to measurement type.

3.3.2 Analysis of Distance Measures in Various Measurement Types

Distance D_1-D_6 are always the same regardless of the type of measurements. Each bin along the level is viewed as an individual independent feature because the correlation between these bins is not considered in computing the distance between two histograms. We shall examine definitions D_1-D_{10} when they are applied to each type of measurements.

Ordinal:

To show the inadequacy of D_1-D_6, consider the following example. Let x represent the length of fish in a pond. Let A, B and C in section 3.2.2 represent samples drawn from three ponds. We are interested in determining the statistical similarity of fish in each of the three ponds. Note that length is an ordinal measurement. We wish to find the histogram most similar to $H(A)$. $H(A)$ and $H(B)$ have more baby fish whereas $H(C)$ has more adult fish. Three fish out of ten in the group A differ by one inch each from the group B whereas two fish differ by seven inches from the group

C. Based on the distance between sets, $H(A)$ is closer to $H(B)$ rather than to $H(C)$. Definitions D_1-D_6 are excellent in counting the number of mismatches but do not consider the difference (inches in fish example) of each mismatch.

The distance values D_1-D_6 and D_{ord} computed by enumerated definitions are shown in Table 3.1.

Table 3.1: Comparisons of Distance Measures D_1-D_6 and D_{ord}

	A,B	A,C	B,C	$\arg\min(D_x(A,\cdot))$
D_1	6	4	6	C
D_2	4.24	2.83	3.74	C
D_3	3	2	3	C
D_4	0.42	0.19	0.33	C
D_5	0.11	0.04	0.09	C
D_6	0.45	0.30	0.41	C
D_{ord}	3	14	13	B

The smallest distance value between two histograms indicates the closest histogram pair. Note that only D_{ord} returns $H(B)$ as the histogram closest to $H(A)$ whereas D_1-D_6 return $H(C)$ as the closest.

The inadequacy of definitions D_1-D_6 on ordinal type histograms can be explained by the following *"shuffling invariance"* property. A distance measure between histograms is *"shuffling invariant"* if and only if the distance does not change when levels, $\{x_0, x_1, \cdots, x_{b-1}\}$ in histograms are permuted or reordered. Measures D_1-D_6 have this property of *"shuffling invariance"*. They are sums of individual distances of each level and due to the commutative law, the distances do not change when levels are permuted among themselves.

In case of ordinal type histograms, the levels cannot be shuffled because of the correlation among levels. If the resulting matrices are not affected by shuffling, the definition of distance is not suitable for ordinal or modulo type histograms. The extreme example in Section 3.1.2 convinces us why the conventional definitions are inappropriate for the ordinal type histograms as definitions D_1-D_6 fail to tell which histograms are more similar than others.

Now consider the distance measures between clusters as shown in Table 3.2. In

Table 3.2: Comparisons of Distance Measures D_7-D_{10} when used in ordinal measurement types

	A,B	A,C	B,C	$\arg\min(D_x(A,\cdot))$
D_7	0	0	0	tie
D_8	7	7	7	tie
D_9	0.3	1.4	1.1	B
D_{10}	3.02	3.36	3.18	B

the given example of three histograms, D_7 and D_8 return 0 and 7 for all cases and thus do not discriminate the distances $D(A,B)$, $D(A,C)$ and $D(B,C)$. The *mean* method, D_9, does return B as the closest one to the set A in ordinal measurement case. However, this method has a disadvantage that it does not discriminate multi-modal histograms. The mean value can be equal although one histogram is unimodal and the other is bimodal.

Finally, the average method, D_{10}, is quite compatible with the new proposed measure. However, its resulting matrix does not have the reflexivity property that is $D(A,A) \neq 0$ but $= 3.08$. Suppose that there is a set D that is identical to the set A. This measure does not return D as the closest set. Another disadvantage of this method is its high complexity, $O(n^2)$ whereas the new measure in Eq.(3.9) is much quicker and it is discussed in the following section.

Nominal:

Now suppose the measurement type is nominal. The distance measures D_1-D_6 return the exactly same matrix as the ordinal measurement type as given in Table 3.1. This is one disadvantage of D_1-D_6. Table 3.3 shows the comparisons of the newly proposed distance measure with D_7-D_{10} when they are used for the nominal type measurement. It is quite compatible with all measures from D_1 through D_6 and it

Table 3.3: Comparisons of Distance Measures D_7-D_{10} and D_{nom} when used in nominal measurement types

	A,B	A,C	B,C	$\arg\min(D_x(A,\cdot))$
D_{nom}	3	2	3	C
D_7	0	0	0	tie
D_8	1	1	1	tie
D_9	$\cdot$	$\cdot$	$\cdot$	Not applicable
D_{10}	0.81	0.78	0.81	C

is exactly the same as D_3 and $D_1/2$. According to the definition of the difference between quantized measurement levels given for the nominal type measurement in Eq.(3.2), some of distances between nominal type clusters are computed and shown in Table 3.3. Again, D_7 and D_8 are meaningless. Also, *Mean* cannot be defined for the nominal type measurement. D_{10} is criticized in the same way as in the ordinal type case.

Modulo:

Finally, consider modulo measurement type. Again, the distance measures from D_1 through D_6 return the exactly same matrix as the ordinal measurement type. Table 3.3 shows the comparisons of the newly proposed distance measure with D_7-D_{10} when they are used for the modulo type measurement. According to the definition

Table 3.4: Comparisons of Distance Measures D_7-D_{10} and D_{mod} when used in modulo measurement types

	A,B	A,C	B,C	$\arg\min(D_x(A,\cdot))$
D_{mod}	3	2	5	C
D_7	0	0	0	tie
D_8	4	4	4	tie
D_9	$\cdot$	$\cdot$	$\cdot$	Not applicable
D_{10}	1.58	1.44	1.62	C

of the difference between quantized measurement levels given for the modulo type measurement in Eq.(3.4), some of distances between modulo type clusters are shown in Table 3.4. Again, D_7 and D_8 are meaningless. Also, *Mean* cannot be defined for the modulo type measurement; what is the mean of $0°$ and $180°$? Is it $90°$, $270°$ or none? Note that D_{10} has the desirable property that it varies depending on the type of measurements. However, again D_{10} is criticized in the same way as in other measurement type cases.

3.4 Algorithms

A naive way to solve the distance between histograms, that is the minimum difference of pair assignments, can be exponential in time as there are $n!$ number of possible assignments. In this section, we introduce efficient algorithms for univariate histograms for each type of measurement variable: nominal $\Theta(b)$, ordinal $\Theta(b)$ and modulo $O(b^2)$ insofar as histograms are given.

For the nominal type histograms, the half of the city block distance shown later in Eq.(3.14) as a distance is equivalent to the minimum difference of pair assignments in Eq.(3.9). For ordinal and modulo type histograms, the new measure $D(H(A), H(B))$ can be realized as the necessary cell movements to transform one histogram into the target histogram as shown in Fig. 3.4. The minimum cost of moving cells within

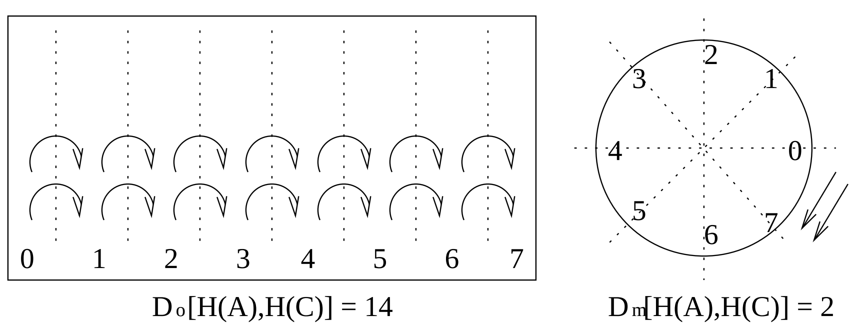


Figure 3.4: *Arrow representation of* $D_{ord}(H(A), H(B))$ *and* $D_{mod}(H(A), H(B))$.

a histogram to make the same configuration as the target one is equivalent to the minimum difference of pair assignments. There needs a few steps of moving cells if two histograms have similar distribution.

3.4.1 Nominal type histogram

A new distance between nominal type histograms is the number of samples that do not overlap or intersect, which is equivalent to Eq.(3.17). Hence, the new definition shown in Eq.(3.9) becomes:

$$\begin{aligned} D_{\text{nom}}(A,B) &= \min_{A,B}\Big(\sum_{i,j=0}^{n-1} d_{\text{nom}}(a_i,b_j)\Big) \\ &= n - \sum_{i=0}^{b-1}\min(H_i(A),H_i(B)) \end{aligned} \tag{3.25}$$

The algorithm for Eq.(3.25) is straightforward and we will not discuss it in detail here. As an alternative algorithm, one can solve this problem using the City Block Distance in Eq.(3.14) as discussed in Lemma 3.3.1.

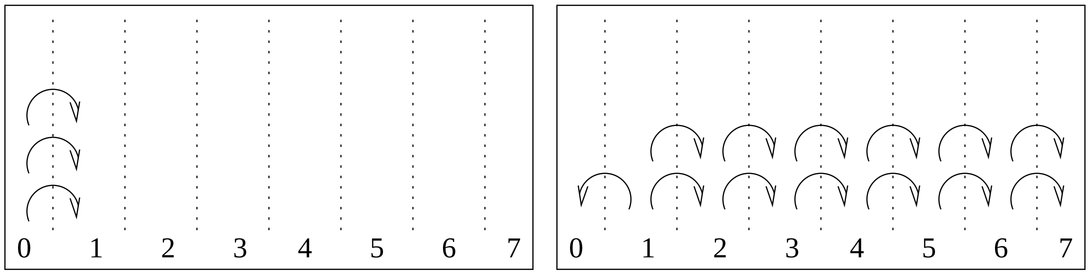
$$D_{\text{nom}}(A,B) = \frac{\sum_{i=0}^{b-1}|H_i(A)-H_i(B)|}{2} \tag{3.26}$$

In either equation, the computational time complexity is $\Theta(b)$.

3.4.2 Ordinal type histogram

A ordinal type histogram is a histogram whose level, x increases linearly. Many histograms fall into this category such as grey level, height, weight, length, temperature, and so forth. Earlier work on ordinal type histogram, motivated to expedite the image template matching problem, has been introduced briefly [11]. In this section, we present the detailed description and analysis of the algorithm to compute the distance between two ordinal type histograms.

As discussed earlier, a histogram $H(A)$ can be transformed into $H(B)$ by moving samples to left or right and the total of all necessary minimum movements is the distance between them. There are two operations. Suppose a cell or sample s belongs to a bin l. One operation is *Move left*(s). This operation results that the cell s belongs to a bin $l-1$ and the cost to do so is 1. This operation is impossible for cells in the left-most bin. Another operation is *Move right*(s). Similarly, after the operation, s belongs to the bin $l+1$ and the cost is 1. The same restriction applies to the right-most bin. These operations are expressed in the arrow representation of two histograms as shown in Fig. 3.5. Fig. 3.5 (a) shows the minimum number of cell



(a) D[H(X),H(Y)]

(b) D[H(Y),H(Z)]

Figure 3.5: *Arrow representation.*

operations required to transform of $H(A)$ into $H(B)$. The total number of arrows is the distance. It is the shortest movement and there is no other way to move cells in shorter steps to build the target histogram. In general, the number of arrows to transform $H(A)$ into $H(B)$ is equivalent to the cost of assigning samples in A to those in B and the minimum number of arrows necessary to transform them, hence $D(H(A), H(B))$ is equivalent to $D(A, B)$ in Eq.(3.9).

There are only one type of directional arrows along the border line between two levels or bins in order for the arrow representation to be the minimum. If there is a border line containing both directional arrows, they can be cancelled out without affecting the transform $H(A)$ into $H(B)$. Cancelling out operation reduces the total costs by two. This means that the arrow representation with mixed directions on a border line is not the minimum cost configuration.

The distance in histograms, that is the minimum number of necessary arrows in the arrow representation, is defined as follows for ordinal type histograms:

$$D_{\text{ord}}(H(A), H(B)) \quad = \quad \sum_{i=0}^{b-1} |\sum_{j=0}^{i}(H_j(A) - H_j(B))| \tag{3.27}$$

It is the sum of absolute values of prefix sum of difference for each level. Therefore, the algorithm for finding the minimum distance between two histograms consists of three steps. The first step is to obtain the differences for each level. The second step is to calculate the prefix sum of the differences for each level. Finally, the absolute values of the prefix sums are added. The following pseudo code shows the exact steps.

Algorithm 1 Distance-ordinal-histogram(int *A, int *B)

```
prefixsum = 0
h_dist = 0
for i = 0 to b-1
    prefixsum += A[i] - B[i]
    h_dist += |prefixsum|
return(h_dist)
```

For the example of $H(A)$ and $H(C)$, Algorithm 1 performs the following calculations:

$$\begin{array}{cccccccccccc}
4 & 1 & 1 & 0 & 0 & 0 & 3 & 1 & & & \cdots & (1) \\
2 & 1 & 1 & 0 & 0 & 0 & 3 & 3 & & & \cdots & (2) \\
\hline
2 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & & & \cdots & (3) \\
2 & 2 & 2 & 2 & 2 & 2 & 2 & 0 & \Rightarrow & 14 & \cdots & (4)
\end{array}$$

The line (1) and (2) represent the histogram $H(A)$ and $H(C)$, respectively and the line (3) is the difference between elements in (1) and (2) on each level. The line (4) is the prefix sum of the elements in line (3). Note that the last element in the prefix sum list is always 0 since both histograms are of same size. The final step is adding the absolute value of each element in the prefix sum list, which is 14.

Both time and space complexities are $\Theta(b)$. The algorithm requires only two integer variables and two arrays for histograms.

Correctness

The following lemma is crucial since it will serve as a stepping stone to support the algorithm. Suppose that we have successfully constructed the arrow representation of the histograms such that the distance is the minimum.

Lemma 3.4.1 *Let A_l denote the number of arrows from the bin l to $l+1$. It is positive if arrows are heading to right, or negative otherwise.*

$$A_l = \sum_{i=0}^{l} H_i(A) - \sum_{i=0}^{l} H_i(B) = \sum_{i=l+1}^{b-1} H_i(B) - \sum_{i=l+1}^{b-1} H_i(A)$$

Proof: Consider two sub-histograms, $H_{0..l}(A)$ and $H_{0..l}(B)$ where bins are 0 to l. After transforming, population of $H_{0..l}(B) + A_l$ must be equal to that of $H_{0..l}(A)$. Suppose $A_l \neq \sum_{i=0}^{l} H_i(A) - \sum_{i=0}^{l} H_i(B)$. Then there is no way to transform $H_{0..l}(A)$ to $H_{0..l}(B) + A_l$. By contradiction,

$$A_l \quad = \quad \sum_{i=0}^{l} H_i(A) - \sum_{i=0}^{l} H_i(B) \tag{3.28}$$

Now the total population is $n = \sum_{i=0}^{l} H_i(A) + \sum_{i=l+1}^{b-1} H_i(A)$ and $\sum_{i=0}^{l} H_i(A) = n - \sum_{i=l+1}^{b-1} H_i(A)$. Similarly, $\sum_{i=0}^{l} H_i(B) = n - \sum_{i=l+1}^{b-1} H_i(B)$. Replacing the terms in (3.28), $A_l = \sum_{i=l+1}^{b-1} H_i(B) - \sum_{i=l+1}^{b-1} H_i(A)$ ∎

The lemma implies that A_l is the difference of populations between two sub-histograms in the left side of the border line of the bin l and $l+1$.

Theorem 3.4.2 *Algorithm 1 correctly finds the minimum distance between two histograms.*

Proof: As Lemma 3.4.1 is true for all levels, the minimum distance is $\sum_{i=0}^{b-1} |A_i| = \sum_{i=0}^{b-1} |\sum_{j=0}^{i} H_j(A) - \sum_{j=0}^{i} H_j(B)|$. This is equivalent to the Eq.(3.27), i.e.
$\sum_{i=0}^{b-1} |\sum_{j=0}^{i}(H_j(A) - H_j(B))|$. ∎

3.4.3 Modulo type histogram

One major difference in an modulo type histogram is that the first bin and the last bin are considered to be adjacent to each other, and hence it forms a closed circle, due to the nature of the data type. Transforming such an modulo type histogram should allow cells to move from the first bin to the last bin or vice versa at a cost of a single movement. This results in a different distance value in modulo type histograms from the one in ordinal type histograms. The same histograms $H(A)$, $H(B)$ and $H(C)$ are now treated as modulo type histograms and redrawn as shown in Fig. 3.6. The

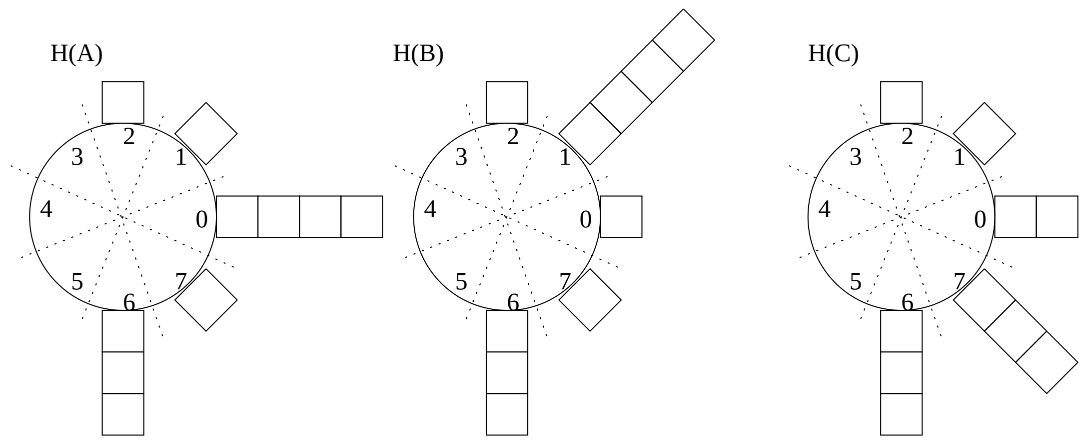


Figure 3.6: *Modulo representation of* $H(A)$*,* $H(B)$ *and* $H(C)$

number inside of each slice represents the level of a bin. Table 3.4 indicates that the two histograms $H(A)$ and $H(C)$ are the closest pair and $D(H(A), H(C)) = 2$ is achieved by moving two cells from bin 0 in $H(A)$ to bin 7 clockwise. Clearly, the difference in measurement type necessitates a new algorithm to find the distance between modulo type histograms. In this section, we modify Algorithm 1 to construct the algorithm for distance between modulo type histograms.

Properties

Before embarking on the new algorithm, it is important to discuss the properties of the arrow representation of the distance between two modulo type histograms. Consider another modulo type histograms, $H(D)$ and $H(E)$ as shown in Fig. 3.7. Blocks or cells can move to clockwise or counter-clockwise directions. Each cell movement to the next level in either direction costs 1. The minimum cost required to build the target histogram from a given histogram is the distance. Again, an intuitively appealing way to explain the distance is to use the arrow representation of two histograms as shown in 3.7. If one establishes an arbitrary one-to-one mapping for the cells between

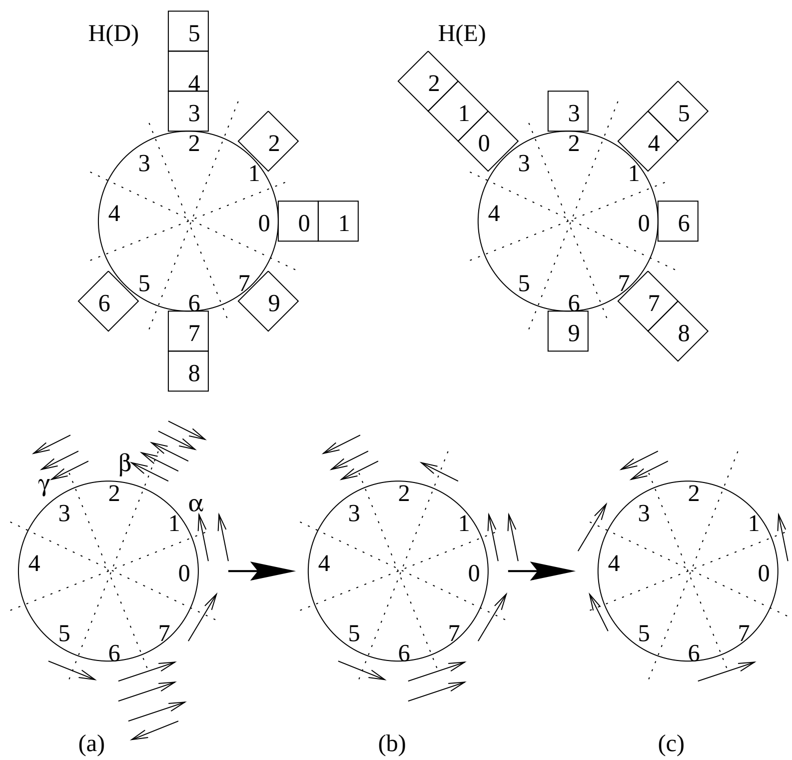


Figure 3.7: *Modulo Histograms and angular arrow representation*

two histograms, one can transform $H(D)$ into $H(E)$ by moving cells in $H(D)$ to the corresponding position in $H(E)$. For the example in Fig. 3.7 (a), the arrows α, β and γ indicate the path from the cell 0 in $H(D)$ to the cell 0 in $H(E)$. There are $n!$ number of ways to transform in this manner. Among these ways, there exists a minimum distance whose number of movements is the lowest. Some sample movements are illustrated as an arrow representation in Fig. 3.7. As a matter of fact, an modulo representation that satisfies the following properties gives the minimum configuration of $D(H(D), H(E))$.

Property 1 *Arrows must be one directional on each border line.*

Property 2 *The number of border lines of one direction cannot exceed $b/2$.*

Property 3 *No more reduction occurs when either basic operations are applied.*

As discussed before in ordinal type histogram case, if there is a border line that has both directional arrows, they are cancelled out. These movements are redundant ones. The configuration in Fig. 3.7 (a) becomes one in Fig. 3.7 (b) by cancelling out the opposite arrows on each border line. By the property 1, there exists no border line of mixed directional arrows and each border line has either clockwise or counter-clockwise directional arrows. Suppose that the number of border lines of one direction is $b' > b/2$ and the first number of arrows were k, then after adding the circle, the number of arrows becomes $k + b$. By cancelling out the opposite directions on the same border line, we have $k + b - 2b' < k$. Therefore, if the number of border lines of one direction exceeds $b/2$, this is not a configuration of the minimum distance. From these properties, two important basic operations are derived as shown in Fig. 3.8. A complete circle in the chain of same directional arrows can be added in any direction and then the opposite arrows on the same line are cancelled out.

Lemma 3.4.3 *Let A_l denote the number of counter-clockwise arrows from bin $l-1$ to l. A_l is positive if arrows are counter-clockwise and negative otherwise. $H_l(A) = H_l(B) + A_l - A_{l+1}$*

Proof: Similar to Lemma 3.4.1. After transforming, population of cells on each level of both histograms must be the same. ∎

Since Lemma 3.4.3 is true for all levels, affecting cells in one bin means affecting all other bins as a chain reaction. Hence, there are only two possible operations to affect changes as shown in Fig. 3.8. The arrow representation of the minimum distance value

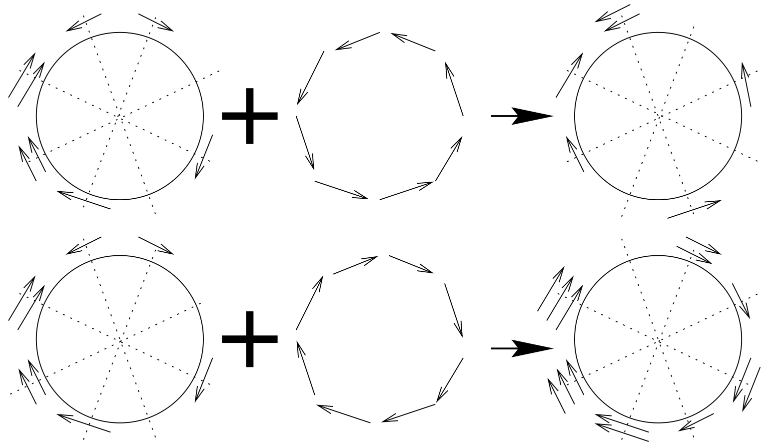


Figure 3.8: *Two basic operations.*

is always constructible by the combination of these two basic operations. Consider a non-minimum distance arrow representation. By applying one of two operations of adding a clockwise or counter-clockwise circle, the lower number of arrows is achieved whereas it is unchanged or increased if the arrow representation has the minimum distance. Although first two properties are sufficient for the the arrow representation of the minimum distance, the third property alone is also sufficient since it admits the other properties.

$D_{\mathbf{mod}}$ Algorithm

An algorithm to compute the distance between modulo type histograms in $O(b^2)$ is presented. It gets an initial arrow representation from Algorithm 1 and then use two basic operations to derive the minimum distance arrow representation that guarantees all properties discuss in the previous section.

Algorithm 2 Distance-modulo-histogram(int *A, int *B)

1 prefixsum[0] = A[0] - B[0]

2 *for* $\forall i$ prefixsum[i] = prefixsum[i-1] + A[i] - B[i]

3 h_dist = $\sum_{i=0}^{b-1}$ |prefixsum[i]|

```
4  for(;;)
5      d = min(positive prefixsum[i])
6      for ∀i temp[i] = prefixsum[i] - d
7      h_dist2 = Σ_{i=0}^{b-1} |temp[i]|
8      if h_dist2 < h_dist
9          h_dist = h_dist2
10         for ∀i prefixsum[i] = temp[i]
11     else break;
12 for(;;)
13     d = max(negative prefixsum[i])
14     for ∀i temp[i] = prefixsum[i] - d
15     h_dist2 = Σ_{i=0}^{b-1} |temp[i]|
16     if h_dist2 < h_dist
17         h_dist = h_dist2
18         for ∀i prefixsum[i] = temp[i]
19     else break;
20 return(h_dist)
```

The algorithm is explained using the example shown in Fig. 3.7 along with the following calculations:

2	1	3	0	0	1	2	1			$\cdots$	(1)
1	2	1	3	0	0	1	2			$\cdots$	(2)
1	-1	2	-3	0	1	1	-1			$\cdots$	(3)
1	0	2	-1	-1	0	1	0	$\Rightarrow$	6	$\cdots$	(4)
1	0	2	-1	-1	0	1	0	$\Rightarrow$	6	$\cdots$	(5)
0	-1	1	-2	-2	-1	0	-1	$\Rightarrow$	8	$\cdots$	(6)

The line (3) is the difference between (1) and (2) (step 1 and 2 in Algorithm 2). The line(4) is the initial arrow representation that is the prefix sum of the difference and the sum of the absolute value of these numbers (step 3). Note that step 1 through 3 are exactly the same as Algorithm 1 that guarantees the property 1.

To ensure the properties 2 and 3, two basic operations in Fig. 3.8 are applied repeatedly. First, circles of clockwise arrows are added to the current arrow representation until there is no more reduction on the total number of arrows (step 4 through 11). The line (5) is the result of these steps. Next, circles of the counter-clockwise arrows are added in the similar manner (step 12 through 19). The line (6) is the result of adding a circle and the resulting value is greater than the previous one. Therefore, the distance is 6.

Correctness

The correctness of the algorithm is asserted by the following theorem.

Theorem 3.4.4 *Algorithm 2 correctly finds the minimum distance between two modulo histograms.*

Proof: The arrow representation of minimum distance can be achieved from any arbitrary valid arrow representation by a combination of two basic operations. Fig 3.9 illustrates the relation between valid arrow representations. The arrows indicate one

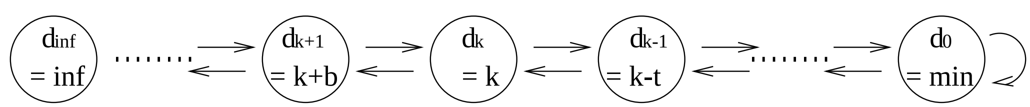


Figure 3.9: *relation between valid arrow representations.*

of the basic operations and the opposite arrow represents the other basic operation. All valid representations are related as a string and the distance value can increase infinitely. There exists only one minima among valid arrow representations. In order to reach to the minima, first test for the one of the two operations whether it gives higher or lower distance value. If the distance reduces, keep applying the operation until no more reduction occurs. Otherwise, check for the other operation in similar manner. Algorithm 2 first computes an arrow representation by Algorithm 1 and then applies the clockwise operation repeatedly until no more reduction occurs and then the counter-clockwise operation similarly. This guarantees the property 3. Therefore, Algorithm 2 is correct. ∎

Algorithm 2 runs in $O(b^2)$ time. The line 1 through 3 is $\Theta(b)$. The line 4 through 11 takes $O(b^2)$ because each iteration removes at least one positive number in the list and there can be up to $b-1$ number of positive numbers in the arrow representation. Similarly, the line 12 through 19 takes $O(b^2)$.

Theorem 3.4.5 *The worst-case time complexity of Algorithm 2 is $O(b^2)$.*

Proof: Here is a worst-case example of two modulo histograms with 30 samples and 10 bins.

5	5	5	5	5	1	1	1	1	1			⋯	(1)
1	1	1	1	1	5	5	5	5	5			⋯	(2)
4	4	4	4	4	-4	-4	-4	-4	-4			⋯	(3)
4	8	12	16	20	16	12	8	4	0	⇒	100	⋯	(4)
0	4	8	12	16	12	8	4	0	-4	⇒	68	⋯	(5)
-4	0	4	8	12	8	4	0	-4	-8	⇒	52	⋯	(6)
-8	-4	0	4	8	4	0	-4	-8	-16	⇒	56	⋯	(7)

The distance is 52. The worst case is that the size of either positive or negative consecutive numbers is $b-1$. Each iteration reduces the size by 1 or 2. Therefore, the running time is $O(b^2)$. ∎

The space required for the algorithm is $O(b)$.

3.5 Experiment on Character Writer Identification

We show the experimental results of the character writer identification problem using the earlier defined new measure on *Gradient directions* histograms. We consider this modulo type histogram features of a character as character level image signatures for identification.

3.5.1 Gradient Direction Histogram

Gradient direction features are computed by the following *Sobel edge detection mask* operators [106] where $I(i,j)$ represents the image of a character.

$$\underbrace{\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}}_{Rowmask}\underbrace{\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}}_{Columnmask}$$

$$\begin{aligned} S_x(i,j) &= I(i-1,j+1)+2I(i,j+1)+I(i+1,j+1) \\ &\quad -I(i-1,j-1)-2I(i,j-1)-I(i+1,j-1) \\ S_y(i,j) &= I(i-1,j-1)+2I(i-1,j)+I(i-1,j+1) \\ &\quad -I(i+1,j-1)-2I(i+1,j)-I(i+1,j+1) \end{aligned}$$

$$\begin{aligned} \text{magnitude} &= \sqrt{S_x^2(i,j)+S_y^2(i,j)} \\ \text{direction} &= \tan^{-1}\frac{S_y(i,j)}{S_x(i,j)} \end{aligned} \tag{3.29}$$

A sample of the gradient direction maps of a character image is shown in Fig. 3.10

3.5.2 Sample "W" characters and histograms

We consider off-line versions of the pages of handwriting captured by the Human Language Technology (HLT) group at CEDAR [89]. The database contains both cursive and printed writing. Three character "W"'s per author are extracted and their gradient direction histograms are computed. There are four writers, $\{A,B,C,D\}$ and the corresponding angular representation of gradient direction histograms are shown in Fig. 3.12.

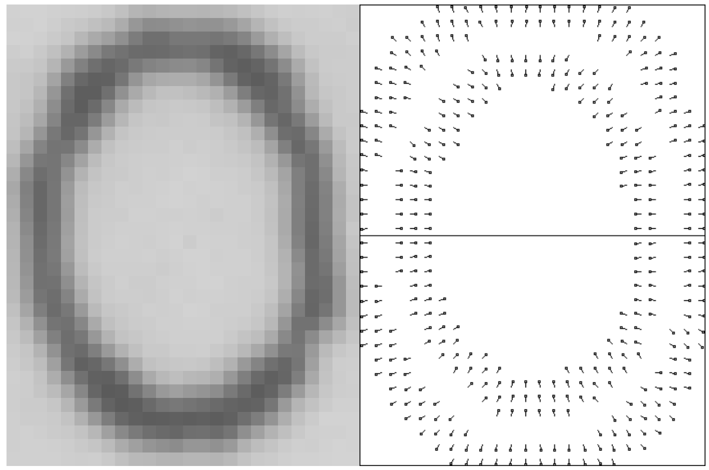
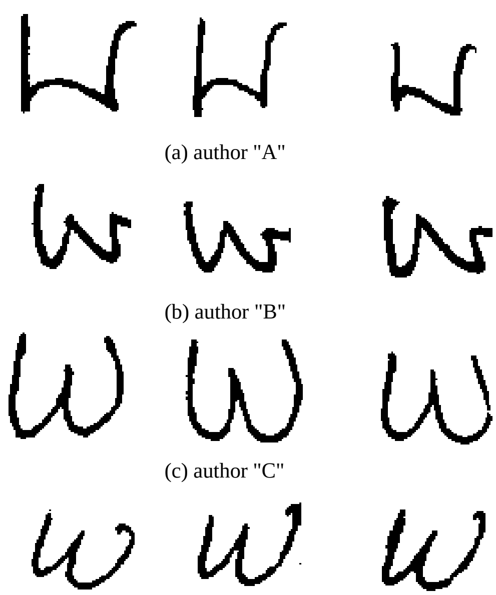


Figure 3.10: *Gradient direction map*



(a) author "A"

(b) author "B"

(c) author "C"

(d) author "D"

Figure 3.11: *Sample W's*

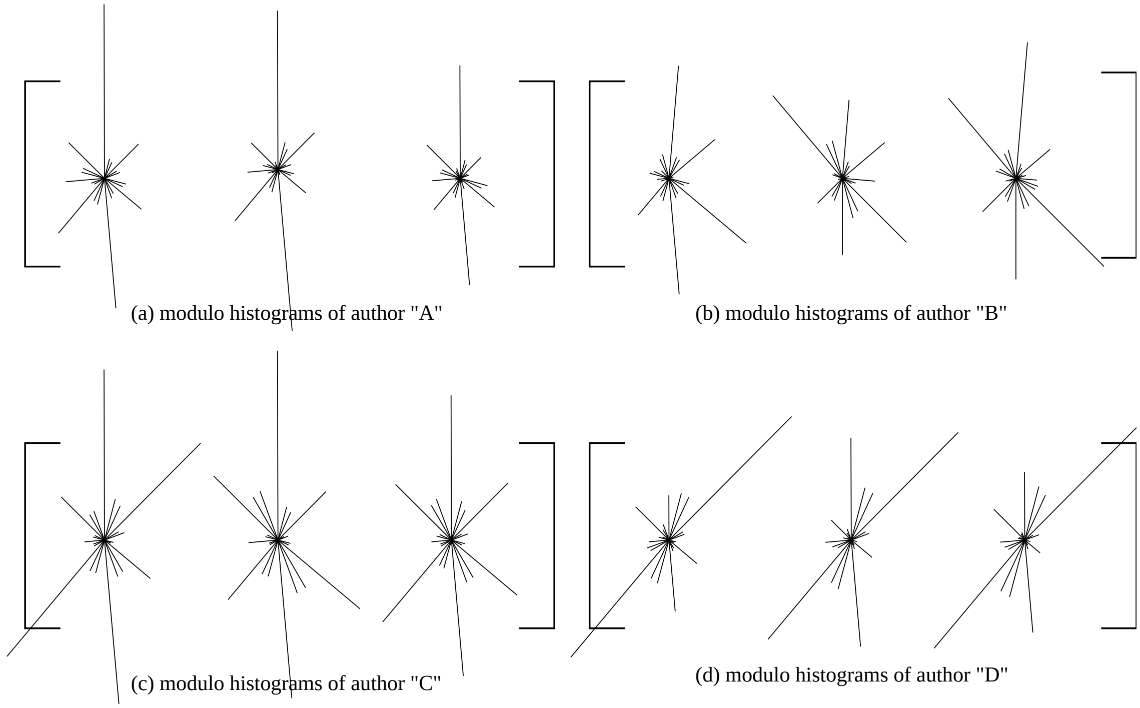


Figure 3.12: *Angular Representation of gradient direction histograms for sample W's in Fig. 3.11*

Table 3.5 shows the distance matrix. When two-dimensional information is represented in the one-dimensional histogram, certain information is lost. Therefore, while it is true that the two histograms from the similar character images tend to be similar, the reverse statement is not always true that two images with the similar histograms tend to be similar. For example, the second sample histogram from author "A" is similar to the first sample from author "C" although their characters are dissimilar. Yet, this histogram distance information are very helpful features to determine the similarity of two letters and we claim that distances in Tab. 3.5 tend to be small if they were written by a same author.

3.6 Conclusions and Future Work

We have criticized inadequacy of the way that existing definitions, D_1-D_6 are used for ordinal and modulo type histograms. We considered three types of histograms

Table 3.5: D^N_{mod} Matrix of Gradient Direction Histograms of Writers

	A-1	A-2	A-3	B-1	B-2	B-3	C-1	C-2	C-3	D-1	D-2	D-3
A-1	0.00	0.99	0.86	1.85	2.24	2.14	1.54	1.59	1.22	3.15	2.94	3.43
A-2	0.99	0.00	1.41	2.20	2.67	2.64	1.39	1.58	1.17	2.68	2.44	2.91
A-3	0.86	1.41	0.00	1.26	1.56	1.34	2.06	1.19	1.35	3.79	3.69	4.22
B-1	1.85	2.20	1.26	0.00	1.00	0.87	2.75	1.26	1.43	4.65	4.54	5.07
B-2	2.24	2.67	1.56	1.00	0.00	0.82	3.23	1.42	1.81	5.12	5.01	5.55
B-3	2.14	2.64	1.34	0.87	0.82	0.00	3.22	1.41	1.80	5.11	5.00	5.54
C-1	1.54	1.39	2.06	2.75	3.23	3.22	0.00	2.12	1.41	2.19	1.92	2.48
C-2	1.59	1.58	1.19	1.26	1.42	1.41	2.12	0.00	0.78	4.04	3.91	4.44
C-3	1.22	1.17	1.35	1.43	1.81	1.80	1.41	0.78	0.00	3.34	3.20	3.74
D-1	3.15	2.68	3.79	4.65	5.12	5.11	2.19	4.04	3.34	0.00	1.06	0.85
D-2	2.94	2.44	3.69	4.54	5.01	5.00	1.92	3.91	3.20	1.06	0.00	1.20
D-3	3.43	2.91	4.22	5.07	5.55	5.54	2.48	4.44	3.74	0.85	1.20	0.00

characterized by their measurement type: nominal, ordinal and modulo. Different algorithms are designed for each type of histograms; Eq.(3.25), Algorithm 1 and Algorithm 2, correspondingly. Their computational time complexities are $\Theta(b)$, $\Theta(b)$ and $O(b^2)$, respectively, insofar as the histograms are given. These algorithms are based on one universal concept of distance between sets that is the problem of minimum difference of pair assignments.

We introduced the problem of minimum difference of pair assignments to grasp the concept of the distance between two histograms. Extending the suggested algorithms 1 and 2 facilitates the solution to this problem in $\Theta(n + b)$ and $O(n + b^2)$ time for ordinal and modulo type univariate data, respectively, reflecting the time complexity of $\Theta(n)$ to build the histogram.

As applications, one can use the measure directly or indirectly to solve the problem of classification, clustering, indexing and retrievals, i.e., image indexing and retrieval based on grey scales or hue value histograms. We strongly believe that there are a plethora of applications in various fields.

Multivariate Histograms

Albeit the histograms that we dealt with in this paper are one dimensional arrays (univariate), there can be any dimensional ones and measuring the distance between multivariate histograms in Eq.(3.13) can be useful in many applications. For example, grey scale images can be considered as two dimensional histograms. The concept of distance introduced in this paper might be generalized and realized for the image similarity. Another challenging problem occurs when variables of histograms are different in type. We leave them as open problems to readers.

Chapter 4

Edit Distance for Approximate String Matching

[1] A string is a sequence of symbols drawn from the alphabet Σ and it is one of the most popular representations of pattern [48]. The approximate string matching algorithm is often used to discriminate between two patterns and it is one of widely studied areas in computer science due to a variety of applications such as genetics and DNA sequence analysis, a spelling corrector, etc [90, 98, 36]. Earlier definition and solution for the traditional *approximate string matching* problem are found in the literature [108, 98] and extensive surveys on various techniques are shown in [51]. It computes the *edit distance*, also known as *Levenshtein distance* that is the minimum number of *indels* (insertions and deletions) and substitutions needed to transform one string to another.

In the earlier definition, the symbol types are assumed to be nominal while the measurement types made on the symbols are various. We consider four versions of the string distance measure, corresponding to whether the type of measurement is nominal, angular, magnitude and cost-matrix. Since the original *Levenshtein* or the *Levenshtein with cost-matrix* edit distances are inadequate for either angular or magnitude, we propose a modified edit distance. The algorithm for the newly defined edit distance uses the *dynamic programming* method of computing the edit distance between two strings [85].

"Find all letters that look like this letter." Such a query has received a great

[1]This chapter contains works published in [14, 15] and is under review in IEEE TPAMI [16].

deal of attention in *character recognition* and *handwriting analysis.* This problem has been formalized by defining a distance metric between characters and finding the nearest characters from the reference set. One promising distance measure is using the edit distance between strings after extracting the on-line stroke and off-line contour sequence strings. This approach has been studied by many researchers [66, 77, 14, 15] dating as far back as 1975 [45]. *Fujimoto et al* developed the *OCR* using the idea of *"Nonlinear Elastic Matching"* to read hand-printed alpha-numerics and Fortran programs [45]. The edit distance with cost matrix technique [108] was used to solve the on-line [77] and off-line [66] character recognition problems. Since symbols in both *Freeman style* on-line stroke and off-line contour sequence strings are not nominal type but angular type, we extend *Levenshtein edit distance* to handle this measurement type utilizing the *"turn"* concept in place of *substitution* for angular strings and takes local context into account in *indels.* We compare the newly defined edit distance with the edit distance with cost matrix which uses predetermined costs for *indels.*

We claim that the smaller the edit distance is, the more similar two characters look like to each other. To appreciate the usage of the approximate string matching for angular string elements in *pattern classification*, we consider three applications: *writer verification, on-line* and *off-line character recognitions.* First, in *writer verification*, one would like to find out whether the given two letters are similar or whether they were written by the same author [88, 6, 80]. The *form* or *shape* is an important feature for characterizing individual handwriting as it is quite consistent with most writers in normal undisguised handwriting [6]. The form can be described by a sequence of directional strokes. For the *on-line character recognition* problem(see [82, 72, 102] for comprehensive and detailed survey on successful techniques and applications of on-line handwriting recognition), the stroke sequence string is obtained from the movements of a mouse or a pen-based device. As a stroke sequence signifies the shape of the individual letters, a letter *"a"* is distinguished from a letter *"b"* by its different stroke sequences. Finally, in *off-line character recognition*, contour sequence strings derived from a *Freeman style chain-code* are used.

The rest of this paper is organized as follows. In section 4.1, four types of strings are defined according to their element type: nominal, angular, linear and cost-matrix type strings. Section 4.2 presents the newly defined approximate string matching algorithm for stroke direction sequence strings. Also, the difference between the newly proposed edit distance and Levenshtein edit distance with cost matrix is stated. In section 4.3, we discuss three applications: *writer verification*, *on-line* and *off-line character recognitions*. Finally, section 4.4 concludes this work.

4.1 Type of Strings

A string is a sequence of symbols drawn from the alphabet Σ. We will use the following notations and symbols throughout the rest of this paper.

$$\begin{aligned} S_1 &= (s_{1,1}, s_{1,2}, \cdots, s_{1,n_1}) \\ S_2 &= (s_{2,1}, s_{2,2}, \cdots, s_{2,n_2}) \end{aligned}$$

S_1 and S_2 are strings and n_1 and n_2 denote the length of each string. Each symbol in the string, $s_{x,y}$ has two index labels where the first index, x indicates the string to which it belongs and the second index, y indicates the location of the symbol in the string $(1 \leq y \leq n_x)$. We consider four types of strings based on the measurement type of the alphabet: *nominal, angular, linear* and *cost-matrix*.

Different types of strings have been considered in the literature of string matching according to three models of alphabet: constant size alphabet, unbounded alphabet and integer alphabet. In this taxonomy, algorithmists divided strings to find the efficient computational time matching algorithms. Briefly to summarize the history, Landau and Vishkin suggested an efficient algorithm; after building suffix trees in $O(f(n))$, the constant time *Least Common Ancestor* algorithm [52] is applied k times for every location of the text [64]. The time complexity is $O(f(n) + kn)$; the time complexity of building a suffix tree depends on three models of alphabet: a constant size alphabet, integer alphabet and unbounded alphabet. Weiner, who introduced the suffix tree, gave $O(n)$ suffix tree construction algorithm for the constant size alphabet [109]. Farach showed an algorithm to build a suffix tree in linear time when symbols are drawn from an integer alphabet [39]. Finally, $\Omega(n \log n)$ algorithm is known for the unbounded alphabet. In our version of categorization, however, we consider four types of strings according to their measurement to design suitable edit distances for each type not to design the efficient algorithm.

4.1.1 Nominal type string

In a nominal type string, each symbol value is named, e.g. string of English alphabets or DNA sequences $[A - A - C - T - G \cdots - A - A]$ where the letters A, G, C, and T stand for the *nucleic acids adenine, guanine, cytosine*, and *thymine*. The distance between two nominal type symbols are either a match or a mismatch:

$$d(s_{1,i}, s_{2,j}) \quad = \quad \begin{cases} 0, & \text{if } s_{1,i} = s_{2,j} \\ 1, & \text{otherwise} \end{cases} \tag{4.1}$$

Traditional approximate string matching problem takes nominal type strings as inputs [98, 51]; given a text string of length n, a pattern string of length m, the number k of differences (*indels* and substitutions) allowed in a match, find every location in the text where a match occurs. The *edit distance*, also known as *Levenshtein distance*, is the minimum number of *indels* and substitutions needed to transform one string to another. Nominal type strings are useful in *spelling correction* that is a post-processing stage of word recognition [90].

4.1.2 Angular type string

In the angular type string, symbol values are angular degrees. Stroke directions are quantized into r directional values as shown in Figure 4.1, e.g. *Freeman style chain-code scheme* has $r = 8$ directional values. One element is different from another

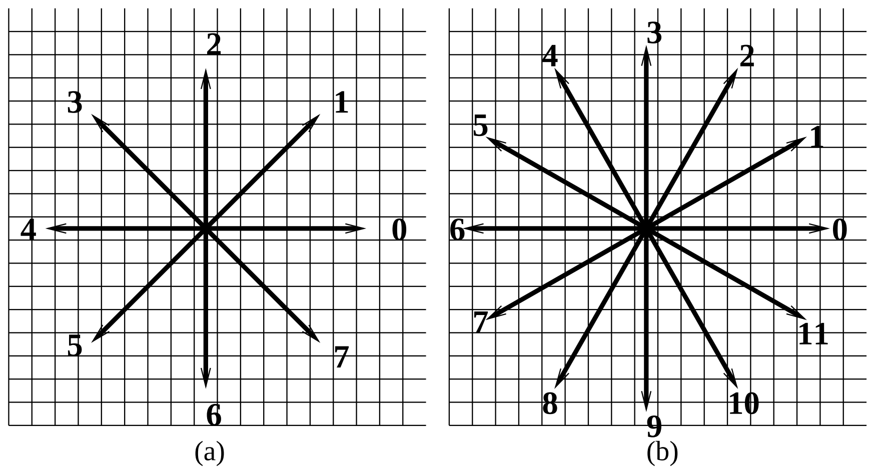


Figure 4.1: Number of stroke directions and length along horizontal and vertical axes: (a)Strokes with 8-directions and 7 pixel length, (b)Strokes with 12-directions and 8 pixel length.

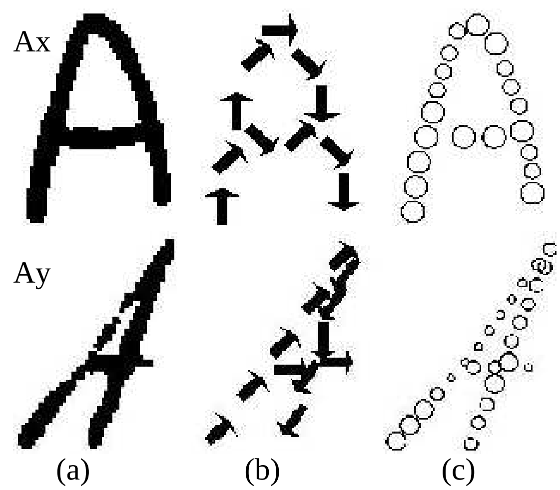
by a number of *turn* operations of changing direction by $\frac{360°}{r}$. The turning distance between two individual directional elements is defined as follows:

$$d(s_{1,i}, s_{2,j}) = \begin{cases} |s_{1,i} - s_{2,j}|, & \text{if } |s_{1,i} - s_{2,j}| \leq \frac{r}{2} \\ r - |s_{1,i} - s_{2,j}|, & \text{otherwise} \end{cases} \tag{4.2}$$

One can edit a stroke direction to make the other stroke direction by turning it clockwise or counter-clockwise whichever is shorter. For example of $r = 8$, $d(\uparrow, \rightarrow) = 2$ as one can turn $\uparrow$ two clockwise steps to make $\rightarrow$. Therefore, the term $d(s_{1,i}, s_{2,j})$ denotes the minimum number of necessary steps in turning.

The equally weighted substitution operation in nominal type strings is not suitable in angular type strings as a distance between two elements, $d(s_{1,i}, s_{2,j})$, is not simply a match or mismatch. Note that element values are categorical in nominal type strings whereas they represent a scale from small to large in angular type strings. For example, the difference between symbols "A" and "B" is the same as that between "A" and "C" whereas the difference between $\rightarrow$ and $\uparrow$ is smaller than that between $\rightarrow$ and $\leftarrow$. Motivated from the aforementioned argument, we utilize the turning distance between two angular string elements in place of substitution to compute the edit distance between two two angular type strings. This idea was originally implemented by Fugimoto and et al in 1975 and they called it *"Nonlinear Elastic Matching"*.

As shown in Figure 4.2, a character image can be represented in a piecewise linear stroke sequence string and we will use A_x and A_y examples the data structures and algorithms throughout the rest of this paper. The string of arrows is called a *Stroke*



$$
\begin{array}{lllllllllllllllll}
\mathrm{SDSS}(A_x) & = (\ \mathrm{SDSS}(A'_x) & = & [\uparrow & \nearrow & \uparrow & \nearrow & \rightarrow & \searrow & \downarrow & \searrow & \downarrow]) & & +(\ \mathrm{SDSS}(A''_x) & = & [\searrow & \nearrow]) \\
\mathrm{SPSS}(A_x) & = (\ \mathrm{SPSS}(A'_x) & = & [\bigcirc & \bigcirc & \bigcirc & \circ & \bigcirc & \bigcirc & \mathrm{O} & \bigcirc & \mathrm{O}]) & & +(\ \mathrm{SPSS}(A''_x) & = & [\bigcirc & \bigcirc]) \\
\mathrm{SDSS}(A_y) & = (\ \mathrm{SDSS}(A'_y) & = & [\nearrow & \nearrow & \nearrow & \nearrow & \nearrow & \swarrow & \swarrow & \downarrow & \swarrow & \swarrow]) & +(\ \mathrm{SDSS}(A''_y) & = & [\rightarrow & \rightarrow]) \\
\mathrm{SPSS}(A_y) & = (\ \mathrm{SPSS}(A'_y) & = & [\bigcirc & \bigcirc & \circ & \circ & \mathrm{O} & \mathrm{O} & \mathrm{O} & \bigcirc & \mathrm{O} & \mathrm{O}]) & +(\ \mathrm{SPSS}(A''_y) & = & [\mathrm{O} & \circ])
\end{array}
$$

Figure 4.2: Sample Stroke Direction and Pressure Sequences: (a)original character images (b) Angular Stroke Direction (c) Stroke Width(Pressure).

Direction Sequence String, or an *SDSS* in short and an *SDSS* is an angular type string. As a character may have more than one contiguous stroke, contiguous strokes are represented in a pair of parentheses, e.g. a capital letter *"A"*, $A_x = (A'_x) + (A''_x)$; a certain character letter is consisted of a sequence of one or more *SDSS*'s.

Choice of Number of Quantization Levels for Stroke Direction and Length

A digital off-line handwriting is represented by rectangular arrays of picture elements called *pixels*. A digital on-line handwriting is represented by a sequence of mouse positions where the positions are also pixels. Each pixel is a rectangle. If one moves from one pixel to another by k pixels in diagonal direction, the exact length of the path is not k but $k\sqrt{2}$. Unlike *Freeman code*, We would like to have all directional arrows to be the same length.

Let $s(l, r, i)$ be a stroke whose length is l and the number of directions is r. The last index, i, indicates the arrow number and thus $i \in \{0, \cdots, r-1\}$. For example, $s(7, 8, 1) = \nearrow$. To illustrate, suppose the arrow begins from the origin. Then the coordinate of $s(l, r, 1)$ must be $(l\cos\frac{2\pi}{r}, l\sin\frac{2\pi}{r})$. Due to the rectangular grid representation of pixels, the actual coordinate in a digital image is $(\lceil l\cos\frac{2\pi}{r}\rceil, \lceil l\sin\frac{2\pi}{r}\rceil)$. These differences introduce two errors in diagonal stroke arrows: length errors and angle errors.

$$\text{Length error} = \left| l - \sqrt{\left\lceil l\cos\frac{2\pi}{r}\right\rceil^2 + \left\lceil l\sin\frac{2\pi}{r}\right\rceil^2} \right| \tag{4.3}$$

$$\text{Angle error} = \left| \frac{2\pi}{r} - \tan^{-1}\frac{\left\lceil l\sin\frac{2\pi}{r}\right\rceil}{\left\lceil l\cos\frac{2\pi}{r}\right\rceil} \right| \tag{4.4}$$

Figure 4.3 shows various combinations of the number of stroke directions and lengths with their length errors and angle errors. For the $s(x, 16, i)$'s, we show only $\frac{\pi}{8}$ stroke length and angle errors as the those of $\frac{\pi}{4}$ stroke are given in $s(x, 8, i)$'s. The figure suggests that there are only two choices whose length and angle errors are lowest at $s(7, 8, i)$ and $s(8, 12, i)$.

One used in this paper is $s(7, 8, i)$. We chose it because the approximate 7 length of diagonal arrows can be achieved by moving 5 pixel to the right and 5 to the north;

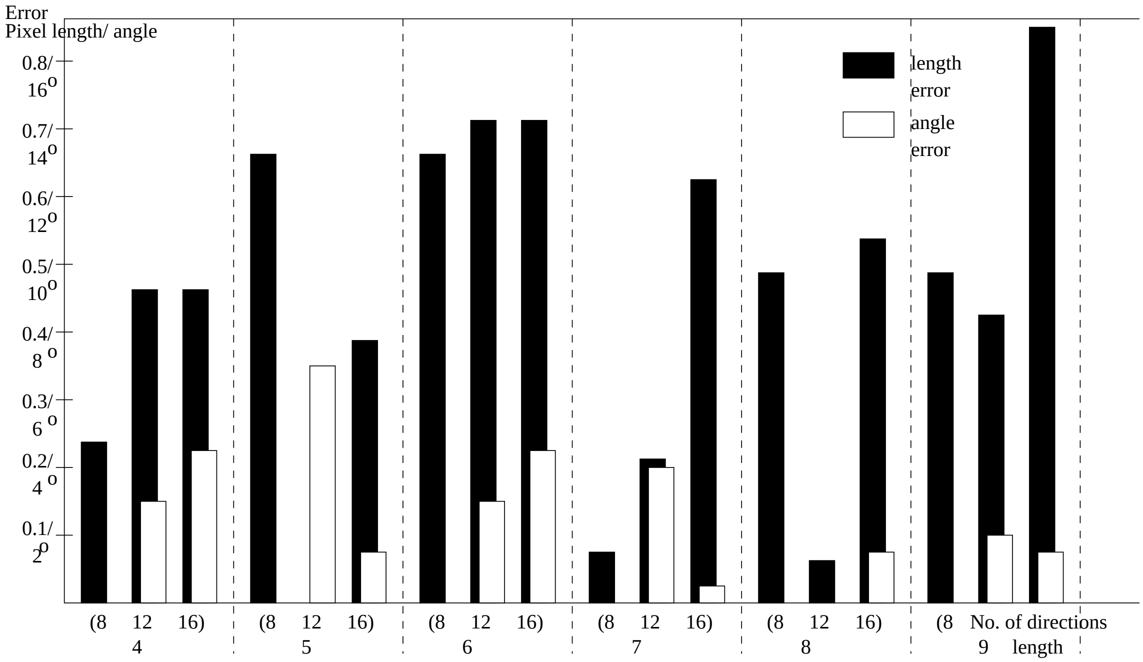


Figure 4.3: Error in representing stroke direction and length for various levels of direction quantization (8,12,16) and length quantization (4-9).

the exact distance is 7.071 as shown in Figure 4.1. An alternative choice is $s(8, 12, i)$. This gives the approximate length for the 1 o'clock direction can be obtained by moving 4 pixels to the right and 7 pixels to the north; the exact distance is 8.062. Although the length error is smaller than the previous case, it has the angle error. The exact angle between the 0 stroke and 1 stroke is 29.74° whereas the desired and intended one is 30°.

4.1.3 Linear type string

Next, in a linear type string, the type of an element value is an integer(discrete) or a real(continuous). The distance between element values $s_{1,i}$ and $s_{2,j}$ is:

$$d(s_{1,i}, s_{2,j}) = |s_{1,i} - s_{2,j}| \tag{4.5}$$

It is called a linear type because a measurement value, $s_{x,y}$, represents a scale strictly from small to large. For instance, a *Stroke Pressure Sequence String* or *SPSS* in short, shown in Figure 4.2 falls into this category.

Pressure information is regarded one of the most important features in *writer verification*. The pressure information is readily available in *on-line signature verification* whereas it is hard to extract in off-line writer verification. In ink pen based handwriting, however, pressure usually appears as a thickness of the stroke in character binary images and an *SPSS* is a sequence of stroke thickness although the pressure can also appear as different grey scale levels in grey scale images.

An *SPSS* is obtained after its *SDSS* is obtained. This is because there is no reason to measure the distance between *SPSS*'s if their *SDSS*'s do not match. Thus, the thickness is measured for each character directional stroke. As shown in Figure 4.19, vertical and horizontal strokes have 7 candidates as their stroke lengths are 7 pixel

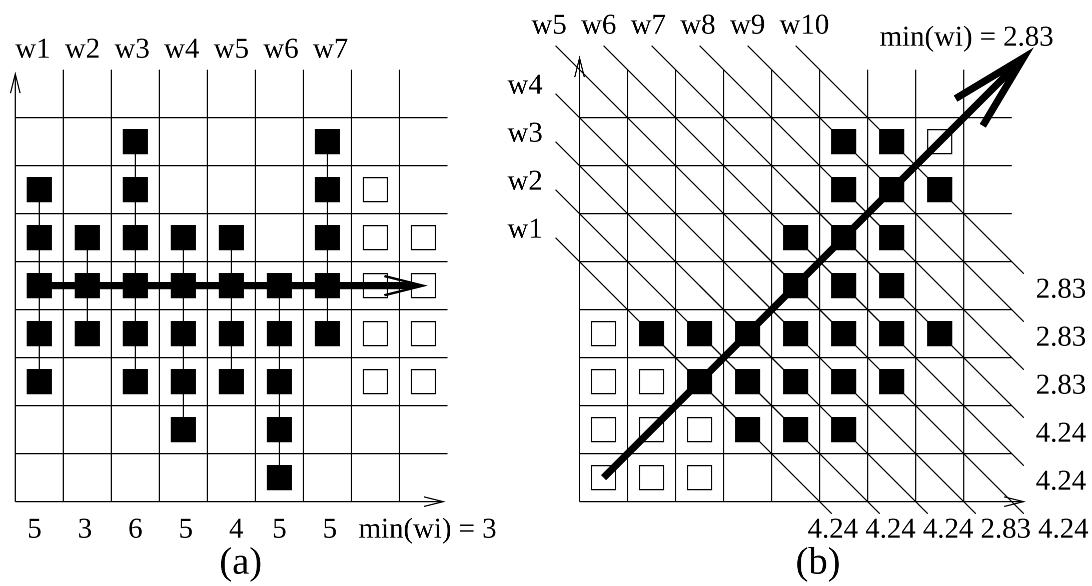


Figure 4.4: Stroke Width: (a) vertical and horizontal stroke width (b) diagonal stroke width.

long: $w_1 \sim w_7$. The width of these strokes is the minimum value of w_i's. For example of Figure 4.4 (a), the stroke width is $\min(w_i) = w_2 = 3$. Diagonal strokes have 10 candidates, $w_1 \sim w_{10}$, as illustrated in Figure 4.4 (b) and $\min(w_i) = w_4, w_8, w_9, w_{10} = 2.83$.

As with the *SDSS*, the *SPSS* is also a pseudo pressure information in off-line images. Suppose two strokes cross each other and the entire part of one stroke is completely covered by the other stroke or strokes. Then the detected width of the hidden stroke, $\min(w_i)$ is the length of the other stroke or more. This extreme case is, however, detectable and can be avoided by replacing it with the interpolated value of the previous and subsequent stroke widths. Now consider a looped letter *"l"* without a visible hole or a letter with a retrace as illustrated in Figure 4.5. The width is

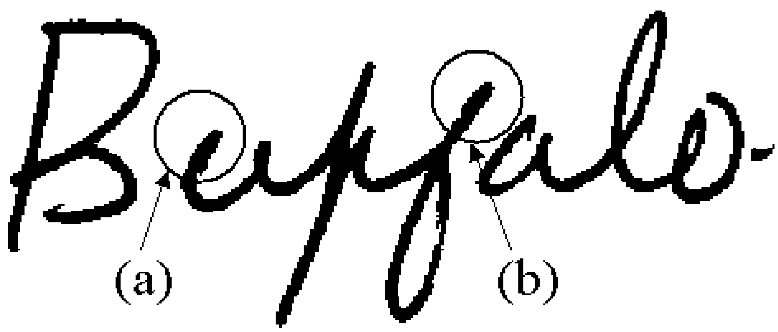


Figure 4.5: (a) a letter with a retrace, and (b) a looped letter without a visible hole.

almost doubled because of the retrace. In a grey scale image, the latter stroke width can be detected by a segmentation technique but there is no way to find the exact width of the bottom stroke. However, we use the width information even for the letter with a retrace as shown in the Figure 4.4 as this erroneous width information may be a unique peculiarity of one's handwriting style. In all, *SPSS* is only a pseudo pressure information.

In computing the distance between linear type strings, *Euclidean* distance is used after aligning them. Aligning process is necessary because *SPSS*'s are of disparate lengths and we would like to utilize the standard vector distance measures. By aligning two strings into the equal length l, the problem becomes the distance between two vectors of the same dimensionality with components of numerical values. Now standard vector norms such as *city block, Euclidean* or *Minkowski* distances can be used as distances between two aligned *SPSS's*. The detailed algorithm is given in the following section 4.3.1.

4.1.4 Cost-Matrix type string

Finally, the Levenshtein distance can be extended to cost-matrix type strings. A proper assignment of cost values for substitution between alphabets is used instead of a uniform penalty value 1 or 2. Levenshtein used 2 since the substitution between $d(s_{1,i}, s_{2,j})$ can be regarded as deleting $s_{1,i}$ from the i'th position of S_1 string and inserting $s_{2,j}$ value in that position. Different substitution cost values can be assigned to different alphabets. As long as the size of the alphabet is finite, the cost-matrix can be built.

Wagner and Fischer [108] used the following notation: $\gamma(a \longrightarrow b)$ meaning the cost of changing the alphabet a to b. If $a = \Lambda$ where Λ is a null string, it is the cost of inserting b to the string. If $b = \Lambda$, it is the cost of deleting a from the string. If none of them is Λ, it is the substitution cost. Thus the size of the cost-matrix is $(|\Sigma| + 1) \times (|\Sigma| + 1)$ where Σ is a finite alphabet; the null value Λ must be included. The section 4.2.2 discusses how our algorithm differs from the edit distance with cost matrix.

4.2 Stroke Direction Sequence String Matching

The approximate string matching for *SDSS*'s is that given two *SDSS*'s, find the minimum edit distance between them by allowing *turn* and *indels*. In this section, we describe the algorithm and compare it with the edit distance with cost matrix. The algorithm for the newly defined edit distance uses the dynamic programming method of computing the edit distance between two strings [85].

4.2.1 Algorithm

Consider two "A" letters in Figure 4.2. As discussed earlier, the type of stroke direction element, $s_{x,y}$ is angular and the distance between two stroke direction elements, $d(s_{1,i}, s_{2,j})$ is given in equation (4.2) as a turning distance. This operation, *turn* is a very important difference between the conventional nominal type strings and angular type strings. While the former allows the substitution with the uniform cost of 1 or 2, the later allows the turn with various cost values.

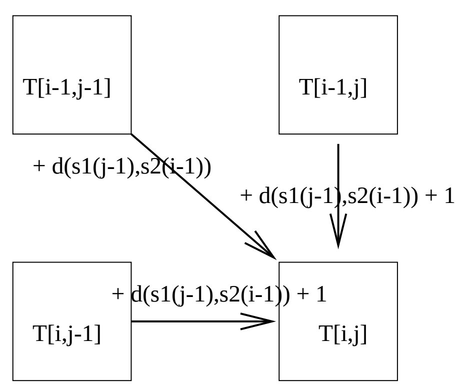
The costs for insertion and deletion are defined in the following equation (4.6).

$$T[i,j] = \min \begin{cases} T[i-1,j-1] + d(s_{1,i-1}, s_{2,j-1}) & \Leftrightarrow \text{turn} \\ T[i-1,j] + 1 + d(s_{1,i-1}, s_{2,j-1}) & \Leftrightarrow s_{1,i-1} \text{ is missing} \\ T[i,j-1] + 1 + d(s_{1,i-1}, s_{2,j-1}) & \Leftrightarrow s_{2,j-1} \text{ is missing} \end{cases} \tag{4.6}$$

Figure 4.6 illustrates the distance computing table for the first parts of sample letters and shows how each entry is computed. Let S_1 lie on the top of the table and S_2

	s1	2	1	2	1	0	7	6	7	6
s2	00	02	03	05	06	08	11	15	18	22
1	02	01	02	04	05	07	10	14	17	21
1	04	03	01	03	04	06	09	13	16	20
1	06	05	02	02	03	05	08	12	15	19
1	08	07	03	03	02	04	07	11	14	18
1	10	09	04	04	03	03	06	10	13	17
5	14	13	09	07	08	06	05	07	10	12
5	18	17	14	11	11	10	08	06	09	11
6	23	22	18	16	14	13	10	07	07	08
5	27	26	23	20	19	17	13	09	09	08
5	31	30	28	24	24	21	16	11	11	10

(a)



(b)

Figure 4.6: (a) Computing edit distance table (b) cell computation.

in the left side of the table. The first row is initialized by inserting a stroke, $s_{1,0}$ to the head of the string S_2 as shown in Algorithm 3 line 4 and 5. The left-most column is initialized by inserting a stroke, $s_{2,0}$ to the head of the string S_1 as shown in Algorithm 3 line 6 and 7. Now the rest of the table entries $T[i,j]$'s where $i = 2 \cdots n_1$ and $j = 2 \cdots n_2$ are computed by taking the minimum value of three values computed in equation 4.6 in line 8 to 10. The distance between two *SDSS*'s is achieved from the table in Figure 4.6 and it is 10.

Algorithm 3 Edit Distance (S_1, S_2)

1 begin

2 $\quad n_1 \leftarrow \mathit{length}\ (S_1),\ n_2 \leftarrow \mathit{length}\ (S_2)$

3 $\quad T[0,0] \leftarrow 0$

4 $\quad \mathit{for}\ i = 1\ \mathit{to}\ n_1$

5 $\qquad T[0,i] = T[0,i-1] + d(s_{1,i-1}, s_{2,0})$

6 $\quad \mathit{for}\ i = 1\ \mathit{to}\ n_2$

7 $\qquad T[i,0] = T[i-1,0] + d(s_{1,0}, s_{2,i-1})$

8 $\quad \mathit{for}\ i = 1\ \mathit{to}\ n_2$

9 $\qquad \mathit{for}\ i = 1\ \mathit{to}\ n_1$

10 $\qquad\quad T[i,j] = \min(T[i-1,j-1] + d(s_{1,i-1}, s_{2,j-1}), T[i-1,j] + 1 + d(s_{1,i-1}, s_{2,j-1}),$
$T[i,j-1] + 1 + d(s_{1,i-1}, s_{2,j-1}))$

11 $\quad \mathit{return}(T[n_2, n_1])$

12 end

The computational time complexity for the algorithm 3 is $O(n_1 n_2)$ and the space requires only $O(n_1)$ because only the entries in the previous column need be stored.

To illustrate, consider Figure 4.7. We would like D("1", *"1"*) to be smaller than D("1", "-"). The equal weighted Levenshtein edit distance is incapable of differentiating these. The D_T and D_L are the distance matrices of the newly defined edit

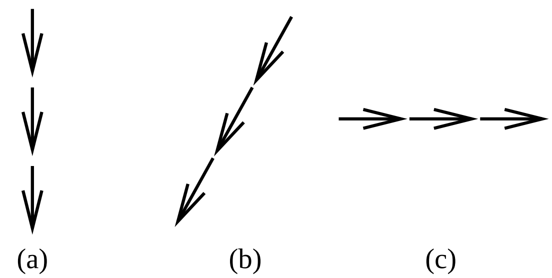


Figure 4.7: Sample Characters (a) "1", (b) skewed "*1*" (c) "-".

distance with turning concept and the equal weighted Levenshtein distance, respectively. The row and column indices are characters in Figure 4.7 in the same order: (a),(b),(c).

$$D_T = \begin{pmatrix} 0 & 3 & 6 \\ 3 & 0 & 9 \\ 6 & 9 & 0 \end{pmatrix}, D_L = \begin{pmatrix} 0 & 3 & 3 \\ 3 & 0 & 3 \\ 3 & 3 & 0 \end{pmatrix}$$

This inadequacy of the equal weighted Levenshtein edit distance can be improved by simply defining different costs for each element of strings so called a cost-matrix. The next section discusses the difference between the newly proposed edit distance and Levenshtein edit distance with cost matrix.

4.2.2 Comparison with cost-matrix string version

The conventional edit distance table used in spelling correction is computed by the following equation:

$$T[i,j] \quad = \quad \min \begin{cases} T[i-1,j-1] + \gamma(s_{1,i-1} \longrightarrow s_{2,j-1}) & \Leftrightarrow \quad \text{substitution} \\ T[i-1,j] + \gamma(\Lambda \longrightarrow s_{2,j-1}) & \Leftrightarrow \quad s_{1,i-1} \text{ is missing} \\ T[i,j-1] + \gamma(s_{1,i-1} \longrightarrow \Lambda) & \Leftrightarrow \quad s_{2,j-1} \text{ is missing} \end{cases} \tag{4.7}$$

The substitution part is the same in both equations (4.6) and (4.7) and we use the turn concept as a proper assignment of cost values to the substitution part of the

Levenshtein matrix. The difference between the previous edit distance with a cost matrix and the newly defined edit distance is the *indel* part.

In equation (4.7), the insertion cost is added by pre-defined costs between the null element Λ and the corresponding alphabets in the other string. For example, consider our example from Figure 4.2:

$$\begin{array}{rcccccccccc} \mathrm{SDSS}(S_1) = & \uparrow & \nearrow & \uparrow & \nearrow & \rightarrow & \searrow & \downarrow & \Lambda & \searrow & \downarrow \\ \mathrm{SDSS}(S_2) = & \nearrow & \nearrow & \nearrow & \nearrow & \nearrow & \swarrow & \swarrow & \downarrow & \swarrow & \swarrow \end{array} \tag{4.8}$$

The penalty for the insertion in S_1 is given by the predefined $\gamma(\Lambda \longrightarrow \downarrow)$. Λ is an arbitrary element that can be any one of eight directional arrows. Despite this fact, using the predefined penalty value is quite absurd.

We have a different approach that uses the local context. Instead of using the predefined penalty value, we first change $\Lambda \longrightarrow s_{1,i-1}$ whose penalty value is 1 and then add the turn distance between them, $d(s_{1,i-1}, s_{2,j-1})$. In the example, we first assign $\Lambda \longrightarrow \downarrow = 1$ that is the left stroke of the Λ, $s_{1,i-1}$ in $SDSS(S_1)$ not $s_{2,j-1}$ and then add $d(\downarrow, \downarrow) = 0$. The penalty value is not uniform but varies depending on the stroke in the left side of Λ. We use only one side of local context instead of the average value between $s_{1,i-1}$ and $s_{1,i}$. This is because the average cannot be defined for directions, e.g. $\mathrm{ave}(\uparrow, \downarrow) = \leftarrow$ or $\rightarrow$?

Marzal and Vidal showed the normalized edit distance with a cost-matrix and applied it to recognize the off-line digit images using the *Freeman style* contour chain-code strings [66]. *M. Parizeau et al* used genetic algorithms to optimize the cost matrix to recognize the on-line digits [77]. Table 4.1 shows the comparison of various cost-matrices used on angular type strings. Note that Marzal's matrix is not symmetric. In Parizeau's matrix, $\gamma(\swarrow, \nwarrow) > \gamma(\swarrow, \uparrow)$. In all, the idea of *turn* and utilizing the local context is more suitable for the stroke direction sequence strings of character images.

Table 4.1: Comparison of various cost-matrices

	Marzal's cost-matrix [66]	Parizeau's cost-matrix [77]	Turning distance
$\gamma(\swarrow,\uparrow)$	9.26	9.7	3
$\gamma(\swarrow,\nwarrow)$	∞	11.0	2
$\gamma(\swarrow,\leftarrow)$	6.43	0.1	1
$\gamma(\swarrow,\swarrow)$	0	0	0
$\gamma(\swarrow,\downarrow)$	6.27	0.1	1
$\gamma(\downarrow,\swarrow)$	5.76	0.1	1
$\gamma(\downarrow,\Lambda)$	3.34	1.9	Varies

4.3 Applications

The approximate string matching technique has found its applications in pattern classification [36], and it utilizes the nearest-neighbor algorithm for the classification as shown in Figure 4.8. We consider three important applications: *writer verification* [88, 6, 80, 25, 21], *on-line* [82, 81, 72, 102] and *off-line character recognitions* [82, 99].

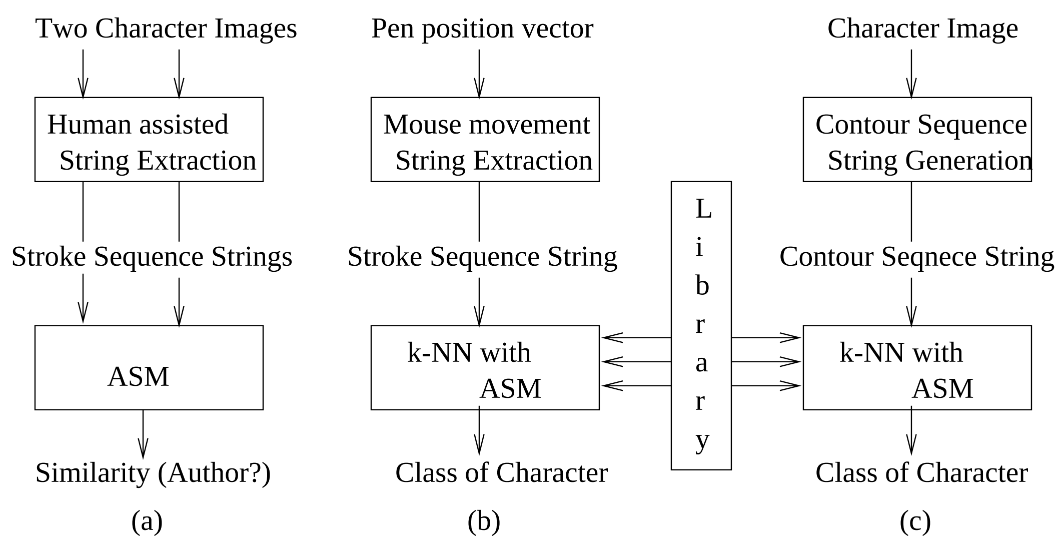


Figure 4.8: Applications of the string distance measure: (a) Writer Verification (b) On-line Recognition (c) Off-line Recognition.

4.3.1 Writer Verification

In Figure 4.8 (a) *writer verification*, one would like to find out whether two characters were written by the same author based on their similarity. Among many features being currently used by the handwriting analysis practitioners or the document examiners, the *form* or *shape* is an important one for characterizing individual handwriting as it is quite consistent with most writers in normal undisguised handwriting [6]. The form can be described by a sequence of directional strokes and the similarity measure can be a key feature in handwriting analysis. Among many features, we consider two types of stroke sequence strings of a character as character level image signatures for writer verification: stroke direction sequence string(*SDSS*) and stroke pressure sequence strings(*SPSS*).

To extract stroke directional/pressure sequence strings for both letters to be compared, a trained human, well known as a document examiner, is involved during the examination. Albeit many methods have been proposed in the literature for accomplishing the task automatically [100, 88], it is impossible to extract the exact stroke sequence of the author of the letter. Hence, we develop a semi-automatic writer verification system as it is fairly easy for a human, especially for document examiners, to extract the pseudo on-line information. A stroke direction sequence string extractor allows an examiner to load a digitally scanned questioned letter or word image and extract the author's pseudo on-line information while he or she is tracing the strokes on screen. Hence, an *SDSS* is a pseudo on-line stroke directional sequence string. A pen based input device is highly recommended rather than a mouse device.

Next, computing the distance between *SPSS*'s first requires the aligning process because two *SPSS*'s may be of disparate lengths. To align them, we utilize Algorithm 3 as a stepping stone. The algorithm constructs the computing edit distance table as shown in Figure 4.6 and then the editing sequence is obtained. New *SDSS*'s with the exactly same lengths are derived according to the editing sequence as expressed in equation (4.8). They are of same lengths by inserting Λ in the respected places. One only needs to consider insertion since deletion in one string means insertion in the other string. Once the *SDSS*'s with Λ symbols are obtained, one can replace the directional arrow elements with respected magnitude values. Two *SPSS*'s in Figure 4.2 are aligned accordingly and they are denoted as SPSS_l's.

$$\begin{array}{lcccccccccc} \mathrm{SPSS}_l(p_1) = & 3.0 & 3.0 & 3.0 & 1.0 & 3.0 & 3.0 & 2.0 & 2.5 & 3.0 & 2.0 \\ \mathrm{SPSS}_l(p_2) = & 3.0 & 3.0 & 1.0 & 1.0 & 2.0 & 2.0 & 2.0 & 3.0 & 2.0 & 2.0 \end{array} \tag{4.9}$$

Note that we use the notation $p_{x,y}$ instead of $s_{x,y}$ to distinguish the pressure of a stroke from its direction. The value in the inserted position, Λ, is the average value between the pressures of the adjacent strokes. By aligning two strings into the equal length l, the problem becomes the distance between two vectors of the same dimensionality with components of numerical values. Now standard vector norms such as *city block, Euclidean* or *Minkowski* distances can be used as distances between two aligned *SPSS's*. Thus, the distance between *SPSS*'s is:

$$D[\mathrm{SPSS}_l(p_1), \mathrm{SPSS}_l(p_2)] \quad = \quad \frac{\sqrt{\sum_{i=1}^{l}(p_{1,i} - p_{2,i})^2}}{l} \text{ where } l = \max(n_1, n_2) \tag{4.10}$$

Demonstration

45 people provided their handwriting exemplars. All writers used the same writing materials: pen, papers and table. Each writer provided three exemplars of a word list that contains all capital letter in the beginning of the word and all small letters in all three positions: beginning, middle and terminal positions of a word. They are {April, Bob, California, December, English, February, Greg, Halloween, Iraq, June, Kentucky, Los Angeles, Markov, November, October, Pennsylvania, Queen, Raj, States, Texas, United, Virginia, What, Xray, York, Zorro, alumni, boy, come, date, enjoy, false, great, have, interest, jazz, keep, leave, millennium, now, of, picnic, question, run, six, time, unique, video, where, xenophobia, you, zero}. Table 4.2

Table 4.2: Count of letters

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

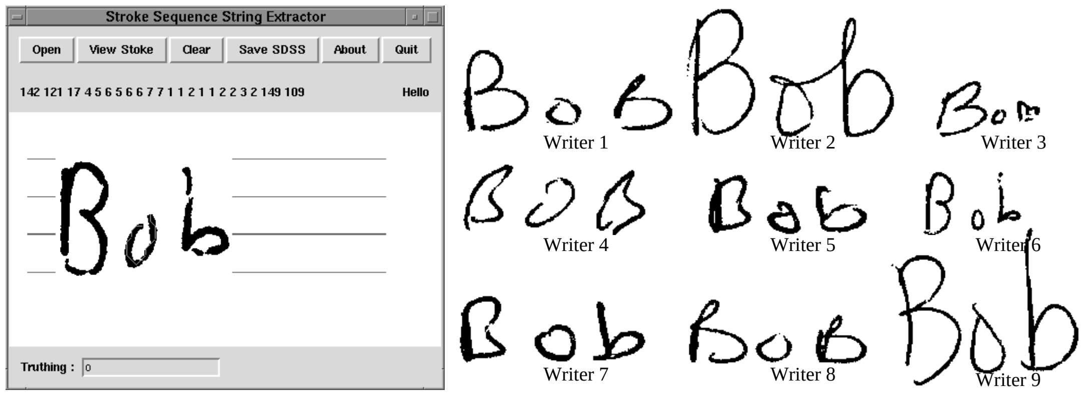
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
17	5	4	1	31	1	3	3	19	1	2	10	5	18	18	2	1	16	6	8	9	4	1	1	1	1
4	1	1	1	9	1	1	1	1	1	1	1	1	4	2	1	1	3	3	3	1	1	1	1	5	1

shows counts of each letter in each position. The second row is the appearance count of the capital letter A through Z in the beginning of the word. The fourth to sixth rows are the appearance counts of the small letter a through z in the beginning, middle and terminal positions, respectively.

The test was designed with 10 exemplars. 10 writers are selected randomly and we chose one word from each writer except for one writer with 2 exemplars since we need one query word. 10 such sample tests [2] are prepared and the system corrected 9 out of 10 whereas expert and non-expert human beings corrected 9 and 7 on average, respectively. One such sample test is given in Figure 4.9. The word image shown in the program window is the query handwriting and the right-side 9 words are known handwritings. Note that the query image is slightly scaled up. Since writers 2, 3 and 9 have two contiguous strokes for the letter *"B"* while the query writer has only 1, they are eliminated immediately from further investigation. Now one can load each image on the *SDSS* extractor. After extracting the *SDSS*'s, the edit distances are computed.

Let $D(W_q, W_x) = \sum_{i=\text{"B"}}^{\text{"b"}} D(\mathrm{SDSS}(S_q(i)), \mathrm{SDSS}(S_{W_x}(i)))$. Note that there are three contiguous strokes: $S_q(\text{"B"})$, $S_q(\text{"o"})$ and $S_q(\text{"b"})$. The authorship of W_q is determined by that of W_x such that $W_x = \arg\min_{i=1\cdots n} D(W_q, W_i)$. The results of the edit distances are $D(W_q, W_1) = 175$, $D(W_q, W_2) = \times$, $D(W_q, W_2) = \times$, $D(W_q, W_4) = 197$, $D(W_q, W_5) = 101$, $D(W_q, W_6) = 53$, $D(W_q, W_7) = 98$, $D(W_q, W_8) = 131$, $D(W_q, W_9) = \times$. We have $W_x = W_6$. Therefore, the author of the query handwriting is more likely to be the writer 6 than other given samples.

[2] "http://www.cedar.buffalo.edu/NIJ"



$q(B)$ = (↓↓↓↓↓↓↓↓↓↓↓↓↓↑↖↗↑↑↑↑↑↗↑↑↗↑↗→↗↘↘↘↘↓↙↓↙↙↙↖↘↓↓↘↘↘↓↓↙↓↙↙←←←↑↑)
$q(o)$ = (↙↙↓↓↓↓→→↗↑↗↑↑↑↑)
$q(b)$ = (↓↓↓↘↓↓↓↓↓↙↓↓↙↗↑↗→↗→→↘↘↙↓↙←←←)

$w_1(B)$ = (↓↓↓↘↓↓↘↓↓↘↓↑↑↑↖↑↑↑↑↖↖↑↑↑↑↗↑↑↑↑↗↗↑↑→→→ ↘→↘↘↘↓↓↓↓↓↓↙←↙←↙↙←↗→↗→→→→→↘↘→↘→↘↘↓↘↓↙↙↙↙←↙←↙←←↙←←←)
$w_1(o)$ = (←↓↙↙↓↓→↘→↘→↗→→↑↑↑↗↑↖←↖←↖←←)
$w_1(b)$ = (↑↑↗↑↗↑↑↗↗↗↗↗↗↗↘↙↙↓↙←↙↙↙↘→→→→→→→→→↘↘↘↘↓↓↙↙↙←←←↙←←←←←←←)
$w_2(B)$ = (↓↓↙↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↙↓↓↓↓↓↘↓↓↙↓↘↓↓)(↑↑↗↑↑↑↗↑↗↑↑↑↗↗↑↑↗→→↗↗→→→→→↘↘↘↓↓↓↓↙↓↙↓↙ ↓↙↙↙↙←↙↙←←↙↙↗↗↗→↗↗→→→↘→↘↘↘↘↘↘↘↓↓↓↓↙↓↙↙↓↓↙↙↙↙↙←←↙←←← ←↖↖)
$w_2(o)$ = (↓↓↙↓↓↓↓↙↓↙↓↓↘↓↘→↘→↗→↗→↗↗↑↗↗↑↑↖↑↖↖↑↖↖↖←↖↖↖←←←←↘↘↘→↗→→→→→↗ →→)
$w_2(b)$ = (↗↗↗↗↗↗↗↗↗↑↑↗↑↗↑↖↑↖←↙↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↘↓↓↓↙↘↓↓↓↓↓↘→→→↗→↗→↗↗↗↑↗↗ ↗↗↗↑↑↖↖↖↑↑↖↖↖↖←↖↙←↙↙↙↙↙↓↙↙↓)
$w_3(B)$ = (↙↙↓↓↙↓↓↙↓↙↓)(↗↗↗→↗→↗↗→↗→→↘↙↓↙↙↙↙↙←↙↙↗→→→→→↘→↓↓↙↓↙←↙←←←↙ ↙←←←)
$w_3(o)$ = (↙↙↙↓↘→→↗↑↑↖)
$w_3(b)$ = (↑↑↑↑↑→↘↙↘↑→→→↘↙↓↙↙←)
$w_4(B)$ = (↑↖↑↗↑↑↑↑↗↑↑↗↗↑↗↗↗↗→↗↗→↗→→→↓↙↓↙↙↙←↙↓↙↙↓↘↘↓↘↘↓↙↙←←←←←←)
$w_4(o)$ = (↙↙↙↙↙↙↓↘→→→→↗→↗↗→↗↑→↗↗↗↗↑↑↑↖↑↖←↖←↙←↙↙↙↓↑)
$w_4(b)$ = (↙↙←←↑↑↑↗↑↑↗↗↑↗↑↗↗↗↗↗↗↗↗→↓↙↙↙↓↓↓↘↓↘↘→↓↓↓↙←↙↙↙←↙↙)
$w_5(B)$ = (↓↓↘↓↓↓↙↓↓↓↓↓↓↓←↑↖↑↑↗↑↗↑↗↑↑↑↗↗↗↗→→→→↓↓↓↙↓↙↓↓↘↓↘↘↓↙↙←←↙←←←↙)
$w_5(o)$ = (←←↙←←↙↙↘↘↘↘→→→→↗↑↑↖↖↑↑↖↖←←↓)
$w_5(b)$ = (←↖←↙↓↓↙↓↓↘↓↓→→↗↗↗→↗→→→→→↘↘↓↓↙↙↙←←←←←←←←↖↖)
$w_6(B)$ = (↓↘↓↓↓↙↓↓↓↓↓↓↓↑↗↖↑↑↗↑↑↑↑↑↑↑↑↗↑↖↑→→↗↘→↘↓↘↘↓↙↓↙↓↙←↖←→↘↘↓↘↓↓↙↓↓↙←↙←←←)
$w_6(o)$ = (↙↙↓↙↘↓→→↗↑↑↑↖)
$w_6(b)$ = (↓↙↓↓↘↓↓↓↙↓↓↓↓↓↗→↗↗→→↘↙←←)
$w_7(B)$ = (↓↘↓↓↓↓↘↓↓↓↓↓↓↙↑↖↑↑↑↑↑↗↑↑↑↑↗↑↗↑↗↗↗→→→→↘↓↓↓↓↙↓↙↓↙↓↓↘↘↘↘↘↙←↙←←←←←)
$w_7(o)$ = (↙←↙↓↓↘↘↘↘↘→→→→→↗↗↗↑↖↑↖↖↑↖←↖←↙←↙)
$w_7(b)$ = (↓↘↓↓↓↘↓↓↘↓↓↘↓↓↓↗↑↗↗↗↗↗→→→↘↘↓↙↙↙↙←↙←↙←←←←←←)
$w_8(B)$ = (↓↓↙↓↓↓↓↓↓↘↖↑↑↗↑↑↗↑↑↑↑↑↑↗↑↗↗↑↑↗→→↘↙↓↙↙↙↙↙←↙↙→↘→→↘→→→→↘→→↘↘↘↘ ↓↓↙↓↙←←↙←←←←↖↑)
$w_8(o)$ = (↙←↙↙↙↓↙↓↘→↘→→↗→↗↑↗↗↑↖↑↖)
$w_8(b)$ = (↓↓↙↓↓↘↓↓↗↑↑↗↑↑↗↑↗↑↗↗↗↓↙↓↙↙←→→→→→→↘→↘↘↘↘↓↓↙↓←←←←←←←←↙←←↖←)
$w_9(B)$ = (↘↘↘↓↘↓↓↓↘↘↓↓↓↓↘↓↓↓↘↓↓↙↓↓↓↓↓↙↓↙)(↑↗↑↗↗↗↗↗→→→→↘→↓↘↓↓↓↓↙↙↓↙↙↙↙↙↙↙← ↙←↙←→↗→→→→→→→→→↘↘↘→↘↘↓↘↘↘↓↙↓↙↓↓↙↓↙↓↙↙←↙↙←↙←↙←←)
$w_9(o)$ = (↙↓↓↙↓↓↓↓↓↓↓↓↓↘↓↘→→→↗↗↗→↑↗↑↖↑↑↖↑↑↖↖↖↖↖↖↖↖↑↖↖↑)
$w_9(b)$ = (↙↓↙↓↓↓↓↓↓↓↓↘↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↘↓↓↓↓↓↓↓↑↑↑↑↗↑↑↗↑↑↗↑↗↗↑↗↗↗↗→→↘↘↓↘↘↓↘↓↓↓↓↙↓↓ ↓↙↙↙↙←←↙←↙←←←←↖↖↖)

Figure 4.9: GUI for *SDSS* extractor, Sample writings and their *SDSS*'s.

Discussion

The purpose of the previous experiment is to demonstrate the procedure of comparing two handwriting sequences. The proposed method is not the solution to the *writer verification* dealing with real cases, but only a part of the entire process. Since the detailed *writer verification* is under preparation in another paper, we give a sketchy outline of the experiment. The full and detailed experimental report and its integration with multiple features can be found in [21, 25] and the complete procedure to assess the authorship confidence of arbitrary handwritten items can be found in [17]. *SDSS* is the most effective one among features that we consider in [21] and gives very low 4.9% *type I error* and 19.5% *type II error* where *type I error* occurs when the same author's handwritings are identified as different writers and *type II error* occurs when the two different writers' handwritings are identified as the same writer. When integrated with many other features, we achieve 97% overall correctness performance.

A problem that arises with the proposed method is that of human involvement during the feature extraction phase. It is an open problem to make the writing sequence extraction completely automatic. Notwithstanding, the proposed semi-automatic system is of great interest because it is less subjective. Interaction with questioned document is generally personal and subjective. The proposed measure can reduce the subjectivity significantly.

4.3.2 On-line Character Recognition

The problem of *on-line character recognition* can be formalized by defining a distance between characters and finding the nearest neighbor in the reference set. We obtain stroke directional sequence strings from the movements of a mouse or a pen-based device. To do so requires that we convert the mouse position vector that is a non-uniform length direction string into the *SDSS* that is a string of uniform length

(7-pixel long) directions.

$$
\begin{aligned}
\text{mouse position} &= \{(0,0),(0,14),(3,15),(5,19),(12,19)\} \\
\overline{(0,0),(0,14)} &= \rightarrow\ \rightarrow \qquad (4.11) \\
\overline{(0,14),(3,15)} + \overline{(3,15),(5,19)} &= \searrow \qquad (4.12) \\
\overline{(5,19),(12,19)} &= \downarrow \\
\text{SDSS} &= \rightarrow\ \rightarrow\ \searrow\ \downarrow
\end{aligned}
$$

When the mouse moves fast, it creates a few of mouse positions and more strokes are filled in between these positions. Such case is depicted in equation (4.11). When the mouse moves slow, on the other hands, it creates many tiny strokes. These tiny strokes are merged into fewer 7-pixel long strokes as in Equation (4.12). In all, geometrically best fitting strokes are selected to create an *SDSS* from the mouse movement vector.

To recognize an unknown on-line character, one measures the edit distances between the input string and reference strings [81, 77, 14, 15]. Next, the class of an input string is determined by votes on k-nearest neighbors. As a stroke sequence signifies the shape of the individual letters, a letter *"a"* is distinguished from a letter *"b"* by its different stroke sequences.

In this experiment, 20 subjects provided each numeral five times in their own natural handwriting. When performing the nearest neighbor classification, each numeral is correctly classified without an error. When the unnatural handwriting is given, the classifier did not recognize many unnatural handwritings. Although the two simple string manipulation techniques diminish the effect of unnaturally written handwritings, much more complex string manipulation operations are required to handle unnaturally written handwriting.

Desirable Invariance Properties

In seeking to achieve an optimal representation for a particular pattern classification, we confront the problem of *invariance* such as orientation, size, rate, deformation and discrete symmetry in the classification problems in scene analysis [36]. The desirable invariance properties in the on-line handwritten character recognition are noise [72], distortions [79], baseline drift [8], slant [9], script size [72], break [111], etc. Nouboud and Plamondon used data smoothing, signal filtering, dehooking and break corrections [72, 81].

One of the most important desirable invariant properties in on-line handwriting recognition is writing speed invariant property [99, 81]. It is a necessary property because a spatially same character can be written temporally differently. Thus, a recognizer trained with temporal information fails to recognize handwritings in different writing speed unless it is invariant to writing speed. This problem of writing speed invariant on-line handwriting recognition is usually tackled using chain-code [81] or stroke direction sequence strings [14, 15]. We present a procedure that convert a non-uniform writing speed and acceleration data into the uniform speed and acceleration data.

We assume that the on-line handwritten character input signals are normalized using the conventional techniques such as position, size normalizations, deskewing and base-line drift correction, etc. In this paper, we focus on the writing speed and writing sequence invariance where most previously developed on-line handwritten character recognition systems fail. First, the *stroke sequence string* representation of given input signals gives the writing speed invariance to the recognizer. The detailed stroke sequence string extraction algorithms can be found in our earlier works [14, 15]. Next, this *stroke sequence string* representation paves a way for the writing sequence invariance property by using several string manipulation operators, e.g., *concatenation, reverse* and *ring* operations. The idea of concatenation was also considered by Plamondon and Nouboud for the no constraint on the writing and the system that can recognize any character defined by the user [81]. We introduce more string manipulation techniques to further reduce the constraints in writing.

Writing Speed Invariance

In this section, we show an example how a spatially same character can be written differently in terms of writing speed and acceleration. Next, we introduce a *stroke direction sequence string* [14, 15] representation of on-line input patterns and finally show that it is a writing speed normalized representation. Several subjects were asked

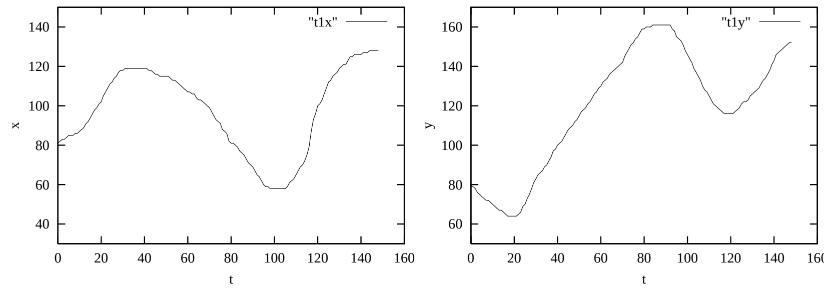
(a) Original digit "2" image

(b) reproduced on-line "2"

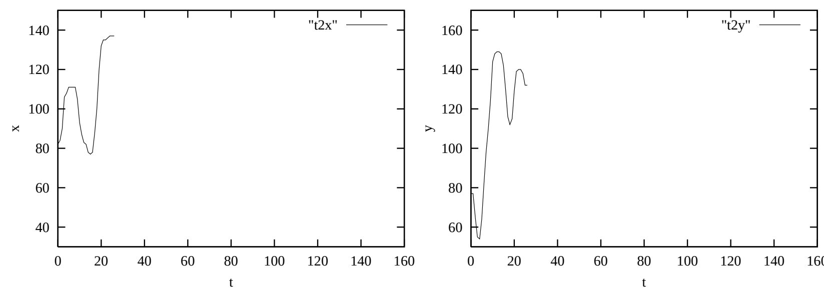
Figure 4.10: Sample digit image "2"

to copy a digit *"2"* image as shown in Figure 4.10 (a) to produce the on-line data as in Figure 4.10 (b) and everyone writes differently in terms of writing speed and time. Figure 4.11 illustrates the case of various temporal writing sequence inputs for the spatially same character shape. Some write fast and some write slow as in Figure 4.11 (a) and (b), respectively. Figure 4.11 (c) indicates the x-y graphs where a subject writes *"2"* in non-uniform writing speed and non-uniform acceleration. Albeit their spatial patterns are exactly the same, their temporal data are different due to the different velocity, $v(t) = [(\frac{dx(t)}{dt})^2 + (\frac{dy(t)}{dt})^2]^{1/2}$ and different acceleration $a(t) = \frac{dv(t)}{dt}$. In this section, we present how to normalize the different temporal data into uniform writing speed data using the *stroke sequence stroke string.*

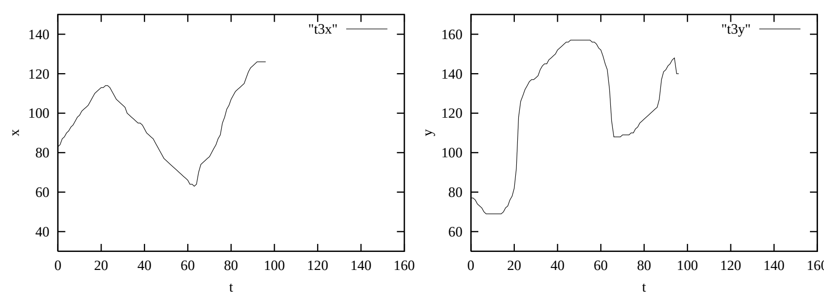
Now *SDSS* is generated based on not temporal data but sequential and spatial data. Thus all three cases in Figure 4.11 (c), (d) and (e) have the same or similar



(a) slow writing X and Y position graphs



(b) fast writing X and Y position graphs

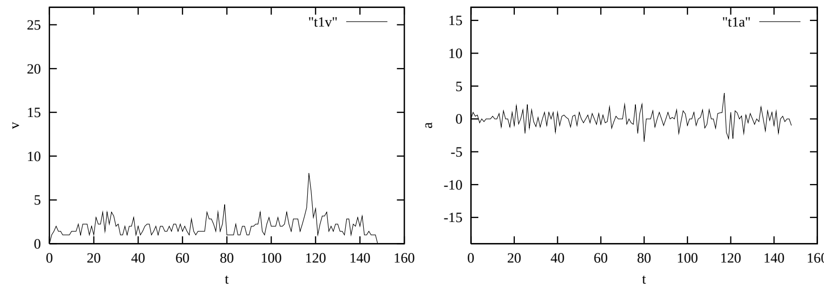


(c) non-uniform writing speed X and Y position graphs

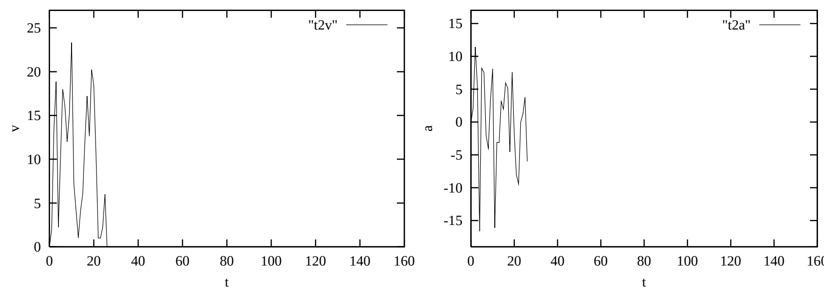
Figure 4.11: Various on-line XY-graphs for spatially same character “2”

SDSS as follows:

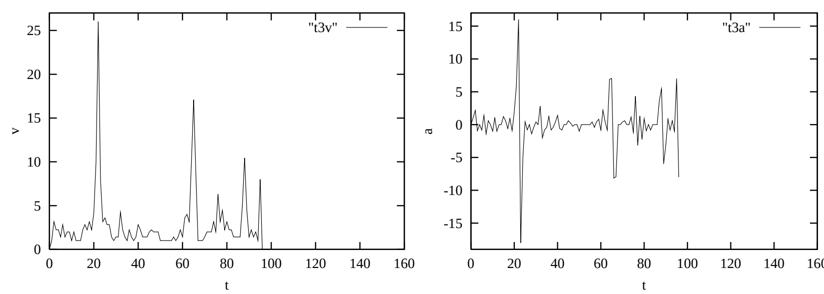
{↗↗↗→↘↘↓↘↓↓↓↓↙↓↙↙↙↓↙↙←←←↖↑↑↑↗↗↗→→→→↘↘↘↘↓↓}. An SDSS can be plotted back into the X-Y position graphs. The length of the string is the time spent to draw the character. All three cases in Figure 4.11 have the same normalized graphs as given in Figure 4.13 (a). Its velocity and acceleration are uniform 7 and 0, respectively.



(a) $v(t)$ and $a(t)$ for Figure 4.11 (a)



(b) $v(t)$ and $a(t)$ for Figure 4.11 (b)

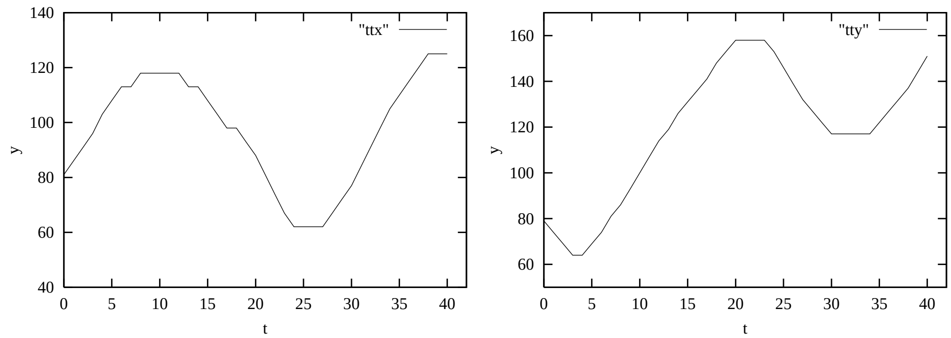


(c) $v(t)$ and $a(t)$ for Figure 4.11 (c)

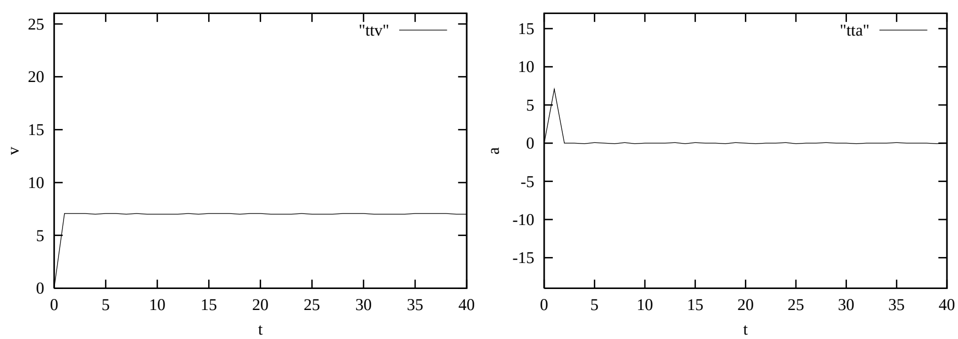
Figure 4.12: Velocity and acceleration graphs for graphs in Figure 4.11

Writing Sequence Invariance

Once a character is represented in (a) string(s), several string manipulation operators, e.g., *concatenation, reverse* and *ring* operations allow to handle the problem of writing sequence invariance. Although a break correction for Chinese characters by a stroke merging technique exists in the literature [111], many on-line handwritten character recognizers would not recognize a character if it were written in different order of drawing or there exists a break in the middle as shown in Figure 4.14.



(a) Writing Speed Normalized X and Y position graphs



(b) Writing Speed Normalized velocity and acceleration graphs

Figure 4.13: Normalized Temporal Writing Sequences for Figure 4.11 character "2"

String Concatenate and reverse Manipulation

This problem of awkward handwriting can be diminished by string operations: *concat* and *reverse*. *Concat* operation will concatenate two strings whose ends points are near. *Reverse* operation will reverse the order of the elements as well as complement each string element. One or several breaks may occur in drawing a single line, curve or circle. This must be considered as a single string rather than multiple strings. To obviate this inadequacy, the *concat* operation is necessary. The *reverse* operation is also needed because people may write a letter in the reverse order.

This section shows the step by step procedure with an example in Figure 4.15. Figure 4.15 (a) shows a handwritten character "X" in a very unnatural way of $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4$. This is one example where the most previous on-line character

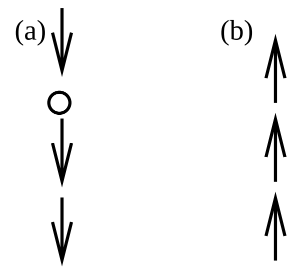


Figure 4.14: Sample Characters “1” (a) a break in the middle (b) written backward.

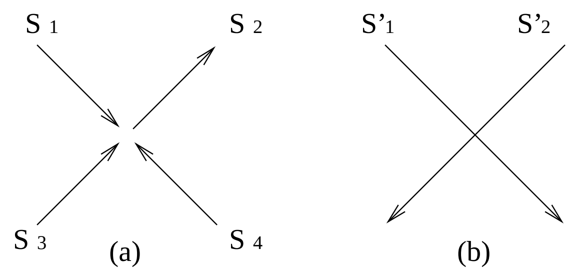


Figure 4.15: Various Writing Sequences (a) Unnatural Writing Sequence for “X” (b) Normal Writing Sequence for “X”.

recognizers fail. We wish to manipulate the strings to make it as in Figure 4.15 (b). As mentioned earlier, we assume that the sample image is size, position, slant corrected, smoothed and deskewed sequences.

Let $\bar{s_x}$ be a reversed writing sequence string of s and (s_x, s_y) be a concatenated string of s_x and s_y strings. First, the step $\xrightarrow{1}$ generates all possible strings by the concat and reverse operation. s_1 and s_2 are concatenated because the position of the terminal element of s_1 is very close to that of the starting element of s_2. $(s_3, \bar{s_4})$ becomes one string by first reversing s_4 and then concatenating s_3 and $\bar{s_4}$. There are 12 possible ways to generate a new set of strings without considering the order.

Second, in the step $\xrightarrow{2}$, we eliminate all elements in the new set of strings which violate the top-down and left-to-right sequence rule. For example, all elements containing a string $(s_4, \bar{s_3})$ are eliminated because $(s_4, \bar{s_3})$ is right-to-left sequence string.

$$
\begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix}
\xrightarrow{1}
\begin{pmatrix}
(s_1, s_2)(s_3, \bar{s_4}), & (s_1, s_2)(s_4, \bar{s_3}) \\
(s_1, \bar{s_3})(\bar{s_2}, \bar{s_4}), & (s_1, \bar{s_3})(s_4, s_2) \\
(s_1, \bar{s_4})(s_3, s_2), & (s_1, \bar{s_4})(\bar{s_2}, \bar{s_3}) \\
(\bar{s_2}, \bar{s_1})(s_3, \bar{s_4}), & (\bar{s_2}, \bar{s_1})(s_4, \bar{s_3}) \\
(\bar{s_2}, \bar{s_3})(s_4, \bar{s_1}), & (\bar{s_2}, \bar{s_4})(s_3, \bar{s_1}) \\
(s_3, \bar{s_1})(s_4, s_2), & (s_3, s_2)(s_4, \bar{s_1})
\end{pmatrix}
\xrightarrow{2}
\begin{pmatrix}
(s_1, s_2)(s_3, \bar{s_4}) \\
(s_1, \bar{s_3})(\bar{s_2}, \bar{s_4}) \\
(s_1, \bar{s_4})(\bar{s_2}, \bar{s_3})
\end{pmatrix}
\xrightarrow{3}
$$

$$
\xrightarrow{3}
\begin{pmatrix}
(s_1, s_2)(s_3, \bar{s_4}) \\
(s_3, \bar{s_4})(s_1, s_2) \\
(s_1, \bar{s_3})(\bar{s_2}, \bar{s_4}) \\
(\bar{s_2}, \bar{s_4})(s_1, \bar{s_3}) \\
(s_1, \bar{s_4})(\bar{s_2}, \bar{s_3}) \\
(\bar{s_2}, \bar{s_3})(s_1, \bar{s_4})
\end{pmatrix}
\xrightarrow{4}
\begin{pmatrix}
(s_1, \bar{s_4})(\bar{s_2}, \bar{s_3}) & 95\% \\
(s_1, \bar{s_3})(\bar{s_2}, \bar{s_4}) & 87\% \\
(\bar{s_2}, \bar{s_4})(s_1, \bar{s_3}) & 30\% \\
(\bar{s_2}, \bar{s_3})(s_1, \bar{s_4}) & 20\% \\
(s_1, s_2)(s_3, \bar{s_4}) & 10\% \\
(s_3, \bar{s_4})(s_1, s_2) & 5\%
\end{pmatrix}
\xrightarrow{5}
\begin{matrix} (s_1, \bar{s_4})(\bar{s_2}, \bar{s_3}) \\ \text{class “X”} \end{matrix}
$$

Figure 4.16: Overview of character recognizer with string concat and reverse capability

To compute whether the string violates the rule, scan the string and sum their corresponding values in Eqn. 4.13.

$$
\begin{pmatrix} \nwarrow & \uparrow & \nearrow \\ \leftarrow & & \rightarrow \\ \swarrow & \downarrow & \searrow \end{pmatrix}
=
\begin{pmatrix} -2 & -3 & -1 \\ -2 & & 2 \\ 1 & 3 & 2 \end{pmatrix}
\tag{4.13}
$$

If the summed value is greater or equal to 0, we accept the string. Otherwise, the string violates the rule and we reject the string. After this filtration, only three strings are left out.

Since the order of strings was not considered, we generate all possible order of strings for each element. The resulting elements are produced by the step $\xrightarrow{3}$. All sequence of strings are candidates for the recognizer. In step $\xrightarrow{4}$, a recognizer takes each candidate and returns its confidence with its class. The resulting set is the ordered list of elements with confidence.

Finally, in step $\xrightarrow{5}$, the element with the highest confidence is chosen. The output sequence, $(s_1, \bar{s_4}) \rightarrow (\bar{s_2}, \bar{s_3})$ is the writing sequence of Figure 4.15 (b).

Ring Operator

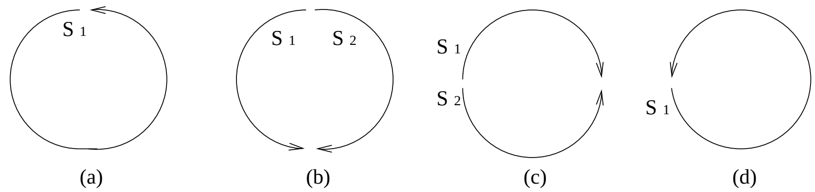


Figure 4.17: Various ways of drawing "O"

Concatenate and reverse string manipulation operators are insufficient to solve the writing sequence invariance problem. One exceptional case is a stroke sequence that forms a ring as depicted in Figure 4.17. We wish Figure 4.17 (b), (c) and (d) to be the equivalent writing sequence as Figure 4.17 (a). This requires a special treatment called, a *ring operator*. When (a) stroke(s) form(s) a ring, i.e, the start and end points are met, a new string starts from the top most point of the string.

Algorithm 4 shows the procedure to achieve the normalized ring that is drawn counter-clockwise from the top.

Algorithm 4 *Ring Normalization*

top = 0;

for $s_{o,1}$ *to* $s_{o,n}$

  if($s_{o,i} \in \{\uparrow, \nearrow, \nwarrow\}$*)*

    top = i + 1;

get $s_o^{'}$ *starting from* $s_{o,top}$

if (average of First quarter of $s_o \simeq \searrow$*)*

  $s_o^{'} = \bar{s_o^{'}}$.

For the example of Figure 4.17 (c), let $s_1 = (\nearrow, \rightarrow, \searrow, \downarrow)$ and $s_2 = (\downarrow, \searrow, \rightarrow, \nearrow)$. Then a new ring string is formed, $s_o = (s_1, \bar{s_2}) = (\nearrow, \rightarrow, \searrow, \downarrow, \swarrow, \leftarrow, \nwarrow, \uparrow)$. Now scan the s_o to find the top (= 2nd position of the string). Next, a new ring string, $s_o^{'}$ is generated starting from the top: $s_o^{'} = (\rightarrow, \searrow, \downarrow, \swarrow, \leftarrow, \nwarrow, \uparrow, \nearrow)$. Since the string is a clockwise sequence ring, reverse s_o' to get the final string. As a result, we have a normalized ring $s_o' = (\swarrow, \downarrow, \searrow, \rightarrow, \nearrow, \uparrow, \nwarrow, \leftarrow)$. Other writing sequences in Figure 4.17 (b) and (d) are handled similarly.

Sub-string Removal

Another example that the most on-line character recognizers fail according to our tests is a double or multiple overwritten strokes case. As illustrated in Figure 4.18, one first

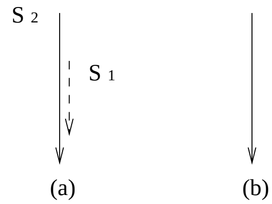


Figure 4.18: Double stroke

writes the s_1 and then overwrites s_1 with the s_2 expecting that the readers recognize it as *"1"*. Clearly, the spatial information is *"1"* but the temporal information is very messy. Aforementioned concatenation technique cannot handle this problem. In this section, we present a *sub-string removal* technique to overcome this issue.

First, check whether both start and end points of a string are laid on another string. If so, they are subject to the sub-string removal procedure. We use the approximate string matching algorithm that computes the edit distance to identify the sub-string occurrence in the longer string with small errors. Since string element type is angular, we use the edit distance defined in [15]. If the edit distance is within a small threshold, we remove the smaller string.

Recognizer

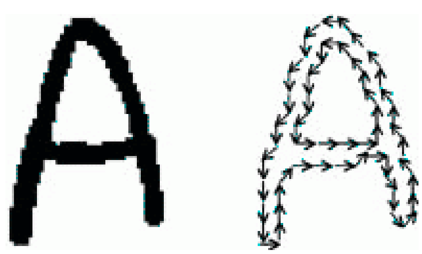
Once on-line handwritten character patterns are well pre-processed by techniques described in the previous sections, we are ready to classify the pattern into its class.

Numerous methods are available for on-line handwritten character recognition and enumerated in a few exhaustive survey papers [82, 72, 102].

The problem of *on-line handwritten character recognition* can be formalized by defining a distance between characters and finding the nearest neighbor in the reference set. To recognize an unknown on-line handwritten character, one measures the edit distances between the input string and reference strings [81, 77, 14, 15]. Next, the class of an input string is determined by votes on k-nearest neighbors. As a stroke sequence signifies the shape of the individual letters, a letter *"a"* is distinguished from a letter *"b"* by its different stroke sequences.

4.3.3 Off-line Character/Digit Matching

The off-line character/digit recognizer takes a character or digit image as an input and classifies its class as shown in Figure 4.8 (c). Since it is difficult to extract the stroke directional sequence string in the off-line images automatically, we use a *contour directional sequence string* or *CDSS* in short, instead. A *CDSS* is defined using the idea of the chain-code. Like a chain-code, a contour sequence is a representation of boundaries of objects in the image. Note that a stroke is 7 pixel long as defined in Def. 4.2. Geometrically best fitting strokes are selected to replace pixel-based strokes in the chain-code. Figure 4.19 shows the contour sequence representation of a character *"A"*. There are one outer contour sequence and one inner contour sequence



(a) (b)

Figure 4.19: (a) original character image "A" (b) contour sequence representation.

stings.

We define the abstract data type of a contour sequence string as follows:

Definition 4.3.1 Contour sequence string

struct Contour_Seq_String {

cctype	:	*boolean* {*inner or outer*}
cclength	:	*integer*
cclist	:	*a list of strokes*
centroid	:	*a coordinate,* $(\bar{x}, \bar{y})$

}

A centroid is the center of a contour sequence:

$$(\bar{x}, \bar{y}) = \left(\frac{\sum_x \sum_y xB(x,y)}{\sum_x \sum_y B(x,y)}, \frac{\sum_x \sum_y yB(x,y)}{\sum_x \sum_y B(x,y)} \right).$$

$B(x, y) = 1$ if the pixel is labeled as an element of the segment and $B(x, y) = 0$ otherwise. Note that the centroid is computed after the character image is resized by adjusting the height to a fixed size. Both *cctype* and *centroid* are important because a character image has multiple contour sequence strings and the decision for selecting a corresponding contour sequence string must be made.

To extract contour sequence strings, several procedures are preceded such as noise removal, connected component analysis and chain-code generation. Although we expect the input character or digit image to be a single clean image, there are many undesired noises. Some broken or smeared character images occur due to the degradation of document images. Moreover, poor segmentation and improper background removal create certain noises.While the image restoration is beyond the scope of this paper, we use simple unary and binary filters that handle salt and pepper noises. The *i-dots* in i and j are preserved.

Second, perform a connected component analysis and find the centroid of every connected component. Next, generate chain-codes by following component outer most boundaries. Finally, starting from the top of each chain-code, generate the contour sequence by fitting strokes to the chain-code. A stroke is 7 pixel long as defined in Def. 4.2. Geometrically best fitting strokes are selected to replace pixel-based strokes in the chain-code.

Next, we find the closest pair from two contour directional sequence strings by considering the inner or outer type, centroid and length of a string as criteria. In order to speed up the search time, only those reference characters with the same number of inner and outer type *CDSS*'s as the query character. We only need to compare the strings of the same type. When there are multiple *CDSS*'s of the same type, the distance between their centroids are computed to select the candidate pair.

After finding all corresponding contour sequence strings, compute each edit distance and accumulate all distances. If there is a string without a corresponding one, add the length of the string times the penalty value 2 to the total edit distance. Finally, select top 5 similar templates whose edit distances are smallest. The class of a query image is determined from a vote of these five templates.

We consider one of *CEDAR digit databases*, called *BR training set* which consists of $18,465$ digit images. The database is divided into 18 different sets resulting in $1,000$ instances each set. The first set is used as the reference or prototype images and the rest sets are used as test sets. Using the *k-NN* classifier, 17 experiments were performed and the average accuracy is about 96.08%. We achieve significant improvement over the original Levenshtein edit distance where the accuracy is about 94.02%. From the error analysis, we find an interesting consistency that the majority of errors are due to the broken characters resulting unexpected discontinuity in the *CDSS*. It is expected that provision on the broken characters will improve the performance significantly.

4.4 Conclusion

In this paper, we categorized strings into four types: nominal, angular, magnitude and cost-matrix. We extended the *Levenshtein edit distance* to handle the angular and linear type strings. It is good for matching stroke and contour directional sequence strings. It takes *turn* and local context into account to compute the edit distance. This technique performs better than Levenshtein edit distance with cost matrix.

We also presented string distance measures to solve writer verification, on-line and off-line character recognition. To perform this, we converted a two-dimensional image to one-dimensional strings and then we measured the edit distance between strings. The smaller the edit distance is, the more similar they look like to each other. The fundamental idea underlying pattern recognition using the approximate string matching is based on the nearest neighbor classification. During training, we stored a full training set of strings and their associated category labels. During classification, a test string was compared to each stored string and the edit distance was computed. The test string was then assigned the category label of the nearest string in the training set.

We used two very important features, *SDSS* and *SPSS* to solve the *writer verification* problem. It is an unique approach to represent the *pressure* of handwriting as a sequence of pressure extracted from an off-line character image. In all, the proposed semi-automatic method provides a distinction or similarity between two handwritings figuratively and numerically. It is expected to help greatly document examiners and signature verifiers to compare handwritings and signatures.

Another major contribution in on-line character recognition is to diminish the effect of unnaturally written characters. Two string manipulation operators, concatenation and reverse, are used as a pre-processing to reduce the effect.

Chapter 5

Auxiliary Distance Measures

[1] In this chapter, we consider two additional distance measures: binary vector distance for GSC features and convex hull distance for ordinal multi-dimensional features.

First, at the heart of research in *GSC* Character Recognizer lies the hypothesis that feature sets can be designed to extract certain types of information from the image. Another important issue is pattern matching which exploits the similarity measure between patterns. Gradient, Structural and Concavity features are regarded very important features and GSC classifier gives the best digit recognition performance among other currently used classifiers. In this paper, we present a technique to evaluate similarity measures using the error vs. reject percentage graph and find a new similarity measure for a compound feature: GSC features. Since the optimized similarity measure performs better on a different testing set than the previously used similarity measure, we claim that an improvement in off-line Character Recognition is achieved.

Second, we present a prototypical convex hull discriminant function. As an output, the program gives the geometrical significances of an unknown input to all classes and helps determine its possible class. This technique is particularly useful in the *Writer Identification* problem, in which the number of samples is limited and very small. Convex hulls of all samples in each document in the reference set are computed as one's style of handwriting during a preprocessing. During the query classification

[1] This chapter contains works published in [18] and a machine learning course project.

process, for all samples in the query document, the average distances to the convex hull of each reference document are computed. The author of the document whose average distance is the smallest or within a certainthreshold value is considered as a candidate for possible author of the query document.

5.1 Learning Similarity Measure for GSC Compound Features

One common method for classifying an unknown input vector involves finding the most or top k similar vectors in a reference set. The k-nearest neighbor or simply k-*nn* has wide acceptance in Character Recognition (see [32, 33] for extensive surveys). There are two important goals in this approach. One is selecting important features from a character image. The other is selecting a similarity measure method.

CEDAR has successfully found and utilized numerous significant features to recognize a character of unknown class. Gradient, Structural and Concavity features are those. Since Favata, Srikantan and Srihari introduced this multiple feature/resolution approach, it has undergone a period of reappraisal of its effectiveness and consideration of alternatives or modification to improve the performance further. While GSC features are considered significant ones, relatively little study has been carried out to select and design a good similarity measure for GSC recognizer. In this paper, we introduce a way to evaluate the similarity measure and find the optimal similarity measure according to the evaluation.

There exist several definitions encountered in various fields such as information retrieval and biological taxonomy [35]. Common definitions include *Euclidean, Minkowski, cosine, dot product, Tanimoto distance*, etc. We examine most currently known similarity measures on *bd-testing data* which consists of mixed hand-printed/cursive characters to select the best one.

Instead of simply using equi-weighted multiple features, we regard them as a compound feature with different weights. It is the weighted additive model of combining a different feature. A improvement in off-line Character Recognition performance is achieved.

5.1.1 Review of GSC features

Among many classifiers, the *Gradient, Structural, and Concavity*, simply known as *GSC classifier*, has the highest accuracy [42]. It is based on the philosophy that feature sets can be designed to extract certain types of information from the image [41, 42, 95, 96]. These types are gradient, structural, and concavity informations. Gradient features use the stroke shapes on a small scale. Next, structural features are based on the stroke trajectories on the intermediate scale. Finally, concavity features use stroke relationship at long distances. All features are listed in Table 5.1. The

	Gradient		**Structural**		**Concavity Features**	
Grid Pos.	*ID*	*Directional*	*ID*	*Rule**	*ID*	*Concavity*
(0,0)	G01-00	$1° \sim 30°$	S01-00	r_1	C-CP-00	Coarse Pixel Density
⋮	⋮	⋮	⋮	⋮	⋮	⋮
(3,3)	G01-33	$1° \sim 30°$	S01-33	r_1	C-CP-33	Coarse Pixel Density
(x,y)	G02-xy	$31° \sim 60°$	S02-xy	r_2	C-HR-xy	horizontal run length
(x,y)	G03-xy	$61° \sim 90°$	S03-xy	r_3	C-VR-xy	vertical run length
(x,y)	G04-xy	$91° \sim 120°$	S04-xy	r_4	C-UC-xy	Upward concavity
(x,y)	G05-xy	$121° \sim 150°$	S05-xy	r_5	C-DC-xy	Downward concavity
(x,y)	G06-xy	$151° \sim 180°$	S06-xy	r_6	C-LC-xy	Left concavity
(x,y)	G07-xy	$181° \sim 210°$	S07-xy	r_7	C-RC-xy	Right concavity
(x,y)	G08-xy	$211° \sim 240°$	S08-xy	r_8	C-HC-xy	Hole concavity
(x,y)	G09-xy	$241° \sim 270°$	S09-xy	r_9		
(x,y)	G10-xy	$271° \sim 300°$	S10-xy	r_{10}		
(x,y)	G11-xy	$301° \sim 330°$	S11-xy	r_{11}		
(x,y)	G12-xy	$331° \sim 360°$	S12-xy	r_{12}		

$*$ see [41] for description of rules in details.

Table 5.1: GSC Features where $x = 0 \cdots 3$ and $y = 0 \cdots 3$

input character image is a binarized and slant normalized image. A bounding box is placed around the image and it is divided into 4×4 grid as shown in Figure 5.1. This

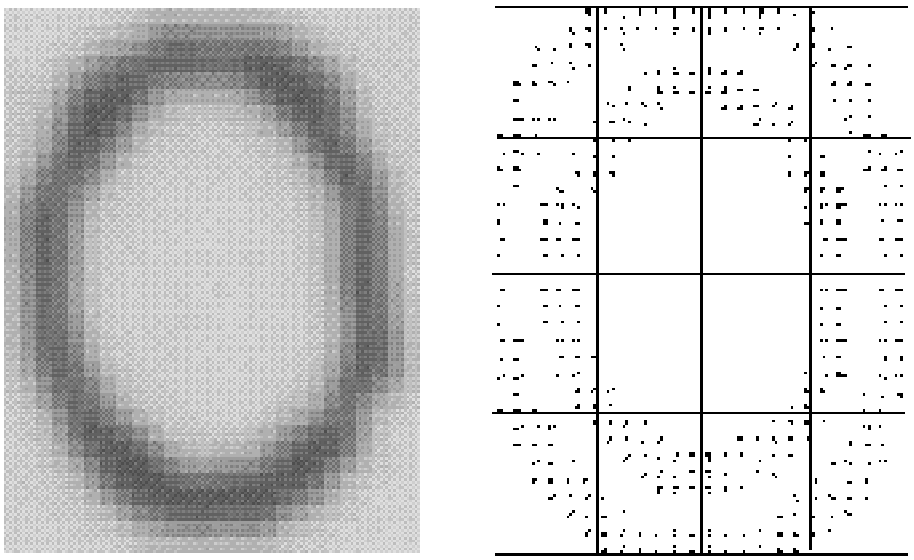
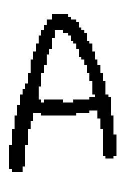


Figure 5.1: 4×4 *grid*

approach is known as a quasi-multiresolution approach. For each grid, all directional, rules and various concavity features are checked. Therefore, Gradient, Structural, and Concavity have 192, 192, and 128 features correspondingly. In all, there are 512 number of features. A sample vector for a character *"A"* is given in Figure 5.2



Gradient	:	0000000000110000000011000011100000001110000000110000001100010000
($192 bits$)		0000110000000000000111001100011111000011100000000010010100000100
		0111001111100111110000001000000100000000000000000010000001001000
Structural	:	0000000000000000000011000011100010000100001000000100000000000001
($192 bits$)		0010100000000001100000101001100001100000000000010010001100110000
		0000000000110010100000000000001100000000000000000000000000010000
Concavity	:	1111011010011111011001100000011011110110100110010000011000001110
($128 bits$)		0000000000000000000000000000000000000000111111100000000000000000

Figure 5.2: *A sample character and its GSC feature vector*

In order to classify an input vector into one of 26 alphabets, the k-nearest neighbor (k-nn) approach is used. To compute the distance, the following definition of similarity is previously used.

$$S[x,y] \quad = \quad x^t y + \frac{\bar{x}^t \bar{y}}{\sigma} \tag{5.1}$$

σ is the contribution factor and usually $1 < \sigma < 5$. The term $x^t y$ indicates the number of 1 bits that match between x and y and the term $\bar{x}^t \bar{y}$ denotes the number of 0 bits that match between them.

Various similarity measures such as *Euclidean, Minkowski, cosine,* and *dot product* have been examined and the definition 5.1 culminates over these definitions. Also, we observe the changes in the recognition performance when various different contribution factors σ are used and found the optimal value, $\sigma = 1.9$ where $\sigma = 2$ in the previous GSC classifier on a particular testing set.

5.1.2 Similarity Measure Evaluation

It is important to choose suitable similarity measure between feature vectors. Several definitions are available and in this section, we will discuss how to evaluate a similarity measure. One can test similarity measure by using a test set but it is very test set dependent. One general way is to divide samples into subsets randomly. For example, a set is divided into k sets for k-fold cross validation [68]. The tuning will be done by the repetitive way to avoid the accidental extremity of one set. The extreme cases will be averaged out by the repetition. Tuning can be misled by random errors and coincidental regularities within the tuning set if we use a single tuning set. However, obtaining many reasonably large subsets is quite costly and hard.

For this reason, we propose a different evaluating method for k-nearest neighbor classifiers. It is *"All k-nearest neighbor* approach. We will perform *All k-nearest neighbor* algorithm for the reference set and count mismatches. *All k-nearest neighbor* problem is to determine k-nearest neighbors for each point of the reference set R. Since we will use this entire reference set to classify input vectors of unknown class, we would like to choose the similarity measure that best clusters reference samples of the same classes.

Consider Fig. 5.3. It is two-dimensional features and type of features are continuous. Suppose $k = 2$. In Fig. 5.3, every element points its two nearest neighbors.

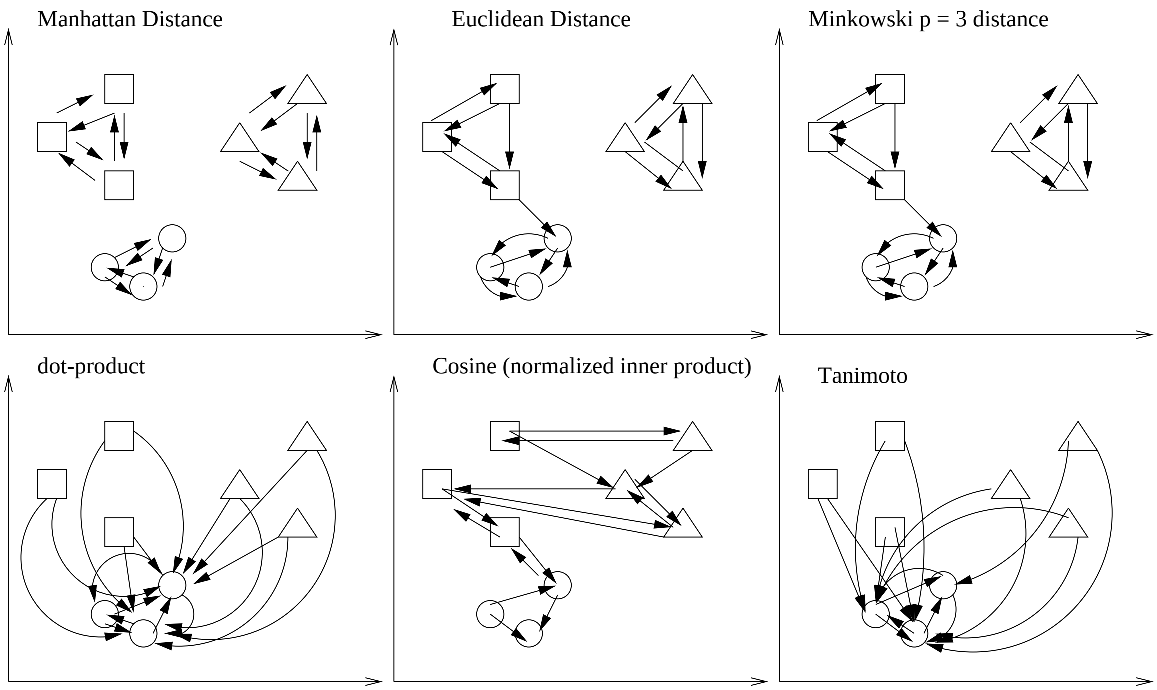


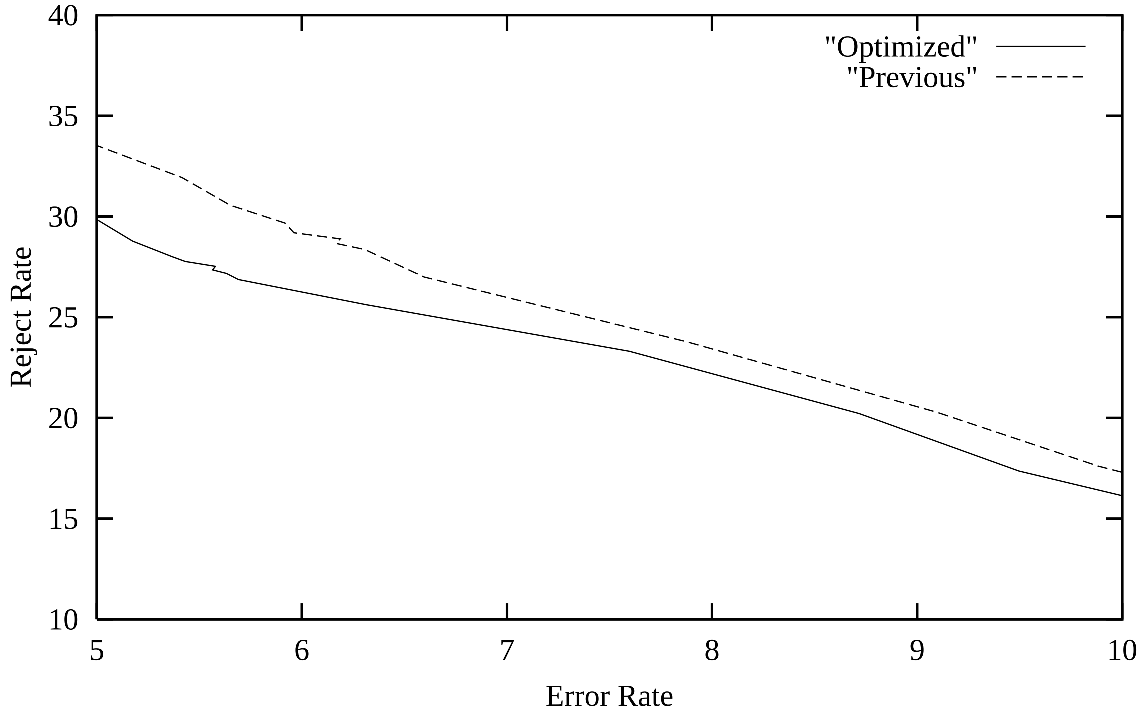
Figure 5.3: *All k-nearest neighbor graph*

Depending on measures, nearest neighbors are different. Now we count the errors for each measure. The evaluation formula is:

$$E(x) = \frac{\text{Number of errors}}{k \times n}$$

where n is the number of samples. For example, $E(\text{Manhattan}) = 0$ while $E(\text{Euclidean} = 0.0556$. We certainly would like to use the Manhattan distance as the measure rather than other measures.

Although computing all k-nearest neighbor takes only $O(n \log n)$ [83], in case of binary feature vectors or non-Euclidean space features cannot be solved in this manner. However, it can be solved surely in $O(n^2)$; apply the the nearest neighbor search for each element in the reference set.



$$\text{Error rate} = \frac{\text{Number of Errors}}{\text{Number of Accepted Queries}} \times 100$$

$$\text{Reject rate} = \frac{\text{Number of Rejections}}{\text{Number of Total Queries}} \times 100$$

Figure 5.4: *Error vs. Reject Percentage Graph.*

It is the simplest way of evaluation to select one whose the error rate is the minimum. Another novel approach is using the error versus reject percentages of recognition graph [31] as shown in Fig. 5.4. The x and y axis represent the error rate and reject rate, respectively. A good recognizer must be as close to the axes as possible. In other words, the measure whose the area between error 5% to 20% is minimized is the best measure. Therefore, the optimized similarity measure is better than what is used previously. We will use this approach to evaluate and optimize the similarity measure. Since computing the exact size of the area is hard, we simply add the heights at $5, 10$ and 20% error rates.

5.1.3 Compound Feature

In contrast to the definition 5.1, we divide the feature set into three inhomogeneous groups: gradient feature set, structural feature set and concavity feature set. We find the optimal similarity measure for each set. Then we use the additive model to combine these measures into one as follows:

$$\begin{aligned} S[x,y] &= \alpha S'[x_g,y_g] + \beta S''[x_s,y_s] + \gamma S'''[x_c,y_c] \\ &= \alpha(x_g^t y_g + \frac{\bar{x_g}^t \bar{y_g}}{\sigma_g}) + \beta(x_s^t y_s + \frac{\bar{x_s}^t \bar{y_s}}{\sigma_s}) + \gamma(x_c^t y_c + \frac{\bar{x_c}^t \bar{y_c}}{\sigma_c}) \end{aligned} \tag{5.2}$$

x_g is a subset of x and it has only gradient feature part of x. $x = x_g \cup x_s \cup x_c$. First, we consider each individual set of GSC feature sets. For $S'[x_g, y_g]$, $S'[x_s, y_s]$ and $S'[x_c, y_c]$, we use the definition 5.1. To find each optimal respective σ's, the same evaluation technique will be used to optimize the performance. The least size of area occurs at $\sigma_g = 2.5$ for gradient features, at $\sigma_s = 2.8$ for structural features and finally at $\sigma_c = 1.7$ for Concavity features (see the appendix for the full data).

Next once σ_g, σ_s and σ_c are determined, we find α, β and γ which are weight coefficients for each similarity measure of a set. This is three dimensional space optimization problem. We search for the best weight coefficients: α, β and γ. The experiment is performed on one of the fastest *Sun Micro-systems* in *CEDAR*. The machine has a *300MHz UltraSparc, SunOS 5.6, 4×782 MIPS*, and *2,048 Memory*. In our reference set in *GSC*, there are 21800 number of characters (A-Z), and approximately 800 references per character. Each epoch (computing all-k-nearest neighbors for all vectors in the reference set) takes approximately 20 minutes. The program has been run over a week and found $\alpha = 0.7, \beta = 1.3$ and $\gamma = 1.6$ (see the Appendix for the full result data). This may not be the global optimal value in the space $0.5 \geq \alpha, \beta, \gamma \geq 2.0$. This program is still running to check all the space but we achieved the improvement for currently known optimal values. Thus it may improve further in about next two weeks.

So far, we have shown the optimized all k-nearest neighbor on the reference set. To validate that this is truly better definition of similarity measure, we use the *bd-testing data set* as a validation set, which consists of mixed hand-printed/cursive characters. The size of this validation set is 1681. This validation set will server as a safety check for better performance. Figure 5.4 shows the improvement on this validation set. The dotted line indicates the performance of equation 5.1 that is previously used in GSC classifier and the solid line indicates the performance of equation 5.2 that is the optimized one. Clearly, the size of the area of the optimized version is smaller than that of the previous one. We can conclude that we have improved the performance of the GSC classifier on off-line character recognition by changing the similarity measures.

5.2 Convex Hull Distance Analysis

Given a set of reference documents and a query document, the *Writer Identification problem* is to find the possible author of the query document among the authors of reference documents. Document examiners or handwriting analysis practitioners find important features to characterize individual handwriting because certain features are consistent with writers in normal undisguised handwriting [6]. Authorship may be determined by retrieving documents that are similar to the query document due to the following hypothesis; people's handwritings are as distinctly different from one another as their individual natures, as their own finger prints.

Suppose that there are N number of disputed documents in the database and one query document. We assume that all features are extracted and stored in the database. In order to find out the authorship, the document examiner may have to retrieve all documents in the database to compare them with the query document one by one. It is very time consuming for examiners to look at all images. It would be nice if a small candidate set with a few documents is selected from the large database automatically or enumerated in order of its similarity value.

This model is called the *"filtration model"* with posed query: "select all reference documents that are consistent with a query document." Ideally, the number of retrieved documents is very small ($\approx 5 \sim 10\%$). Once the filtered documents are obtained, quantitative methods for finer analysis can be used.

Various techniques may be applied to this filtration problem. They are *Density estimation*, *Fisher* or other linear discriminant functions, *k-nearest neighbor estimation* or *Fuzzy classification* [36]. The effectiveness of these approaches depends on the sufficiently large number of samples in each reference class. However, in handwriting identification, the number of samples is very small and it is hard to generalize the style of one's handwriting.

For this reason, we take a different approach. The new discriminant estimation function is called the *Convex Hull discriminant function.* It is a technique to access the geometrical significance of patterns. Convex hulls of all samples in each document in the reference set are computed as a preprocessing. During the query classification process, for all samples in the query document, the average distances to each convex hull of the reference document are computed. The author of the document whose average distance is the smallest or within a certain threshold value is considered as a candidate for possible author of the query document.

The dimension of samples is often more than two dimensions. There is a wealth of literature regarding the problem of finding convex hulls in more than two dimensions dating as far back as 1970. Chand and Kapur proposed the *"Gift Wrapping"* method also known as the *"Subfacet-based"* method [27] analyzed later by Bhattacharya [5][83]. About a decade later, the *Beneath and Beyond"* method had been proposed by Kallay [59][83] (see [83][38] for the extensive study).

5.2.1 Ordinal Measurement Type Features

Features can be categorized in terms of their measurement types: nominal, ordinal, modulo, hierarchical, and so forth [65]. Existence of ligature or hiatuses, pen type and writing material belong to nominal type features. Height, width, etc belong to ordinal type features. Gradient direction features belong to the modulo type feature.

In this paper, however, we discuss only the ordinal measurement type features because these features can be plotted in the Cartesian coordinate system and solved by the computational geometry. In other words, an ordinal multivariate statistical sample can be viewed as a point in Euclidean space [83]. Although we consider all types of features in the ultimate system, we use ordinal type features only in this paper to demonstrate the convex hull discriminant function.

5.2.2 Algorithm

Here is an algorithm for accessing the geometrical significance of patterns by using the convex hull discriminant function:

Input: A query document D_q and a set of N documents from the database $D = \{D_1, D_2, \cdots, D_N\}$. A document D_i has m samples and there are f number of features. A j'th sample is represented as a f dimensional point $(p_1(j), p_2(j), \cdots, p_m(j))$ and a document D_i is represented as a set of f dimensional points: $D_i = \{(p_1(1), p_2(1), \cdots, p_f(1)), \cdots, (p_1(m), \cdots, p_f(m))\}$. $CH(D_i)$ denotes the convex hull of the points in the document D_i.

Output: An ordered list of documents D_i in D by the distance:

$$d_{\text{ave}}(D_q, CH(D_i)) = \frac{\sum_{j=1}^{m} d(q(j), CH(D_i))}{m} \tag{5.3}$$

where m is the number of points in D_q and $q(j) \in D_q$.

$d(q(j), CH(D_i))$ is the shortest distance between a point $q(j)$ to the convex hull $CH(D_i)$.

If the number of features is three, there are four cases of the distance as shown in Fig. 5.5. Case 1 is such that the query point is in the inside of the convex hull and

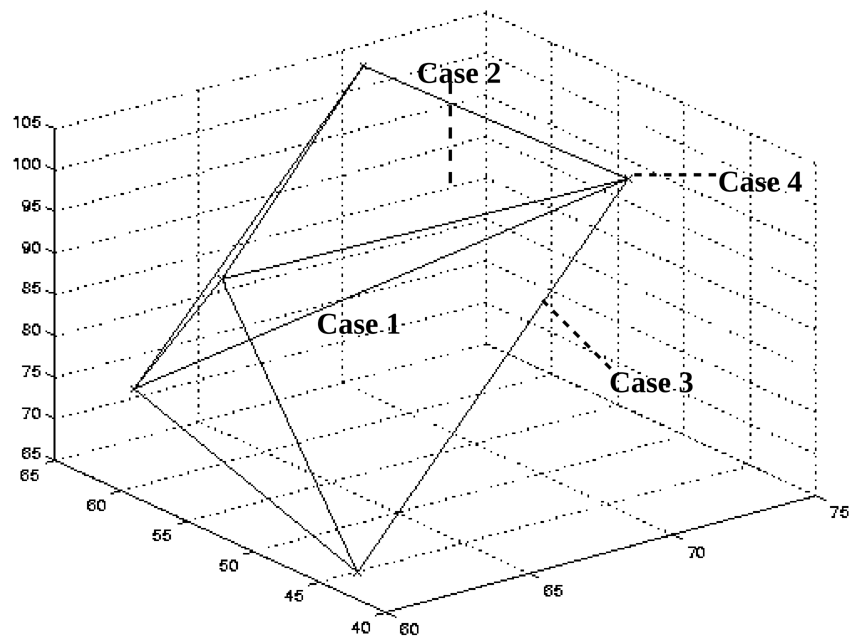


Figure 5.5: 4 *cases of distance from a point to a convex hull*

the distance is 0. Consider three points, P_1, P_2 and P_3 of the convex hull such that they consist one $(f-1)$ facet that is a triangle plane, $ax + by + cz + d = 0$. Let the intersecting point q' that makes the shortest distance between the query point q and the triangle plane. If q' is inside of the triangle, the distance is

$$\frac{|aq_1 + bq_2 + cq_3 + d|}{\sqrt{a^2 + b^2 + c^2}}$$

Note that the query point is in the opposite side of the plane to those points in the reference document. In case that q' is outside of the triangle, the distance can be from the query point to either $(f-2)$ or $(f-3)$ facets. $(f-2)$ facet is a line in the convex hull and $(f-3)$ facet is a point in the convex hull.

The output is the list of documents in descending order. As a result, the examiner will retrieve the document that is the most likely to the query document first. One can give a certain threshold value and retrieve documents whose distance is within the threshold value.

5.2.3 Prototype

We consider off-line versions of the pages of handwriting captured by the Human Language Technology (HLT) group at CEDAR [89]. The database contains both cursive and printed writing, as well as some writing which is a mixture of cursive and printed. The database has a total of twelve passages, selected from a variety of different genres of text. The passages reflect many different types of English usage (e.g. business, legal, scientific, informal).

Features

For the visualization purpose, we use three features from the letter *"W"*. where H and W are the height and width of the letter and H_p is the height of the middle peak of *"W"* and W_v is the x-distance between two valleys of *"W"* as shown in Fig. 5.6.

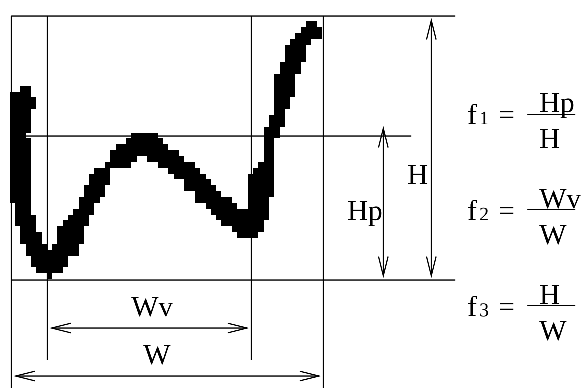


Figure 5.6: *Features from the letter "W"*

The relative values rather than absolute values are used as features in order to facilitate proper comparison among the collected samples due to the difference in size and slant angle.

Sample Documents

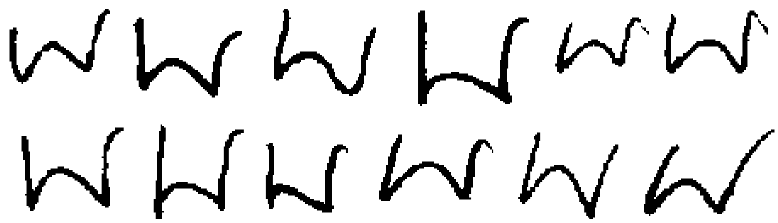
Consider one query document and five reference documents written by five different writers. To find out the authorship, we give an example of *"W"* among alphabets. Fig. 5.7 shows "W"'s extracted from the query document written by an unknown author. Those in Fig. 5.8 are from the reference documents written by possible authors or suspects. Note that contents of the documents are all different in this experimental exemplar. The reference documents are labeled as Document $A, \cdots, E$. The author of the reference document B is that of the query document.

Figure 5.7: *W's from query document*

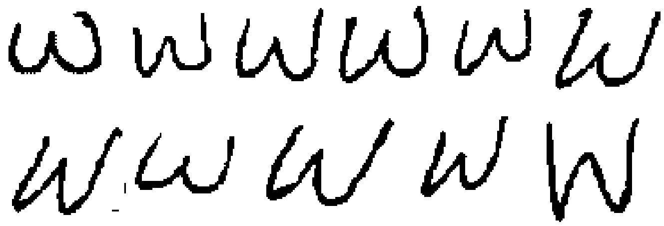
Each letter in one document differs from others and it is known as *intra-variation*; no one can write the exact same letter in the exact same way. "W"'s of different authors are also quite distinctively different and it is called *inter-variation*. We will show the geometrical significance of these inter and intra variations to determine the possible authorship.

3D Convex Hull Visualization and distance result

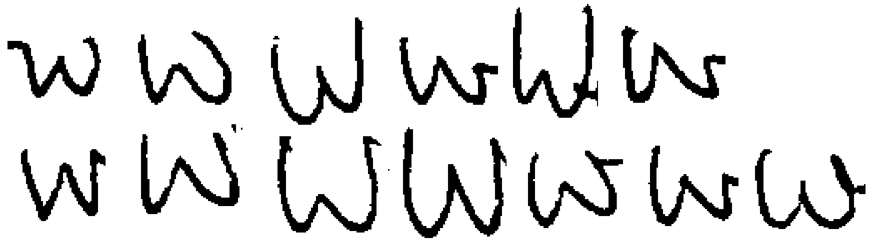
Figs. 5.9 through 5.13 show the relation between the convex hull of the query document and those of the reference documents from $A \cdots E$, respectively. $\diamond$ indicates the sample point of the query document and $\diamond \cdots \diamond$ indicates the boundary of the convex hull of the query document. x represents the sample point of the reference document



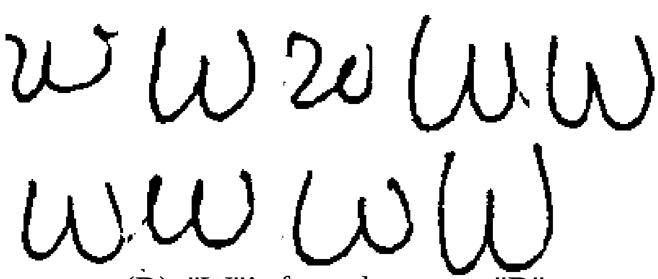
(A) "W"'s from document "A"



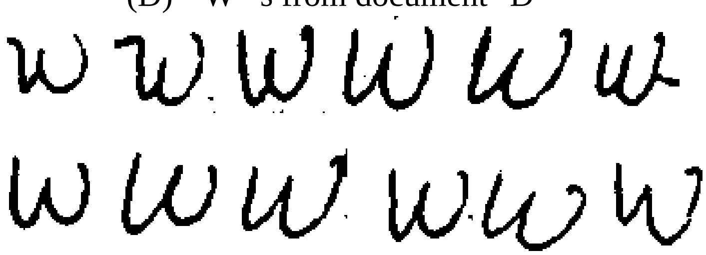
(B) "W"'s from document "B"



(C) "W"'s from document "C"



(D) "W"'s from document "D"



(E) "W"'s from document "E"

Figure 5.8: *W's from Reference Documents*

and $x-x$ indicates the boundary of their convex hull. As shown in Fig. 5.10, two convex hulls overlap each other more than any other relations do.

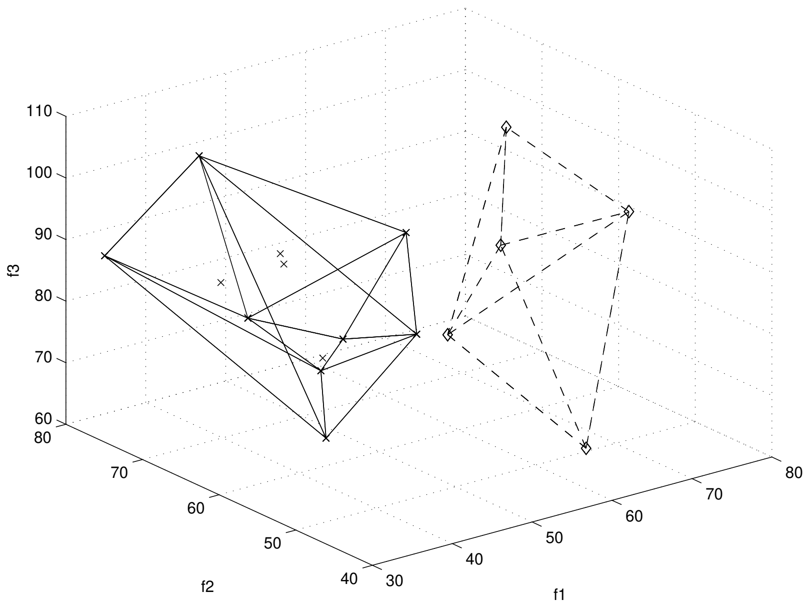


Figure 5.9: *Convex hulls from Document Q and A*

Table 5.2 shows the average distance from all sample points of query document to each convex hull. The table suggests that the author of the document *"B"* is

Table 5.2: Average distance to each convex hull

d-CH(A)	d-CH(B)	d-CH(C)	d-CH(D)	d-CH(E)
15.6080	3.7227	5.3908	9.4360	11.5649

that of the query document because the average distance of every sample in the query document to the convex hull of samples from document *"B"* is the smallest. Document *"C"* also has quite a small average distance. However, the distance is quite large in other letter cases while the small distances are still observed for the document *"B"*.

Table 5.3 shows the average distance from all sample points of query document to each convex hull. In the second row, the document A is considered as a query document and the average distance to the other reference documents are computed

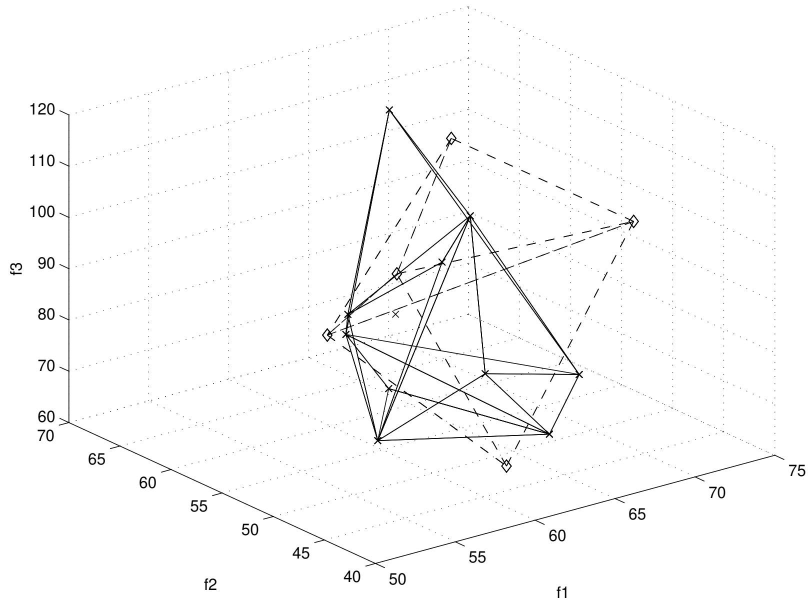


Figure 5.10: *Convex hulls from Document Q and B*

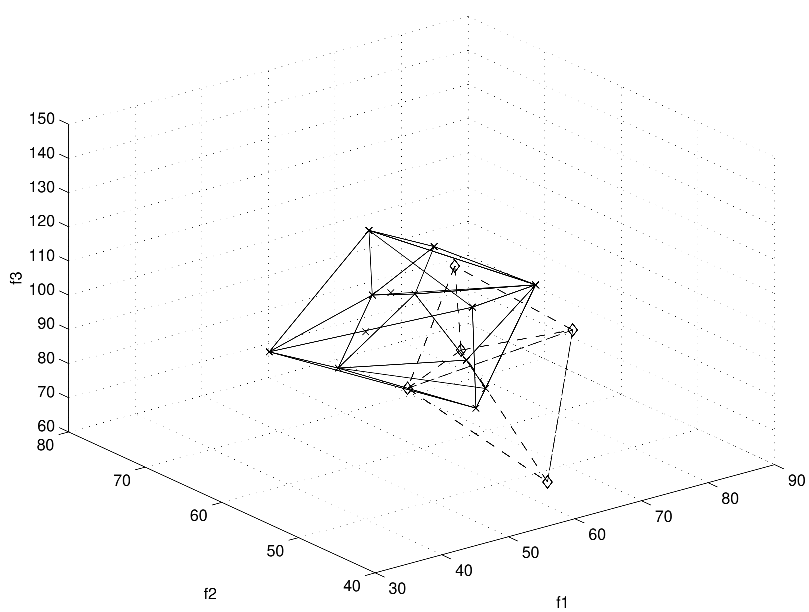


Figure 5.11: *Convex hulls from Document Q and C*

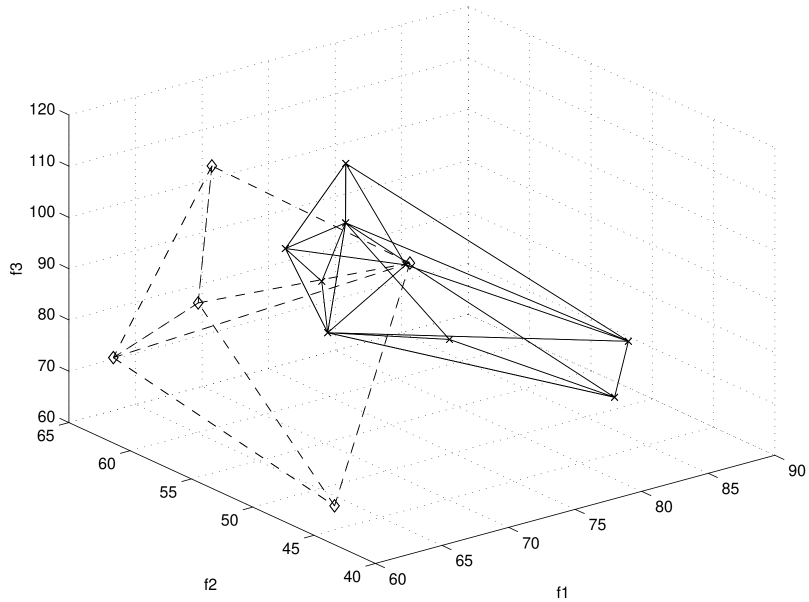


Figure 5.12: *Convex hulls from Document Q and D*

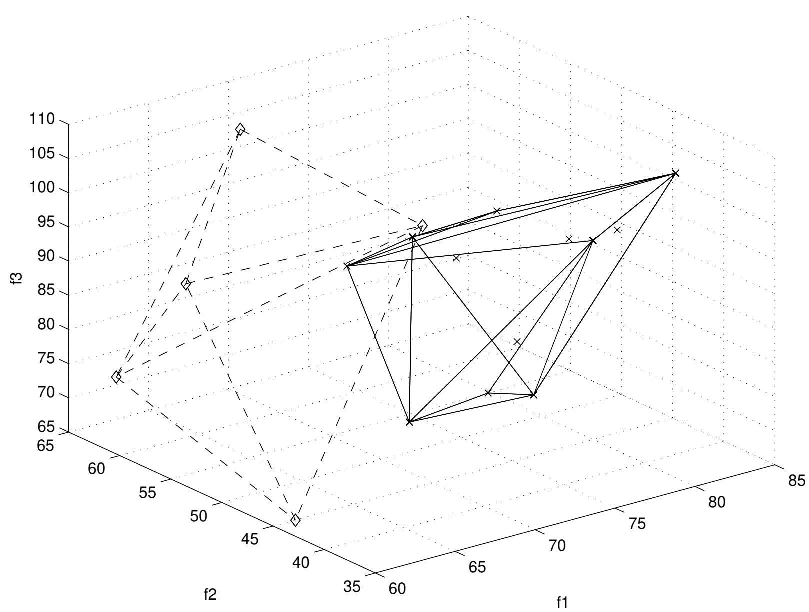


Figure 5.13: *Convex hulls from Document Q and E*

Table 5.3: Distance matrix of all document

	d-CH(A)	d-CH(B)	d-CH(C)	d-CH(D)	d-CH(E)
A	0.000	23.967	17.138	34.120	37.278
B	14.644	0.0000	7.463	15.489	13.206
C	18.485	18.395	0.000	22.148	25.250
D	20.452	5.972	4.610	0.000	6.347
E	27.204	10.326	11.693	5.411	0.000

and listed. A distance matrix is build as shown in Table 5.3 and it suggests its non-symmetry. The fifth row suggests that the author of the document C is likely to be that of the document D however, the fourth row indicates that the author of the document D is not likely to be that of the document C. The style of document C includes that of document D.

Consider two writers: one with a neat handwriting style and the other with a sloppy handwriting style. One with a neat handwriting style is capable of writing a sloppy style while it is not likely for one with a sloppy handwriting style to write neatly. Therefore, non-symmetric distance matrix is a desirable property in the *handwriting identification* problem.

5.3 Conclusion

To conclude, it is worth mentioning again that selecting and designing a similarity measure is as important as finding significant features. Poor choice of similarity measure would result in unsatisfactory performance in recognition. In designing a similarity function, it is necessary to have a tuning set in addition to a reference set and validation set if coefficients are associated with it. The major contribution is presenting a modified similarity measure for GSC recognizer and achieving a better performance on the off-line character recognition.

Determining the convex hull is a basic step in several statistical problems such as robust estimation, isotonic regression, clustering, etc [83]. We have shown another example that is *pattern classification* applied in *Writer Identification* application. A prototypical convex hull discriminant function is presented. The convex hull of the given samples is regarded as one's handwriting style of a particular letter.

This technique is useful in case that the number of samples is small. When the number of samples is large, the presented technique is still advantageous in terms of speed as it deals with only samples on the convex hull. However, it could give a wrong classification when the distribution is non-Gaussian or non-convex.

Chapter 6

A Fast Nearest Neighbor Search Algorithm by Filtration

[1] One common method for classifying an unknown input vector involves finding the most or top k similar templates in the reference set. Not surprisingly, this problem, so-called *k-nearest neighbor* or simply *KNN problem* has received a great deal of attention because of its significant and practical value in pattern recognition (see [32, 33] for extensive surveys).

6.0.1 History

One straight-forward method is computing feature by feature for all templates in the reference set and it takes is $O(nd)$ where n is the number of templates in the reference set and d is the number of features or dimension. This is very time consuming for users to wait for the output. Hence, there is a wealth of literature regarding computational expenses of the *KNN* problem dating as far back as 1970. Papadimitiou and Bentley showed $O(n^{1/d})$ worst-case algorithm [76] and Friedman, Bentley, and Finkel suggested possible $O(\log n)$ expected time algorithm [44]. There are two main streams of implementing a fast algorithm: lossy and lossless search algorithms. There are three general algorithmic techniques for reducing the computational burden: computing partial distance, pre-structuring, and editing the stored prototypes [36].

[1]This chapter contains works published in [22, 20].

Partial distance:

First, the partial distance technique is often called a *sequential decision technique*; decision for match between two vectors can be made before all features in the vector are examined. It requires a predetermined threshold value to reduce computation time.

Pre-structuring:

The most famous method focuses on preprocessing the prototype set into certain well-organized structures for the fast classification processing. Many approaches utilizing *multidimensional search trees* that partition the space appear in the literature [46, 62, 71, 7]. In these approaches, the range of each feature must be large. Otherwise, if features are binary, we achieve little speedup. Furthermore, the dimension of feature space must be low. Quite often in image pattern recognition, each feature is thresholded and binary and the dimension is high.

A different type of preprocessing on the prototypes has been introduced to generate a useful information that helps reduce the overall search time. As a result of the preprocessing, a metric can be built. In a study of utilizing the metric, Vidal *et al* [107] claimed that the approximately constant average time complexity is achieved only by the metric properties. Although it was their claim, what has been shown is that the average number of prototypes necessary for feature by feature comparison is constant [40]. It is $\Omega(d+n)$ on average and even $O(n^2+nd)$ in the worst case. In some applications, this approach is quite prohibitive as it requires $O(n^2)$ space and the number of templates is often too big.

Editing the stored prototypes:

Another important approach is the prototype reduction method. It reduces the size of prototype set to improve speed with sacrifice of accuracy. The condensed nearest neighbor rule [53] and the reduced nearest neighbor rule [47] are used to select a subset of training samples to be the prototype set. In this approach, we must sacrifice accuracy for speed. Hong *et al* [56] successfully implemented a fast nearest neighbor classifier for the use of Japanese Character Recognition. They combined a non-iterative method for *CNN* and *RNN* and a hierarchical prototype organization method to achieve a great speed-up with a little accuracy drop.

6.0.2 Proposal : Additive Binary Tree

Computing the distance for all templates in the reference set would be very time consuming for users to wait for the output. A threshold value may be used to reduce computation time such as the *sequential decision technique*; decision for match between two vectors can be made before all features in the vector are examined. We present a technique to speed up further than that.

The new algorithm utilizes both partial distance and prestructuring techniques. We reduce computation time by using an *Additive Binary Tree (ABT)* data structure that contains additive information which is frequency information in binary features. The idea behind the *ABT* approach in finding the nearest neighbor is filtration by which the unnecessary computation can be eliminated. It makes this approach unique from the others such as redundancy reduction or metric method. First, take a quick glance at the reference set and select candidates for match. Next, take a harder look only at those candidates selected from previous filtration to select a fewer candidates, and so on. After several filtration, take a complete thorough look only at the final candidates to verify them. All matches whose distance is less than or equal to a threshold are guaranteed to be in all candidate sets.

6.0.3 Organization

In this paper, we will describe the *additive binary tree (ABT)* and the new nearest neighbor search algorithm based on several famous similarity measures. We will give the simulated experimental results. Finally, we report the experimental results on our *OCR* system using *GSC* classifier (gradient, structural and concavity feature classifier).

6.1 Preliminary

The performance of pattern classification depends significantly on its definition of similarity or dissimilarity measure between pattern vectors. Several definitions have been proposed and encountered in various fields such as information retrieval and biological taxonomy [35]. Among many definitions, *Euclidean distance, absolute differences*, and *Minkowski distance* are the most famous ones. Section 3 discusses the algorithm where the distance measure function is the absolute difference. We chose this definition as it is the most simple measure to explain the new algorithm. *Euclidean* and *Minkowski's distances* are almost identical when features are binary. The absolute difference, well known as the *city block distance* or the *Manhattan metric*, is defined as follows:

Definition 6.1.1 Manhattan distance

$$D[x, y] = \sum_{i=1}^{d} |x_i - y_i|$$

where d is the number of features in vectors. A normalized inner product appears often as a non-metric similarity function:

Definition 6.1.2 A normalized inner product

$$S[x, y] = \frac{x^t y}{\|x\|\|y\|} \ \mathit{where}\ \|x\| = \sqrt{x^t x}$$

It is the cosine of the angle between two vectors. Another similarity definition used in *OCR* system using *GSC* classifier developed in CEDAR is:

Definition 6.1.3

$$S[x,y] = x^t y + \frac{\bar{x}^t \bar{y}}{\gamma}$$

where $\bar{x}$ means the negation of x and γ is the contribution factor and usually $\gamma > 1$. This definition has the highest accuracy in our *OCR* system among known definitions and it performs best when $gamma = 1.9$. The difference among these definitions can be explained by its weight in case that features are binary. The *Minkowski* family definition including *Manhattan* distance gives the same weight for the case where both patterns have the feature and the case where both patterns do not have the feature. The normalized inner product computes the credit for the case where both patterns have the feature but also reflects the other case as it is normalized. The third definition gives a full credit for the case where both patterns have features and also allows to control the credit for the other case. Section 4 discusses the third definition of similarity and *OCR*.

We consider the case where a threshold value, t, is known. The *KNN with threshold* problem is that we reject a query vector if the nearest neighbor is not close enough, meaning that the distance exceeds t. Let $C(x)$ denote the class of the vector x and n is the number of template vectors in the reference set, R.

Definition 6.1.4 k-nearest neighbor with threshold

Input: *A query vector q, a reference set , $R = \{r_1, r_2, \cdots, r_n\}$ and a threshold value, t.*

Output:

$$C(q) = \begin{cases} \textit{on the votes of } k \textit{ -NN} & : \ |M| \geq k \\ \textit{on the votes of } |M| \textit{ -NN} & : \ 0 < |M| < k \\ \textit{Rejected} & : \ M = \emptyset \end{cases}$$

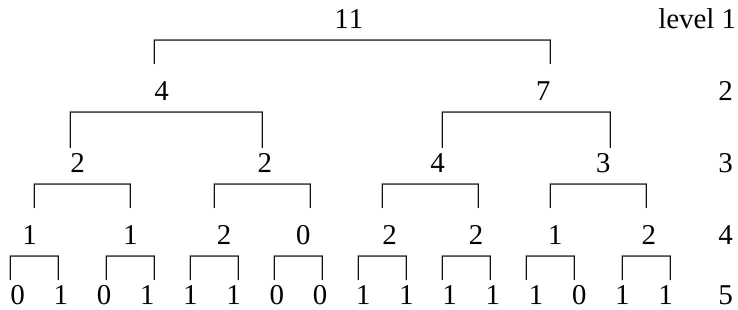


Figure 6.1: *A sample Additive Binary Tree: the value at each node is the sum of the values of its children nodes.*

Let M be a set of matches whose dissimilarity value is less than or equal to t or whose similarity value is greater than or equal to t. In the K-nearest neighbor with threshold model, the classification decision is made by up to k nearest neighbors because not enough matches may be in the set M. This model is frequently used in many applications considering the fact that the higher reject rate usually lessens the error rate.

We introduce an additive binary tree data structure, ABT in short. It is a binary tree whose nodes are sums of its direct children. Consider the example in Figure 6.1. The root is $\sum_{i=1}^{d} x_i$. Nodes at the second level are $\sum_{i=1}^{d/2} x_i$ and $\sum_{i=d/2+1}^{d} x_i$ from left to right. Leaves are $x_1, x_2, \cdots, x_d$. Let $B(x)$ be a list of nodes, $\{B_1(x), B_2(x), \cdots, B_{2d-1}(x)\}$ in breadth first order.

Definition 6.1.5 Additive Binary Tree

$$\textit{There are } 2^{l-1} \textit{number of nodes at level,} l. \tag{6.1}$$

$$\textit{Total number of nodes} = 2d - 1. \tag{6.2}$$

$$\textit{The depth of } ABT = \log d. \tag{6.3}$$

$$B_i = B_{2i} + B_{2i+1}. \tag{6.4}$$

The structure is named "additive" because additive informations produced by (4) are appended in every vector. Each element in a vector lies in the leaf level from left to right correspondingly. If vectors are binary, the root is the frequency of 1's and each node is the frequency of 1's in sub-part of the vector.

6.2 Nearest Neighbor Search using $\boldsymbol{ABT}$ in City block distance measure

6.2.1 Algorithm

There are three phases: the pre-processing, candidate selection and verification, and voting for classification phases. During the pre-processing phase, the ABT's for all templates are built. Add a pair of elements and store it in their parent node. The following pseudo code builds an ABT for a single pattern.

Algorithm 5 Build ABT (x)

begin

    for every level $l = \log d$ *down to* 1

        for $j = 2^{l-1}$ *to* $2^{2l} - 1$

            if $l =$ leaf

                $B_j(x) = x_{j-2^{l-1}}$

            else

                $B_j(x) = B_{2j+1} + B_{2j+2}$

end

Building an ABT for one template takes $O(d)$ and thus, total pre-processing takes $O(nd)$ to build ABTs for all templates in the reference set. The space required is also $O(nd)$ which is at most twice bigger than the size of the reference set.

In the second phase, we search for matches before choosing top k similar templates for votes. The object of the candidate selection and verification procedure is quickly to find the smallest subset M of the reference set which contains top or up to k templates. Consider the following pseudo code for finding matches.

Algorithm 6 Candidate Selection and Verification

begin
    Build ABT(q)
    for every template x_i
        for every level $l = 1$ *to* $\log d$
            if $\sum_{j=2^{l-1}}^{2^{2l}-1} |B_j(x_i) - B_j(q)| > t$
                break
            else if l = leaf
              Verify the Match
                if verified, update t
end

The inner loop computes the sum of absolute differences of every node in level, l. The sequential decision technique may be applied to this step; we may not have to compare all nodes in the level because the value may exceed the threshold before all nodes in a specific node are examined. We use the parent level operations as filtration functions. Those candidate templates survived from the parent level are only considered in the next level. After finding all matches, search for top or k-nearest neighbors only in the set, M.

In the second last line of Algorithm 6, we would like to update the threshold if k number of matches are found and maximum distance is less than t. This is a critical step because the lower the t is, the more filtration occurs. Also, if a threshold value

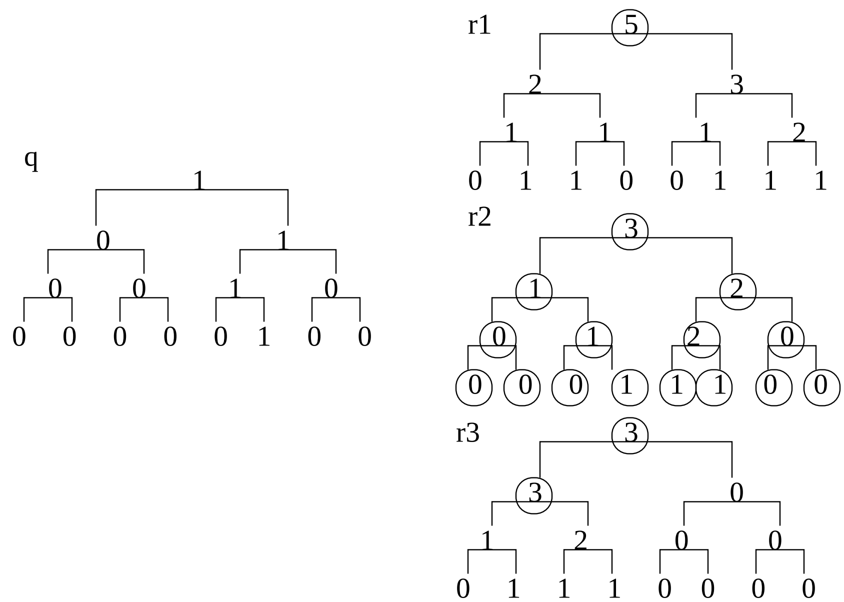


Figure 6.2: *Sample ABTs.*

is not given at the start, then it is assigned at the point right after k templates are examined and updated if a less value is found later on.

6.2.2 Example

Consider a sample reference set with $n = 3, d = 8$ and $t = 2$,

$$R = \begin{cases} r_1 &= \underline{0110011}1 \\ r_2 &= \underline{00011100} \\ r_3 &= \underline{0111}0000 \end{cases}$$

$$q = 00000100$$

When a brute force method is performed, 24 number of comparisons take place. The underlined parts of each reference vectors are those contributing the comparisons when the sequential decision technique is used; there are 19 number of comparisons. Figure 6.2 shows *ABTs* for the query vector and all templates. First, the root level is compared. r_2 and r_3 are considered as candidates. r_1 is discarded because $(|B_1(r_1) - B_1(q)| = 4) > (t = 2)$. Next, the second level of trees are compared. For r_2,

$|B_2(r_2) - B_2(q)| + |B_3(r_2) - B_3(q)| = 2$ while that of r_3 is 4. But only the $B_2(r_2)$ is computed because $|B_2(r_2) - B_2(q)|$ already exceeds the threshold. Hence, only r_2 is considered as a candidate. At the level 3, r_2 still is a candidate and at the leaf level, it is verified as the nearest neighbor to q. $|R| = 3, |L_1| = 2, |L_2| = 1, |L_3| = 1, |M| = 1$ The number of comparisons occurred is 18, the number of circled nodes as shown in the example in Figure 6.2.

6.2.3 Correctness

In this section, we show that the suggested algorithm correctly finds all matches. Let f_M and f_l be functions for matching and for selecting candidates at the level, l: $f_M(x,q) = \sum_{i=1}^{m} |x_i - q_i|$ and $f_l(x,q) = \sum_{i=2^{l-1}}^{2^{2l}-1} |B_i(x) - B_i(q)|$. Let L_l be a set of templates, x's chosen by the function $f_l(x,q) \leq t$.

Theorem 6.2.1 *A set of matches is always a subset of all candidate sets.*
$M = L_{\log d} \subset \cdots \subset L_{l+1} \subset L_l \subset L_{l-1} \subset \cdots \subset L_1 \subset R.$

Proof: $f_{L_1} \leq \cdots \leq f_{L_{\log d}} = f_M$. Proved by the well-known fact $|a + b + c| \leq |a|+|b|+|c|$. For example, $x = (x_1, x_2, x_3, x_4)$ and $q = (q_1, q_2, q_3, q_4)$. Then, $f_M(x,q) = |x_1-q_1|+|x_2-q_2|+|x_3-q_3|+|x_4-q_4|$ and $f_{l=1}(x,q) = |x_1+x_2+x_3+x_4-q_1-q_2-q_3-q_4|$. And $f_{l=2}(x,q) = |x_1 + x_2 - q_1 - q_2| + |x_3 + x_4 - q_3 - q_4|$. Clearly, $f_1 \leq f_2 \leq f_M$. Now, if there exists $x \in M$ meaning that $f_M \leq t$, then $f_l \leq t$ and $x \in L_l$. Therefore, $M \subset L_{l+1} \subset L_l \subset L_{l-1} \subset L_1 \subset R$ as shown in the Figure 6.3. ∎

The theorem means that all patterns whose distance is within the threshold are guaranteed in all candidate sets of every level of filtration. Therefore, this method is a sort of dynamic prototype reduction method but there is no loss of accuracy at all. It is as accurate as the brute force search method but much faster.

The extra space required for ABT is $d - 1$ per vector which is the number of inner nodes by definition. Therefore, the total space is $O(nd)$. Selecting Candidate

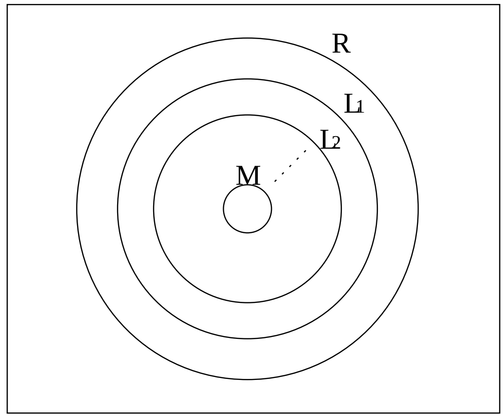


Figure 6.3: *a set of matches in candidate sets.*

process is made in the fewer computational time than that of the verification process. Number of comparisons needed at the level, l, is the number of nodes at the level which is 2^{l-1}. That of the top root level, for example, is 1 while that of the leave level is m. The number of operations is multiplied by 2 in every filtration process. The number of candidates is, on the other hand, reduced in every filtration process.

6.2.4 Simulated experiment

We present simulation results which demonstrate the performance of the new algorithm on 4 different reference sets with different feature numbers, $d = \{8, 16, 32, 64\}$. The size of the reference set, $|R| = 10,000$ for each reference set. Each feature in vectors is generated by a random function which returns a number between 0 and 99. Possible $\min(D(x,q)) = 0$ which is an exact match and possible $\max(D(x,q)) = 6336$ when $d = 64$. Corresponding test sets, Q's with $d = \{8, 16, 32, 64\}$, are prepared as well. The size of each query set is $|Q| = 10,000$. The experiment is performed on one of the fastest *Sun Microsystems* in *CEDAR*. The machine has a *300MHz UltraSparc2 CPU, SunOS 5.6, 588 MIPS*, and *256 Memory*. Figure 6.4 shows the performance of a naïve method and methods using ABT structure with several different threshold values.

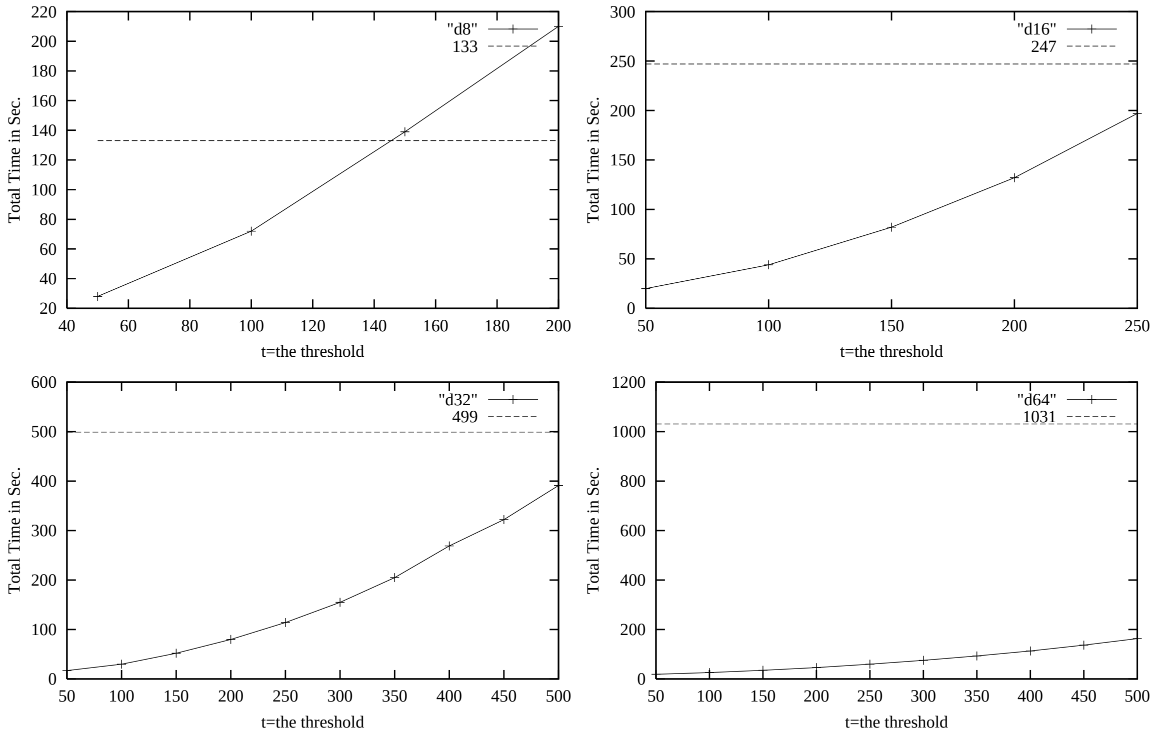


Figure 6.4: *Cumulated elapsed time for 10,000 queries over 10,000 templates.*

Observation 6.2.2 *The smaller threshold value, t, the faster ABT algorithm runs but the reject rate increases.*

A smaller threshold means fewer candidates and the algorithm runs faster. As the threshold increases, on the contrary, candidates abound. Too small threshold, on the other hand, might reject most of inputs.

Observation 6.2.3 *The larger dimension d is, the better performance the ABT algorithm achieves the better execution time performance.*

It is necessary to compare the results of d's with the same ratio of t. Consider the case that $d = 64, t = 300$, then the ABT runs 13.8 times faster than the naive method. On the other hands, when $d = 32, t = 150$, it is only 9.6 times faster.

Filtering from top to all the way down to the level, $l - 1$ may not give always the most expeditious running time. It would be wise to stop filtering and jump to the verification stage if the number of candidates is small. Table 6.1 indicates this behavior.

Method	time in millisec.
Naïve	103.3
Sequential Decision	24.1
Filtration $l = 1$	16.7
Filtration $l = 2$	12.4
Filtration $l = 3$	7.2
Filtration $l = 4$	9.5

Table 6.1: *Comparisons for methods with different filtering level.*

Observation 6.2.4 *There exists a level, $\hat{l}$ which results the minimum elapsed time.*

The deeper the level is, the more computations which is the number of nodes at the level, are required. $\hat{l}$ is the starting point that $|L_{\hat{l}}| \times 2^{\hat{l}} \leq |L_{\hat{l}+1}| \times 2^{\hat{l}+1}$ which means that we gain no advantage of filtration from this level. Therefore, finding $\hat{l}$ is important because it gives not only the minimum elapsed time, but also less required space for ABT data structure.

The sequential decision method in Table 6.1 is that all distance calculations for templates are carried out but terminated if they exceed the threshold before all feature differences are cumulated.

Observation 6.2.5 *The* ABT *technique is superior to the sequential decision technique on average.*

In the worst case when every reference is a match, both sequential decision and ABT would not give any speed up. Technique using ABT would be even twice slower than the naive method in case of a full ABT.

6.2.5 Auxiliary

Lookup Table

Suppose $d = 512$ and all features are binary, then a vector is stored as an array of 16 unsigned integer values. We build a lookup-table for counts of 1's for all 16 bit combination. The size of lookup-table is $262K$ bytes. This look-up table for unsigned integers is built at preprocessing stage. To find $D[x, y]$, we perform table look-ups for high and low 16 bits of each unsigned integer ($= x \otimes y$). There are total 32 table look-ups to determine the distance. Where vectors are stored as unsigned integers, we consider counts of 1's of each unsigned integer value as leaves of ABT. Lookup table enables significant speedup over the method counting the number of bits without the table.

Ordered List

The simulated experiment tells that smaller the threshold, the faster algorithm runs (see Observation 6.2.2). The threshold value is dynamically changed as more templates are examined. It would be desirable if the smaller threshold value is assigned early. Therefore, we would like to order the templates so that the less threshold value is set early in the search.

Fact 6.2.6 *The closer in frequency two vectors are, the smaller in absolute difference distance they tend to be.*

If we order templates by its frequency and search from a template whose difference of frequency is smallest, then the low threshold value might be set in the earlier search stage resulting in a great speed up. First, order the reference set by B_1. For each bin, the average number of templates is n/m. Next, order each bin by B_2. During the search stage, start searching from the template, x whose $\min(|B_1(x) - B_1(q)|)$ and then $\min(|B_2(x) - B_2(q)|)$. As a result of ordering, not only we find the low threshold value quickly, but also, our search space is reduced down to the size of L_2 in Figure 6.3.

Selection

We have seen that ABT facilitates filtrating some reference templates from consideration by using the lower bounds at each level. In addition to the lower bounds, ABT also provides upper bounds. If an upper bound is less than or equal to the threshold, the match is verified without further calculation. This selection technique is useful in a query such that we would like to find all matches to the query vector within a threshold. Consider w-ary vectors x and y where each element can have a value between 0 and $w-1$.

Lemma 6.2.7 *$D[x,y]$ has upper and lower bounds at the root level: $|B_1(x)-B_1(y)| \leq D[x,y] \leq \min(B_1(x)+B_1(y),(wd-B_1(x))+(wd-B_1(y)))$.*

Proof: The lower bound is shown in Theorem 6.2.1. Similarly, $\sum_{i=1}^{d}|x_i-y_i| \leq \sum_{i=1}^{d}x_i+\sum_{i=1}^{d}y_i$ by the fact of $|a+b| \leq |a|+|b|$. Now, $D[x,y]=\sum_{i=1}^{d}|x_i-y_i| = \sum_{i=1}^{d}|(w-x_i)-(w-y_i)|$. This is the absolute difference of two inverse vectors and it is the same as the absolute difference of the original vectors. Clearly, $\sum_{i=1}^{d}|(w-x_i)-(w-y_i)| \leq \sum_{i=1}^{d}(w-x_i)+\sum_{i=1}^{d}(w-y_i)$ Therefore, $D[x,y] \leq \min(\sum_{i=1}^{d}x_i+\sum_{i=1}^{d}y_i, \sum_{i=1}^{d}(w-x_i)+\sum_{i=1}^{d}(w-y_i))$ ∎

Consider two vectors, x,y where they are binary, $w=1$.

$$\begin{cases} x = 00011111 & : \quad B_1(x)=5 \\ y = 01001011 & : \quad B_1(y)=4 \end{cases}$$

According to the lemma 2.9, we achieve $1 \leq D[x,y] \leq 8$ immediately.

Theorem 6.2.8 $\sum_{i=2^{l-1}}^{2^{2l}-1}(B_i(x)-B_i(y)) \leq D[x,y] \leq \min(\sum_{i=2^{l-1}}^{2^{2l}-1}(B_i(x)+B_i(y)),$ $\sum_{i=2^{l-1}}^{2^{2l}-1}((\frac{wd}{l}-B_i(x))+(\frac{wd}{l}-B_i(y))))$

Proof: Again, the lower bounds are proved in Theorem 3.1. For the upper bounds, divide vectors into 2^{l-1} number of sub-vectors. For each sub-vectors, Lemma 5.2 holds. Thus, $D[x,y] \leq \min(\sum_{i=2^{l-1}}^{2^{2l}-1}(B_i(x)+B_i(y)), \sum_{i=2^{l-1}}^{2^{2l}-1}((\frac{wd}{l}-B_i(x))+(\frac{wd}{l}-B_i(y)))$ ▮

For the above example, we have the upper and lower bounds at the second level of ABT: $1 \leq D[x,y] \leq 3$.

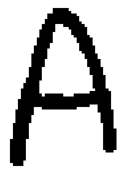
6.3 Using *ABT* for GSC classifier

In this section. we report implementing the suggested technique in the field of *OCR*. First, we give a brief introduction to the GSC classifier and then explains the implementation of the algorithm with the proof of correctness. Finally, we give the experimental results on the isolated mixed hand-printed/cursive character recognition problem.

6.3.1 GSC classifier

Among many classifiers in *OCR*, the *Gradient, Structural, and Concavity classifier*, simply known as *GSC classifier*, has 98% accuracy on 24,000 handwritten digits taken from various databases [42]. GSC classifier is based on the philosophy that feature sets can be designed to extract certain types of information from the image [41, 42, 95, 96]. These types are gradient, structural, and concavity informations. Gradient features use the stroke shapes on a small scale. Next, structural features are based on the stroke trajectories on the intermediate scale. Finally, concavity features use stroke relationship at long distances. Gradient, Structural, and Concavity has 192, 192, and 128 feature vectors correspondingly. In all, there are 512 number of features in a vector. Figure 6.5 illustrates a sample GSC binary feature vector.

The *K-nearest neighbor* approach is often used in the *GSC classification.* It selects the top k similar templates in the training set and returns its class based on the vote



Gradient	:	0000000000110000000011000011100000001110000000110000001100010000
(192$bits$)		0000110000000000000111001100011111100001111000000010010100000100
		0111001111110011111100000100000100000000000000000001000001001000
Structural	:	0000000000000000000011000011100010000100001000000100000000000001
(192$bits$)		0010100000000001100001010011000011000000000000010010001100110000
		0000000000110010100000000000000001100000000000000000000000010000
Concavity	:	1111011010011111011001100000011011110110100110010000011000001110
(128$bits$)		0000000000000000000000000000000000000000000111111000000000000000

Figure 6.5: *A sample character and its GSC feature vector.*

of these k template vectors. The definition of this similarity, $S[x, y]$, currently used in the GSC classifier follows:

$$S[x, y] = \sum_{i=0}^{n} S[x_i, y_y]$$

$$S[x_i, y_i] = \begin{cases} 1 & : \quad x_i = y_i = 1 \\ 1/\gamma & : \quad x_i = y_i = 0 \\ 0 & : \quad \text{otherwise} \end{cases}$$

When both x_i and y_i are 1's, it is the case that we find the features that we want to find. It is denoted as $S_{11}[x, y] = x^t y$. When both x_i and y_i are 0's, it is the case that we do not find the features that we do not want to find. It is denoted as $S_{00}[x, y] = \bar{x}^t \bar{y}$. It would be reasonable to rely more upon S_{11} than S_{00}. The definition of the similarity measure becomes :

$$S[x, y] = S_{11}[x, y] + \frac{S_{00}[x, y]}{\gamma}$$

γ is the contribution factor and usually $\gamma \geq 1$. The recent experiment shows that $\gamma = 1.9$ gives the best performance. Note that when $\gamma = 1$, it is the inverse of the city block distance between two vectors.

6.3.2 Algorithm for GSC classifier

The preprocessing which builds the ABT's for all templates is the same as Algorithm 5. There is a slight difference in candidate selection and verification procedure as its definition of similarity differs. Let $B_i(x)$ and $W_i(x)$ be the number of 1's and 0's at the i's node of vector x respectively.

Algorithm 7 Candidate Selection and Verification

1 begin
2 Build ABT(q)
3 *for every* template x_i
4 *for every* level $l = 1$ *to* $\log d$
5 *if* $\sum_{i=2^{l-1}}^{2^{2l}-1} (\min(B_i(x), B_i(q)) + \frac{\min(W_i(x), W_i(q))}{\gamma}) \geq t$
6 *break*
7 *else if* $l =$ leaf
8 Verify the Match
9 *if* verified, update t
10 end

Only the line 5 is different from Algorithm 6.

6.3.3 Correctness

Fact 6.3.1 $B_i(x) = \frac{d}{2^{l-1}} - W_i(x)$ *where* l *is the level of* i*'s node.*

Lemma 6.3.2 *If* $\min(W_i(x), W_i(y)) = W_i(x)$*, then* $\min(B_i(x), B_i(y)) = B_i(y)$ *and vice versa.*

Proof: $B_i(x) = \frac{d}{2^{l-1}} - W_i(x)$ and $B_i(y) = \frac{d}{2^{l-1}} - W_i(y)$ by Fact 6.3.1. Now if $B_i(x) \leq B_i(y)$, then $\frac{d}{2^{l-1}} - W_i(x) \leq \frac{d}{2^{l-1}} - W_i(y)$. By rearranging the formula, we get

$W_i(x) \geq W_i(y)$. ▮

Lemma 6.3.3 $S[x, y] \leq \min(B_1(x), B_1(y)) + \frac{\min(W_1(x), W_1(y))}{\gamma}$.

Proof: When only frequency values are available, the maximum value $S_{11}(x, y)$ is when 1's in $B_1(x)$ and $B_1(y)$ are aligned together. Thus, $S_{11}(x, y)$ cannot exceed $\min(B_1(x), B_1(y))$. The same for $S_{00}(x, y)$. ▮

The maximum similarity value that two vectors x and y can have at the root level of ABT is $\min(B_1(x), B_1(y)) + \frac{\min(W_1(x), W_1(y))}{\gamma}$. Suppose $\gamma = 2$. In the examples in Figure 6.2, $f_1(q, r_1) = \min(1, 5) + \frac{\min(7,3)}{2}$ which is 2.5. Similarly, $f_1(q, r_2) = 3.5, f_1(q, r_3) = 3.5$. These values are always larger than or equal to $S[x, y]$'s defined in GSC classifier. When there is an exact match meaning that $x = y$, $S[x, y] = B_1(x) + \frac{W_1(x)}{\gamma}$. Therefore, the maximum similarity value varies from vector to vector by their frequencies.

Let f_l be a function for selecting candidates at the level, l:
$f_l(x, q) = \sum_{i=2^{l-1}}^{2^{2l}-1} (\min(B_i(x), B_i(q)) + \frac{\min(W_i(x), W_i(q))}{2})$. Let L_l be a set of templates, x's chosen by the function $f_l(x, q) \geq t$.

Corollary 6.3.4 *All members in M are guaranteed to be in all candidate sets.*

$$M = L_{\log m} \subset \cdots \subset L_{l+1} \subset L_l \subset L_{l-1} \subset \cdots \subset L_1 \subset R$$

Proof: Let $B(x)$ and $W(x)$ be the frequency of 1's and 0's of one internal node. Let $B_l(x), W_l(x)$ and $B_r(x), W_r(x)$ be those of the left and right children of the node. By the definition, $B(x) = B_l(x) + B_r(x)$ and $W(x) = W_l(x) + W_r(x)$. Consider two vectors x and y. Suppose $B(x) = \min(B(x), B(y))$, then the maximum possible value at the internal node can have become

$$B(x) + \frac{W(y)}{\gamma} = B_l(x) + \frac{W_l(y)}{\gamma} + B_r(x) + \frac{W_r(y)}{\gamma}$$

Now at its children's nodes, there are three possible cases and one impossible case. First, if $B_l(x) = \min(B_l(x), B_l(y))$ and $B_r(x) = \min(B_r(x), B_r(y))$, then the maximum value at the both children nodes can have is the same as that of their parent's node by definition :

$$B_l(x) + B_r(x) \leq B_l(y) + B_r(y)$$

$$W_l(x) + W_r(x) \geq W_l(y) + W_r(y)$$

Next, if $B_l(x) = \min(B_l(x), B_l(y))$ but $B_r(y) = \min(B_r(x), B_r(y))$, then it is smaller than or equal to that of their parent's node.

$$B_l(x) + \frac{W_l(y)}{\gamma} + B_r(y) + \frac{W_r(x)}{\gamma} \leq B_l(x) + \frac{W_l(y)}{\gamma} + B_r(x) + \frac{W_r(y)}{\gamma}$$

When $B_l(y) = \min(B_l(x), B_l(y))$ but $B_r(x) = \min(B_r(x), B_r(y))$, it is also smaller than or equal to that of their parent's node. It is impossible by definition if $B_l(y) = \min(B_l(x), B_l(y))$ but $B_r(y) = \min(B_r(x), B_r(y))$. Therefore, the upper bound of a parent level is always greater than or equal to that of its children's nodes. This is true for all internal nodes. The upper bound at a certain level is the sum of all upper bounds at all nodes in that level. Let f_l be the upper bound function at a level, l, then

$$f_{L_1} \geq \cdots \geq f_{L_{\log m}} = f_M.$$

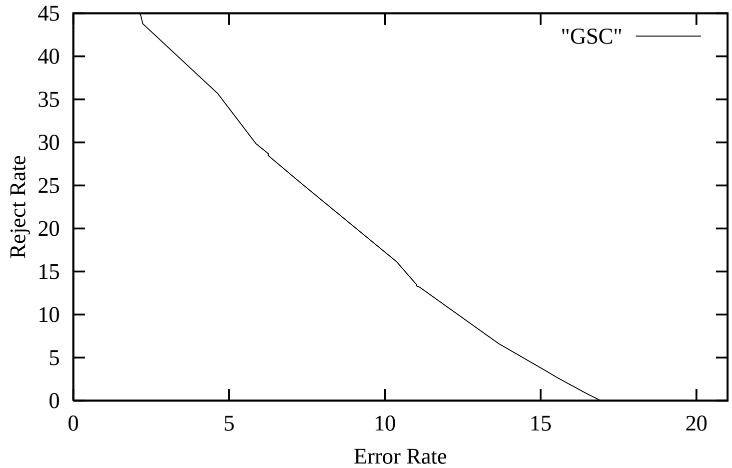
Therefore, $M = L_{\log m} \subset \cdots \subset L_{l+1} \subset L_l \subset L_{l-1} \subset \cdots \subset L_1 \subset R.$ ▮

This corollary is the sine qua non which guarantees the correctness and speedup of Algorithm 7.

The following lemma further accelerates the search process, which we state it without a proof.

Lemma 6.3.5 $S_{00}(x, y) = d - B_1(x) - B_1(y) + S_{11}(x, y).$

This means that once we compute the similarity for $S_{11}(x, y)$, $S_{00}(x, y)$ is computed in constant time.



$$\text{Error rate} = \frac{\text{Number of Errors}}{\text{Number of Accepted Queries}} \times 100$$

$$\text{Reject rate} = \frac{\text{Number of Rejections}}{\text{Number of Total Queries}} \times 100$$

Figure 6.6: *Error vs. Reject graph for GSC classifier.*

6.3.4 Experiment

The experiment on a GSC classifier using ABT is accomplished on the same specifications of hardware and O.S. as used in the previous section 6.2.4. Before embarking on the results, it is important to discuss the relationship between error and reject rates of a recognizer. As shown in Figure 6.6, the error versus reject percentages of recognition graph is a great way to evaluate the performance of OCR systems. A good recognizer must be as close to the axes as possible. Typically, the higher reject rate, the lower error rate. The threshold value must be made depending on the costs of rejects and errors. As the threshold value increases, the reject rate also increases and the error rate decreases.

We use two sets of isolated mixed hand-printed/cursive character images. The first set is the reference set whose size is 21800 and approximately 800 per character (A-Z). This set is not case sensitive. The other set is a test set whose size is 1681.

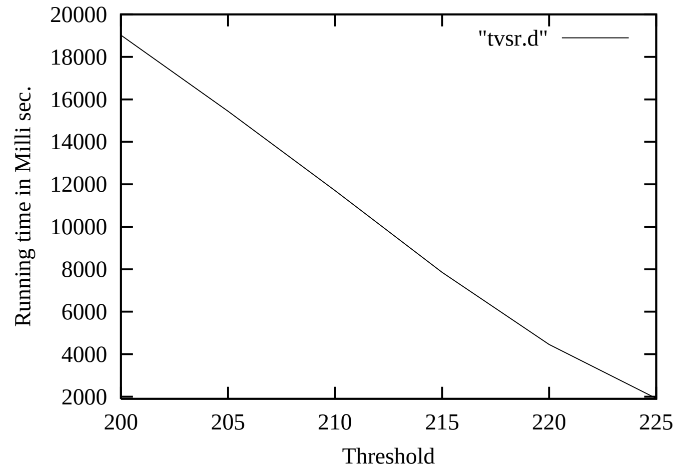


Figure 6.7: *Threshold vs. running time.*

While the average running time of the brute force method is 39.470 millisecond, one using ABT is 19.714 millisecond where the depth of tree is 2 and $t = 0$.

Observation 6.3.6 *The higher threshold value the classifier has, the faster it runs.*

Consider Figure 6.7 The higher threshold value means the more rejects which save running time. Hence, the importance of a threshold is twofold. First, it enables to control the error and reject rates. It also gives speed up.

6.4 Finale

Nearest Neighbor searching is an extremely well studied area and a couple of techniques to speed up the *OCR* system were already developed and published. They are prototype thinning and clustering techniques which cause a little degradation in performance. Nevertheless, we studied it again because even a slight degradation in performance is often too costly in real *OCR* applications. In this paper, a new fast nearest neighbor search algorithm with no degradation is proposed. Filtration with a threshold is the key idea for expedition. To perform this, we introduced *ABT* structure. The additive information of pattern feature vectors ensures that the query processing time reduces significantly. Furthermore, the idea is effective even in combination with other techniques like prototype thinning or clustering.

There are two parameters that can affect the speed of search : i) depth of ABT and ii) the number of branches. As stated in Observation 6.2.4, the depth of ABT influences the speed significantly. We introduced the additive binary tree, however, the tree can be an arbitrary branch tree. Additive N-ary Tree might perform better than the binary tree.

While one major achievement is the improvement of OCR system using GSC feature sets in terms of speed, the technique can extend to arbitrary applications. Moreover, the idea of filtration using ABT can extend to other definitions with a little embellishment. Consider the normalized inner product defined in Definition 6.1.2. The maximum value for this definition is 1 when there is an exact match. We exclude the exceptional case where all features are 0 as it makes the denominator be 0. A filtration occurs in case that

$$\sum_{i=2^{l-1}}^{2^{2l}-1} \frac{\min(B_i(x), B_i(q))}{\|x\|\|q\|} \geq t$$

Again, consider the example in Figure 6.2. Suppose the threshold is 0.5. At the root level filtration, r_1 is filtrated because the upper bound is 0.447.

Chapter 7

Data Mining for Sub-category Discrimination Analysis

The sub-category classification problem is that of discriminating a pattern to all sub-categories. Not surprisingly, sub-category classification performance estimates are useful information to mine as many researchers are interested in any trend of pattern in specific sub-category. This chapter presents a data mining technique to mine a database consisting of experimental and observational unit variables. Experimental unit variables are those attributes which make sub-categories of the entity, e.g., patient personal information or a person's identity and observational unit variables are features observed to classify the entity, e.g., test results or handwriting styles, etc. Since there are an enormously large number of sub-categories based on the experimental unit variables, we apply the *Apriori algorithm* to select only sub-categories that have enough support among all possible ones in a given database. Those selected sub-categories are then discriminated using observational unit variables as input features to the *Artificial Neural Network (ANN)* classifier. The importance of this paper is twofold. First, we propose an algorithm that quickly selects all sub-categories that have enough both support and classification rate. Second, we successfully applied the proposed algorithm to the field of handwriting analysis. The task is to determine similarity of handwriting style of a specific group of people. Document examiners are interested in trends in the handwriting of specific groups, e.g., (i) does a male write differently from a female? (ii) can we tell the difference in handwriting of age group between 25 and 45 from others?, etc. Subgroups of white males in the age group 15-24 and white females in the age group 45-64 show 87 % correct classification performance.

7.1 Apriori for Classification

The problem of discrimination and classification is mainly concerned with differentiating between $g (g \geq 2)$ mutually exclusive populations and with classifying subjects or objects on the basis of multivariate observation [30, 36]. The mathematical problem of discrimination is to divide the observation space R^p into g mutually exclusive and exhaustive regions $R_1, \cdots, R_g$, so that if an observation x falls in R_i, then it is classified as belonging to the ith population. The *sub-category classification problem* involves defining mutually exclusive populations according to their attributes and classifying an unseen instance into its sub-categories using multiple features. We call the attributes contributing defining the sub-categories as *experimental unit variables* and observed features as *observational unit variables*.

For example of classifying a person, possible attributes (experimental unit variables) are gender, ethnicity, age, etc. One can observe various features such as a person's appearances (observational unit variables) to classify him or her to the proper sub-categories. Consider a subject with three experimental unit variables where each variable has two values: $V_1 = \{v_{1,1}, v_{1,2}\}, V_2 = \{v_{2,1}, v_{2,2}\}, V_3 = \{v_{3,1}, v_{3,2}\}$. As shown in Figure 7.1, there are seven possible sub-category classification problems. Subsets of p_4, p_5, p_6, and p_7 also form sub-category classification problem, e.g., $p_i \rightarrow \{(v_{1,1}, v_{2,2}, v_{3,1}) : (v_{1,1}, v_{2,2}, v_{3,2})\}$, etc. Hence, there are combinatorially large number of sub-category classification problems that one database can create. Enumerating and classifying them all is inefficient.

For this reason, we present an efficient technique to mine a database consisting of both experimental and observational unit variables. The information extracted from

	V_{11}		V_{12}	
	V_{31}	V_{32}	V_{31}	V_{32}
V_{21}				
V_{22}				

$$
\begin{aligned}
p_1 &\rightarrow \{v_{1,1} : v_{1,2}\}\\
p_2 &\rightarrow \{v_{2,1} : v_{2,2}\}\\
p_3 &\rightarrow \{v_{3,1} : v_{3,2}\}\\
p_4 &\rightarrow \{(v_{1,1}, v_{2,1}) : (v_{1,1}, v_{2,2}) : (v_{1,2}, v_{2,1}) : (v_{1,2}, v_{2,2})\}\\
p_5 &\rightarrow \{(v_{1,1}, v_{3,1}) : (v_{1,1}, v_{3,2}) : (v_{1,2}, v_{3,1}) : (v_{1,2}, v_{3,2})\}\\
p_6 &\rightarrow \{(v_{2,1}, v_{3,1}) : (v_{2,1}, v_{3,2}) : (v_{2,2}, v_{3,1}) : (v_{2,2}, v_{3,2})\}\\
p_7 &\rightarrow \{(v_{1,1}, v_{2,1}, v_{3,1}) : (v_{1,1}, v_{2,1}, v_{3,2}) : (v_{1,1}, v_{2,2}, v_{3,1}) : (v_{1,1}, v_{2,2}, v_{3,2}) :\\
&\quad\ (v_{2,1}, v_{2,1}, v_{3,1}) : (v_{2,1}, v_{2,1}, v_{3,2}) : (v_{2,1}, v_{2,2}, v_{3,1}) : (v_{2,1}, v_{2,2}, v_{3,2})\}
\end{aligned}
$$

Figure 7.1: Sample sub-category classification problems

the technique is the classification ratio of each sub-category. Not surprisingly, sub-category discrimination analysis is valuable information to be mined because many researchers are interested in any trend in specific subgroup.

In order to answer whether one can build a machine that can classify an unseen instance into its sub-category, each class (subgroup) must have a substantial number of instances for the sake of valid statistical inference. This Sine-qua-non is called *support*. We apply the *Apriori* algorithm to select all sub-categories that have enough support among all possible ones in a given database. Those selected sub-categories are then discriminated using the *Artificial Neural Network (ANN)* classifier. Finally, the performance measures for each selected sub-category problem are reported as final outputs. We use the *Artificial Neural Network (ANN)* because it is equivalent to multivariate statistical analysis. There is a wealth of literature regarding a close relationship between neural networks and the techniques of statistical analysis, especially multivariate statistical analysis, which involves many variables [29, 36].

Apriori Algorithm was originally designed for the purpose of efficient association rule mining by *Agrawal et al* [3, 2]. The concept of association rules was introduced in 1993 [1] and many researchers have endeavored to improve the performance of algorithms that discover the association rules in large datasets. The *Apriori algorithm* is an efficient association discovery algorithm that filters item sets by incorporating item constraints (support). We apply this filtration by support approach to mine the subgroup classification information in a large database with one table with experimental units (writer information) and the other table with observational units (document image features).

As an illustrative example, we consider a CEDAR letter database [26] consisting of $1,000$ writer data with six writer attributes and features extracted from a handwriting sample to determine similarity of a specific group of people. Document examiners are interested in any trend of handwriting in specific group, e.g., *i)* does male write differently from female? *ii)* can we tell the difference in handwriting of age group between 25 and 45 from others?, etc.

7.2 Database

The CEDAR letter database is a database consisting of $3,000$ handwritten document images written by $1,000$ subjects representative of the US population by stratification and proportional allocation [26]. Each individual provided three handwriting samples, written with black ballpoint pens on plain white sheets by copying text that has all alphabets in every position of a word, as well as numerals. The database was originally created for the purpose of the *writer identification* study [25, 21].

We stratified our database along six experimental unit variables: gender (G) {male, female}, handedness (H) {right, left}, age (A) {under 15, $15 \sim 24$, $25 \sim 44$, $45 \sim 64$, $65 \sim 84$, over 85}, ethnicity (E) {white, black, hispanic, asian and pacific islander, American Indian, Eskimo, Aleut}, highest level of education (D) {below highschool graduate, above}, and place of schooling (S) {USA, Foreign}. We can analyze the association between different combinations of these variables. Studying the association between only one variable is called *1-constraint* subgroup analysis (male vs. female, or black vs. white vs. hispanic), studying the association between pairs of variables is called *2-constraint* subgroup analysis (white male vs. white female), and so on. The size of 2-constraint subgroup is the multiplication of each size of 1-constraint subgroup. Figure 7.2 shows some of the possible combinations. As mentioned earlier, there can be a combinatorially large number of subgroup classification problems one can analyze. Since our database was not stratified heavily across each variable, it is necessary to identify those subgroups that have enough support or coverage.

There seem to be conflicting views about whether or not group-specific handwriting features can be attributed to the sex, age, ethnicity, or handedness of writers. Correlations between these groups and handwriting features are dealt with in [57]. For instance, while tremors caused by aging may have a bearing on handwriting, the direction of horizontal strokes, amount of pressure exerted on up-strokes or down-strokes, consistency in letter slopes, slope of writing, direction of curves may be affected by handedness. Ethnicity may also affect handwriting. Hispanic writers have a tendency to ornateness in the formation of capital letters, slope of writing in France, United Kingdom and India tends to be vertical or even slightly backhand and is clearly forehand in Germany.

7.3 Algorithm

The algorithm has two filtering stages with two user defined threshold values. Figure 7.3 illustrates the algorithm figuratively. To find the subgroup classification problem that is statistically valid (having enough supporting instances), we show the efficient *Apriori* algorithm without aggregation. In this algorithm, the subgroup with

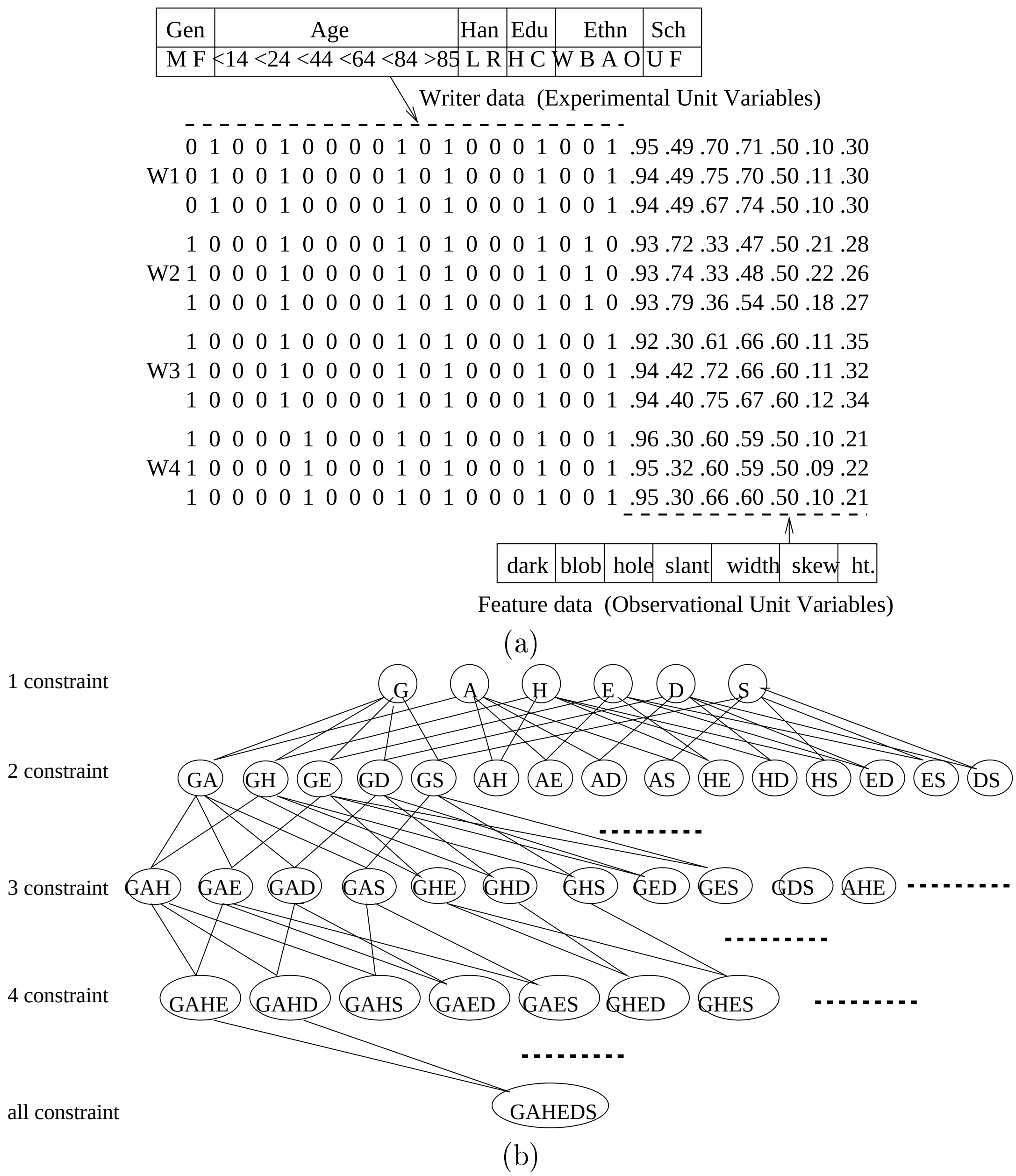


Figure 7.2: (a) Sample Entries of CEDAR letter database and (b) List of sub-categories where G, A, H, E, D, and S correspond to Gender, Age, Handedness, Ethnicity, Degree of education, and place of Schooling, respectively.

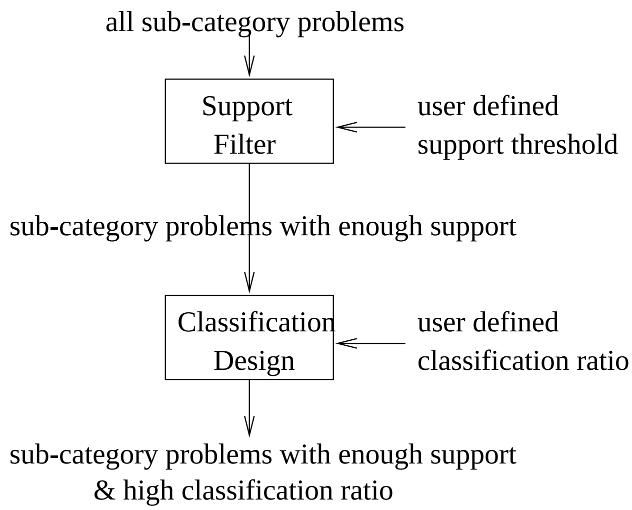


Figure 7.3: Apriori algorithm overview.

aggregation such as {white vs. (black $\cup$ hispanic)} is not considered. First, for each attribute value of every variable, count the occurrences to find whether the sum exceeds the user defined minimum support. If so, add it to the *1-constraint* output list: {(Male: Female), $(15 \sim 24 : 25 \sim 44 : 45 \sim 64)$, (white: black: hispanic)}. Second, from the *1-constraint* output list, generate the *2-constraint* output list. They are {(Male, $15 \sim 24$: Female, $15 \sim 24$: Male, $25 \sim 44$: Female, $25 \sim 44$), (Male, white: Female, white), (white, $15 \sim 24$: white, $25 \sim 44$)}. Next, from the *2-constraint* output list, generate the *3-constraint* output list, it is {(white, male, $15 \sim 24$: white female, $15 \sim 24$)}. Repeat generating the higher constraint list until it is empty or it reaches the all-constraint list. It is efficient because we do not consider all possible combinations but generate the higher constraint list from only those elements in the lower constraint list.

7.4 Experimental Results

Once we identify those subgroup classification problems that have enough support (over 200 samples), we train an ANN (Artificial Neural Network) to evaluate the accuracy with which we can classify these subgroups as shown in Figure 7.4. The

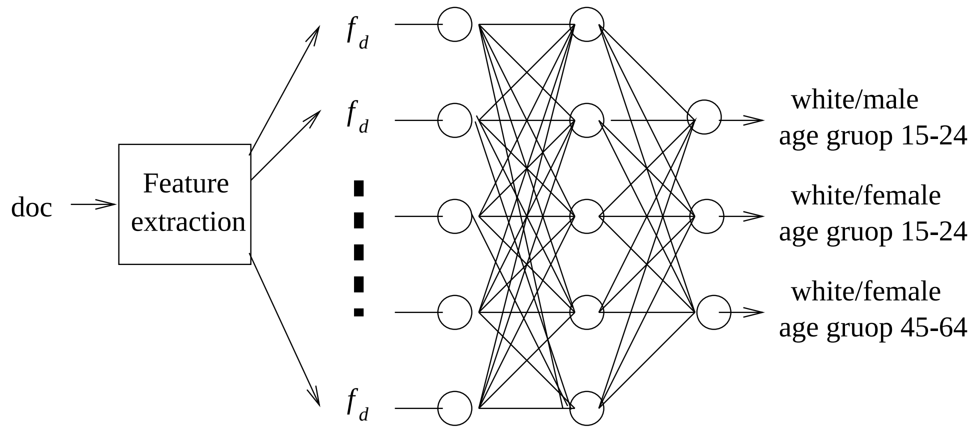


Figure 7.4: Artificial Neural Network classifier for writer subgroup classification problem.

samples of interest were divided into 4 groups - one for training, one for validation, and two test sets. An ANN was trained using the feature values of samples. Examples of *1-constraint* subgroups and their classification rates are Male vs. Female (70.2%), black vs. white (64.5%), left vs. right (59.5%), below vs. above high-school graduate (61%). The 2-constraint subgroup of white males, and white females was studied and 68% (type-I error: 24%, type-II error: 40.5%) performance was obtained. The 3-constraint subgroup of white males in the age group 15-24, and white females in the age group 45-64 was studied. The performance on the two tests sets was 83% (type-I error: 21%, type-II error: 12%) and 87% (type-I error: 14%, type-II error: 12%). It is observed that the more constraint, the higher classification rate.

7.5 Conclusion

In this paper, we presented a data mining technique for sub-category classification problem. *Apriori* algorithm is applied to filter the sub-category problems with insufficient support. We considered a database consisting of writer data and features obtained from a handwriting sample, statistically representative of the US population, to determine similarity of a specific group of people. Higher classification rate is achieved with higher constraint sub-categories. In this paper, the subgroup with aggregation was not considered. The algorithm can be altered to handle the subgroup with aggregation.

Chapter 8
Conclusion

In this dissertation, we considered the problem of evaluating and utilizing various distance measures in the domain of handwriting analysis. Three applications are used as the illustrative examples to appreciate the use of distance measures in handwriting analysis: *writer verification*, *on-line* and *off-line handwriting recognition*. In this chapter, we summarize the major achievements of this dissertation that contribute greatly in the field of handwriting analysis as well as pattern recognition.

8.1 Achievements

8.1.1 Individuality Validation

One of the problem encountered in this dissertation is that of validating individuality in handwriting. We showed that the multiple category classification problem can be viewed as a two-categories problem by defining the distance and taking those values as positive and negative data. This paradigm shift from the polychotomizer to the dichotomizer makes the writer identification that is a hard U.S. population multiple class problem very simple. We compared the proposed *dichotomy* model in feature distance domain with the *polychotomy* model in feature domain from the view point of tractability and accuracy. We designed an experiment to show the individuality of handwriting by collecting samples from people that is representative of the US population. Given two randomly selected handwritten documents, we can determine whether the two documents were written by the same person or not. Our performance is 97%.

One advantage of the dichotomy model working on distribution of distances is that many standard geometrical and statistical techniques can be used as the distance data is nothing but scalar values in feature distance domain whereas the feature data type varies in feature domain. Thus, it helps to overcome the non-homogeneity of features. Techniques in *pattern recognition* typically require that features be homogeneous. While it is hard to design a polychotomizer due to non-homogeneity of features, the dichotomizer simplifies the design by mapping the features to homogeneous scalar values in the distance domain.

The work reported in this paper is applicable to the area of *Forensic Document Examination*. We have shown a method to access the authorship confidence of handwritten items utilizing the *CEDAR letter* database. It is a procedure for determining whether or not two or more digitally scanned handwritten items were written by the same person. Thanks to the completeness of the *CEDAR letter* database, it is a panacea for the analysis for any handwritten item by synthesization.

8.1.2 Designing Distance Measures

Histogram

On the distance between histograms, we have criticized inadequacy of the way that existing definitions, D_1-D_6 are used for ordinal and modulo type histograms. We considered three types of histograms characterized by their measurement type: nominal, ordinal and modulo. Different algorithms are designed for each type of histograms; Eq.(3.25), Algorithm 1 and Algorithm 2, correspondingly. Their computational time complexities are $\Theta(b)$, $\Theta(b)$ and $O(b^2)$, respectively, insofar as the histograms are given. These algorithms are based on one universal concept of distance between sets that is the problem of minimum difference of pair assignments.

We introduced the problem of minimum difference of pair assignments to grasp the concept of the distance between two histograms. Extending the suggested algorithms 1 and 2 facilitates the solution to this problem in $\Theta(n+b)$ and $O(n+b^2)$ time for ordinal and modulo type univariate data, respectively, reflecting the time complexity of $\Theta(n)$ to build the histogram.

As applications, one can use the measure directly or indirectly to solve the problem of classification, clustering, indexing and retrievals, i.e., image indexing and retrieval based on grey scales or hue value histograms. We strongly believe that there are a plethora of applications in various fields.

String

In Chapter 4, we categorized strings into four types: nominal, angular, magnitude and cost-matrix. We extended the *Levenshtein edit distance* to handle the angular and linear type strings. It is good for matching stroke and contour directional sequence strings. It takes *turn* and local context into account to compute the edit distance. This technique performs better than Levenshtein edit distance with cost matrix.

We also presented string distance measures to solve writer identification, on-line and off-line character recognition. To perform this, we converted a two-dimensional image to one-dimensional strings and then we measured the edit distance between strings. The smaller the edit distance is, the more similar they look like to each other.

We used two very important features, *SDSS* and *SPSS* to solve the *writer identification* problem. It is an unique approach to represent the *pressure* of handwriting as a sequence of pressure extracted from an off-line character image. In all, the proposed semi-automatic method provides a distinction or similarity between two handwritings figuratively and numerically. It is expected to help greatly document examiners and signature verifiers to compare handwritings and signatures.

Another major contribution in on-line character recognition is to diminish the effect of unnaturally written characters. Concatenation and reverse operations are used as a pre-processing to reduce the effect.

Binary Vector

In Chapter 5, we discussed the binary vector distance and convex hull distance. It is worth mentioning again that selecting and designing a similarity measure is as important as finding significant features. Poor choice of similarity measure would result in unsatisfactory performance in recognition. In designing a similarity function, it is necessary to have a tuning set in addition to a reference set and validation set if coefficients are associated with it. The major contribution is presenting a modified similarity measure for GSC recognizer and achieving a better performance on the off-line character recognition.

Convex Hull

Determining the convex hull is a basic step in several statistical problems such as robust estimation, isotonic regression, clustering, etc [83]. We have shown another example that is *pattern classification* applied in *Writer Identification* application. A prototypical convex hull discriminant function is presented. The convex hull of the given samples is regarded as one's handwriting style of a particular letter.

This technique is useful in case that the number of samples is small. When the number of samples is large, the presented technique is still advantageous in terms of speed as it deals with only samples on the convex hull. However, it could give a wrong classification when the distribution is non-Gaussian or non-convex.

Efficient Search

In Chapter 6, a new fast nearest neighbor search algorithm with no degradation is proposed. Filtration with a threshold is the key idea for expedition. To perform this, we introduced ABT structure. The additive information of pattern feature vectors ensures that the query processing time reduces significantly. Furthermore, the idea is effective even in combination with other techniques like prototype thinning or clustering.

Discovery

Finally, in Chapter 7, we presented a datamining technique for sub-category classification problem. *Apriori* algorithm is applied to filter the sub-category problems with insufficient support. We considered a database consisting of writer data and features obtained from a handwriting sample, statistically representative of the US population, to determine similarity of a specific group of people. Higher classification rate is achieved with higher constraint sub-categories.

8.2 Future Work

In Chapter 2, albeit the simulating TOI saves enormous time to build the database to support the authorship confidence, it still takes a while to spot and segment the word and character out of the document unless the questioned TOI is already segmented and stored in the image database. The automatic line, word and character segmentation problem is one of open problems.

Standardizing procedures require a community-based effort. In order for the proposed procedure to be recognized as a part of the standard forensic document examination protocol, it is necessary to provide the validation that it uniquely identifies the writer using further statistical experimental design and protocol evaluation techniques.

In Chapter 3 on the distance between histograms, we dealt with in this paper are one dimensional arrays (univariate). However, there can be any dimensional ones and measuring the distance between multivariate histograms in Eq.(3.13) can be useful in many applications. For example, grey scale images can be considered as two dimensional histograms. The concept of distance introduced in this paper might be generalized and realized for the image similarity. Another challenging problem occurs when variables of histograms are different in type. We leave them as open problems to readers.

In Chapter 7, the subgroup with aggregation was not considered. The algorithm can be altered to handle the subgroup with agregation.

Appendix A
Features

A.1 Finding Connected Components Algorithms in Binary Image

A *connected component* is a subgraph such that for all vertices u and v in the subgraph, there is at least one path between u and v. It must not be contained in any larger connected subgraph. Connected components are widely used in many *OCR* systems for the process to isolate individual characters from the text image prior to *Character Recognition* [93]. Furthermore, it is employed to speed up *Document Analysis operations*; performance is better at the *Connected Component* level rather than at the image level [94].

The problem of finding the *Connected Components* of a graph can be solved by using *depth first search* with very little embellishment in $\theta(max(n, m))$ where $n = |V|$, the number of vertices and $m = |E|$, the number of edges [4]. This is true, however, when the graph is represented by the *adjacency list structure*. Each node in the *adjacency list* is visited once and thus running time is linear. The running time complexity would vary depending on how the input graph is originally represented. In case of binary images, the graph is represented in a primitive picture format. We might have to list all vertices, edges and the adjacency list as a pre-processing.

In this report, I will describe two linear time algorithms for finding connected components in binary image and their analysis. One is a recursive version and the other is a stack version. Due to the fact that edges are constant for all vertices in binary image, we still achieve the linear running time complexity. At last, I will analyze the source code currently in use in *NABR* system at *CEDAR* [10].

A.1.1 Definition

We assume that digital images are represented by rectangular arrays of picture elements called *pixels*. A pixel consists of either binary or a grey level represented numerically. In this report, only binary image is concerned and 0 and 1 are white and black respectively. Throughout the rest of this paper, n and m are numbers of rows and columns in the given image, correspondingly.

The problem of finding *Connected Components* is defined as follows:

Input: An image $I(i, j)$ where $i = (1..n)$ and $j = (1..m)$.

Output: a list of all *Connected Components*, $C = \{C_1, C_2, \cdots, C_k\}$ where k is the number of *Connected Components*. Each C_x is consisted of a list of pixels or blocks, B. Every element B_x of B must be an element in only one of C_y. Also, there exists at least one path between (B_x, B_y) if and only if B_x and B_y belong to a same C_z.

In the description of the problem, blocks are mapped to vertices instead of pixels for the avoidance of noise effect. Usually, 8 consecutive pixels in a row is a block and a block is 1 if the number of pixel 1's exceeds the threshold. Obviously, the ending block of each row might be shorter in size.

1	1	1	0	1	0	1	0	1	1	1
1	0	0	0	1	1	1	0	1	0	1
1	0	0	0	1	0	1	0	1	1	1
1	1	1	0	1	0	1	0	1	0	1

Figure A.1: A binary image and its connected components

Consider the figure A.1. There are 4 rows and 11 blocks in each row. There are 3 connected components and each set contains blocks. There are no intersections among sets, C's.

$C_1 = \{B_{(1,1)}, B_{(1,2)}, B_{(1,3)}, B_{(2,1)}, B_{(3,1)}, B_{(4,1)}, B_{(4,2)}, B_{(4,3)}\}$

$C_2 = \{B_{(1,5)}, B_{(2,5)}, B_{(3,5)}, B_{(4,5)}, B_{(2,6)}, B_{(2,7)}, B_{(1,7)}, B_{(3,7)}, B_{(4,7)}\}$

$C_3 = \{B_{(1,9)}, B_{(1,10)}, B_{(1,11)}, B_{(2,9)}, B_{(2,11)}, B_{(3,9)}, B_{(3,10)}, B_{(3,11)}, B_{(4,9)}, B_{(4,11)}\}$

In often cases, however, runs, R's, are used. Runs are connected subgraphs whose elements lie on a same row. They are consecutive 1's in a row. In the example of the figure A.1, there are 17 runs: $R_1 = \{B_{(1,1)}, B_{(1,2)}, B_{(1,3)}\}, R_2 = \{B_{(1,5)}\}, R_3 = \{B_{(1,7)}\}, \cdots, R_{17} = \{B_{(4,11)}\}$. These runs can be considered as vertices.

A.1.2 Algorithm 1 : a recursive version

The algorithm uses the recursive version of depth-first search. Consider the binary image as a four-nary tree which means that each node, block, can have up to four branches. These branches are top, bottom, left, and right nodes of the given node. Boundary blocks have less than four branches. Depending on your definition of connectivity, you might make the 8-nary tree. We do not have to build the tree data structure. The tree exists only in the conceptual level. All information we need is in the image. Consider the pseudo C code below. Initially, no node is visited. For every node, we will perform the depth first search. During the search, all nodes visited will be marked as visited. When visited node is met during the search, the branch is pruned; no further search is executed in that branch. In this sense, all nodes will be visited just once or constant times.

```
#include <stdio.h>

BlockListtype CComponent[MaxRows*MaxBlockCols];
int CNumber;
int Image[MaxRows][MaxBlockCols];
int Visited[MaxRows][MaxBlockCols];
```

```

main()
{
   int i, j;
   initialize0(Visited);
   CNumber = 0;
   for(i = 0; i < MaxRows; i++)
      for(j = 0; j < MaxBlockCols; j++)
         {
            if (Image[i][j] == 1 && Visited[i][j] == 0)
               {
                  CNumber++;
                  TravCComponent(i, j);
               }
         }
}

TravCComponent(int i, j)
{
   Visited[i][j] = 1;
   Add Block(i,j) to CComponent[CNumber];
   if(Image[i-1][j] == 1 && Visited[i-1][j] == 0)
      TravCComponent(i-1, j);
   if(Image[i+1][j] == 1 && Visited[i+1][j] == 0)
      TravCComponent(i-1, j);
   if(Image[i][j-1] == 1 && Visited[i][j-1] == 0)
      TravCComponent(i-1, j);
```

```
34      if(Image[i][j+1] == 1 && Visited[i][j+1] == 0)
35          TravCComponent(i-1, j);
36      return;
37  }
```

Correctness

The algorithm to work correctly must satisfy the definition of the problem.

Lemma A.1.1 *For all blocks, B_x is an element of only one C_y.*

Proof:. At the visit of all B_x's, B_x is assigned to one of C_y. This is done at the line 27. Every B_x is assigned to at least one of Connected Components because we loop the traversing for all blocks. At the first visit of all B_x's, the flag, *Visited*, is set to 1. Thus visited node will get changed no longer. ∎

Lemma A.1.2 *For all B_x and B_y where B_x, B_y are elements in C_z, there is a path between B_x and B_y in the sub-graph, C_z.*

Proof:. The root of each connected component is the left-most block in the top row of the connected component. Call this root as R. Suppose $R = B_x$. There is a path between B_x and B_y because B_y is traversed from the root. If $R = B_y$, there is a path in the same sense. Consider the case when R is neither B_x or B_y. What we know is that there is a path from R to B_x and a path from R to B_y. The sub-graph is bidirectional. Therefore, there is always a path between B_x and B_y. ∎

Complexity

Lemma A.1.3 *The computational time complexity of the algorithm is $\theta(nm)$*

Proof: Each node is visited only up to four times. At the first visit of one arbitrary node, the flag, *visited*, for the node is set to 1. Other visits can happen only when its neighbors are visited for the first time and try to traverse this node. But since the flag is set, it will be pruned. There are up to 4 or 8 neighbors depending on the definition. Therefore, the time complexity is linear to the number of blocks. There are up to $n \times m$ number of blocks. $\blacksquare$

Lemma A.1.4 *The space used is $\theta(nm)$.*

Proof: The size of input image is $n \times m$. In the declaration of the above code, in the line 6, we used an array of flags, *visited* which is only $n \times m$. We also declared CComponent array. This would be nicer to make it a linked list since the size of connected components is dynamic. In either way, it affects neither the time complexity nor the space complexity. $\blacksquare$

A.1.3 Algorithm 2 : a stack version

Depth first search can be implemented by using *stacks* still guaranteeing the linear time complexity. It is actually implemented in *NABR* system [10]. In this version, we will use runs, R's as vertices. This version of finding Connected components has mainly three steps. First, create runs, R, as a pre-processing, next connect runs, and finally detect all connected components. These steps can be viewed as listing vertices, building the adjacency lists, and then finally searching accordingly. Consider the following pseudo code.

```
for (i=0; i < n; i++)   /* finding vertices*/
  for (j=0; j < m; j++)
    find_the_linked list for R[i];

for (i=0; i < n; i++)   /* finding adjacency lists*/
  for all R_x in R[i]
    find runs in R[i-1] which are adjacent to R_x;
```

```
8      find runs in R[i+1] which are adjacent to R_x;
9
10 for (i=0; i < n; i++)   /* depth first search */
11   for all R_x in R[i]
12     if (flag for R_x is not set)
13       set flag for R_x;
14       CNumber++;
15       Add R_x to CComponent[CNumber];
16       Push all R_y's which are adjacent to R_x;
17       Pop(R_y);
18       while (!stackempty)
19         if (flag for poped R is not set)
20            set flag for poped R_y
21            Add R_y to CComponent[CNumber];
22            Push all R's which are adjacent to R_y;
23            Pop(R);
```

Correctness

The algorithm must satisfy the definition of the problem. Note that a connected component is a set of runs instead of blocks used in the first algorithm.

Lemma A.1.5 *For all runs, R_x is an element of only one C_y.*

Proof:. If R_x is a root, it is assigned to C_y in the line 15. Otherwise, it is assigned to C_y in the line 21. Because initially all flags were not set, and we search for all runs, all runs are certainly assigned to at least one of connected components. When a run is popped or a root, the flag is set. This means once one run is assigned to one of connected component, it never gets changed. ∎

Lemma A.1.6 *For all R_x and R_y where R_x, R_y are elements in C_z, there is a path between R_x and R_y in the sub-graph, C_z.*

Proof:. omitted since very similar to lemma 3.2 ∎

Complexity

Lemma A.1.7 *The computational time complexity of the algorithm is $\theta(nm)$*

Proof: First, creating runs is done in linear time by scanning the image once. The complexity is $O(nm)$ where n is the number of rows and m is the number of cols. We assume that the number of runs in one row, $|R|$ is $O(m)$. Because the size of each run tend to be constant. Therefore, $R = m/c$ where c is some constant value. The connecting runs takes also $O(nm)$. Detecting the connected component step is also $O(nm)$. This can be seen in the stack. Each run goes to the stack either 0 for the root or once for the non-root. Therefore, the total running time is linear. ∎

The running time of the step connecting runs in the source code is, however, $O(nR^2)$ or $O(nm^2)$. This is because the äthor of the program used the linked list structure for runs. Runs in a row is linked together from left to right. In order to examine connections, one has to traverse from the beginning of the previous run linked list. It stops if the column location of the previous run passes the end of current row. This saves some time, but it does not change its complexity in order because traversal takes approximately Rth position number of times for the run, R. The running time is $\sum_{i=0}^{R} i = (R-1)R/2$ for each row. This could be simply improved to $O(nR)$ by not initializing the previous run and keeping the location of where it stopped.

The space used is $\theta(nm)$.

References

[1] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, volume 22, pages 207–216, June 1993.

[2] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and Inkeri Verkamo. Fast discovery of association rules. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 12. AAAI Press, 1996.

[3] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th Int'l Conference on Very Large Databases*, volume 2, pages 478–499, September 1994.

[4] Sara Baase. *Computer Algorithms Introduction to Design and Analysis*. Addison-Wesley, 2nd edition, 1988.

[5] B. Bhattacharya. Worst-case analysis of a convex hull algorithm. unpublished manuscript, February 1982.

[6] Russell R. Bradford and Ralph B. Bradford. *Introduction to Handwriting Examination and Identification*. Nelson-Hall Publishers: Chicago, 1992.

[7] Alan J. Broder. Strategies for efficient incremental nearest neighbor search. *Pattern Recognition*, 23(12):171–178, 1990.

[8] M K. Brown and S. Ganapathy. Preprocessing techniques for cursive script word recognition. *Pattern Recognition*, 16(5):447–458, 1983.

[9] D J. Burr. Designing a handwriting reader. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5:554–559, 1983.

[10] CEDAR. Gencmp.c. Source Code in NABR system at CEDAR.

[11] Sung-Hyuk Cha. Fast image template and dictionary matching algorithms. In *Proceedings of ACCV '98 LNCS - Computer Vision*, volume 1351, pages 370–377. Springer-Verlag, Jan 1998.

[12] Sung-Hyuk Cha. Efficient algorithms for image template and dictionary matching. *Journal of Mathematical Imaging and Vision*, 12(1):81–90, February 2000.

[13] Sung-Hyuk Cha, Yong-Chul Shin, and Sargur N. Srihari. Algorithm for the edit distance between angular type histograms. Technical Report CEDAR-TR-99-1, SUNY at Buffalo, April 1999.

[14] Sung-Hyuk Cha, Yong-Chul Shin, and Sargur N. Srihari. Approximate character string matching algorithm. In *Proceedings of Fifth International Conference on Document Analysis and Recognition*, pages 53–56. IEEE Computer Society, September 1999.

[15] Sung-Hyuk Cha, Yong-Chul Shin, and Sargur N. Srihari. Approximate string matching for stroke direction and pressure sequences. In *Proceedings of SPIE, Document Recognition and Retrieval VII*, volume 3967, pages 2–10, January 2000.

[16] Sung-Hyuk Cha and Sargur N. Srihari. Approximate string matching for angular string elements with applications to on-line and off-line handwriting recognition. *TPAMI*, 2000. In Review.

[17] Sung-Hyuk Cha and Sargur N. Srihari. Assessing the authorship confidence of handwritten items. In *Proceedings of WACV 2000*, pages –. IEEE Computer Society, December 2000.

[18] Sung-Hyuk Cha and Sargur N. Srihari. Convex hull discriminant function and its application to writer identification. In *Proceedings of JCIS 2000 CVPRIP*, volume 2, pages 139–142, Febrary 2000.

[19] Sung-Hyuk Cha and Sargur N. Srihari. Distance between histograms of angular measurements and its application to handwritten character similarity. In *Proceedings of 15th ICPR*, pages 21–24. IEEE CS Press, 2000.

[20] Sung-Hyuk Cha and Sargur N. Srihari. A fast nearest neighbor search algorithm by filtration. *Pattern Recognition Journal*, 2000. In print.

[21] Sung-Hyuk Cha and Sargur N. Srihari. Multiple feature integration for writer verification. In *Proceedings of 7th IWFHR 2000*, pages 333–342, September 2000.

[22] Sung-Hyuk Cha and Sargur N. Srihari. Nearest neighbor search using additive binary tree. In *Proceedings of CVPR 2000*, volume 1, pages 782–787. IEEE Computer Society, June 2000.

[23] Sung-Hyuk Cha and Sargur N. Srihari. On measuring the distance between histograms. *submitted to Pattern Recognition*, 2000.

[24] Sung-Hyuk Cha and Sargur N. Srihari. System that identifies writers. In *Proceedings of 7th National Conference on Aritificial Intelligence*, page 1068. AAAI, August 2000.

[25] Sung-Hyuk Cha and Sargur N. Srihari. Writer identification: Statistical analysis and dichotomizer. In *Proceedings of SS&SPR 2000 LNCS - Advances in Pattern Recognition*, volume 1876, pages 123–132. Springer-Verlag, September 2000.

[26] Sung-Hyuk Cha and Sargur N. Srihari. Handwritten document image database construction and retrieval system. In *Proceedings of SPIE, Document Recognition and Retrieval*, volume 4307, pages 13–21, January 2001.

[27] D. R. Chand and S. S. Kapur. An algorithm for convex polytopes. *JACM*, 17(1):78–86, January 1970.

[28] Chihau Chen. *Statistical pattern recognition.* Rochelle Park, N.J., Hayden Book Co., 1973.

[29] Vladimir Cherkassky, Jerome H. Friedman, and Harry Wechsler. *From Statistics to Neural Networks: Theory and Pattern Recognition Applications.* Springer, nato asi edition, 1994.

[30] Sung C. Choi and Ervin Y. Rodin. *Statistical Methods of Discrimination and Classification, Advances in Theory and Applications.* Pergamon Press, 1986.

[31] Chao K. Chow. On optimum recognition error and reject tradeoff. *IEEE Transactions on Information Theory*, 16:41–46, 1970.

[32] Belur V. Dasarathy. Visiting nearest neighbors- a survery of nearest neighbor pattern classification techniques. In *Proceedings of the International Conference on Cybernetics and Society*, pages 630–636. IEEE, September 1977.

[33] Belur V. Dasarathy. *Nearest Neighbor Pattern Classification Techniques.* IEEE Computer Society Press, 1991.

[34] Daubert. Daubert vs. merrell dow pharmaceuticals. 509 U.S. 579, 1993.

[35] Richard O. Duda and Peter E. Hart. *Pattern classification and scene analysis.* New York, Wiley, 1st edition, 1973.

[36] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification.* John Wiley & Sons, Inc., 2nd edition, 2000.

[37] Olive Jean Dunn and Virginia A. Clark. *Applied Statistics: Analysis of Variance and Regression.* John Wiley & Sons, 2nd edition, 1987.

[38] Jeff Erickson. Computational geometry pages. "http://compgeom.cs.uiuc.edu/~jeffe/compgeom/compgeom.html".

[39] Martin Farach. Optimal suffix tree construction with large alphabets. In *38th Annual Symposium on Foundations of Computer Science*, pages 137–143, Miami Beach, Florida, October 1997. IEEE.

[40] András Faragó, Tamás Linder, and Gábor Lugosi. Fast nearest neighbor search in dissimilarity spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:957–962, 1993.

[41] John T. Favata and Geetha Srikantan. A multiple feature/resolution approach to handprinted digit and character recognition. *International Journal of Imaging Systems and Technology*, 7:304–311, 1996.

[42] John T. Favata, Geetha Srikantan, and Sargur N. Srihari. Handprinted character/digit recognition using a multiple feature/resolution philosophy. In *IWFHR-IV*, pages 57–66, December 1994.

[43] Myron Flickner, Harpreet Sawhney, Wayne Niblack, Jonathan Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, David Steele, and Peter Yanker. Query by image and video content: the qbic. *Computer*, 28(9):23–32, Sept 1995.

[44] J. Friedman, J. Bentley, and R. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3:209–226, 1977.

[45] Yoshiji Fugimoto, Syozo Kadota, Shinichi Hayashi, Masao Yamamoto, Syunichi Yajima, and Michio Yasuda. Recognition of handprinted characters by nonlinear elastic matching. In *The Third International Joint Conference on Pattern Recognition*, pages 113–118, November 1976.

[46] K. Fukunaga and P. Narendra. A branch and bound algorithm for computing k-nearest neighbors. *IEEE Transactions on Computers*, 24:750–743, 1975.

[47] G.W. Gates. The reduced nearest neighbor rule. *IEEE Transactions on Information Theory*, IT-18(3):431–433, May 1972.

[48] Rafael C. Gonzalez and Michael G. Thomason. *Syntactic Pattern Recognition: an Introduction*. Addison-Wesley, 1978.

[49] C M. Greening and V K. Sagar. Image processing and pattern recognition framework for forensic document analysis. In *IEEE Annual International Carnahan Conference on Security Technology*, pages 295–300. IEEE, 1995.

[50] C M Greening, V K Sagar, and C G Leedham. Handwriting identification using global and local features for forensic purposes. In *IEE Conference Publication*, number 408, pages 272–278. IEE, 1995.

[51] Patrick A. V. Hall and Geoff R. Dowling. Approximate string matching. *ACM Computing Surveys*, 12(4):381–402, December 1980.

[52] D. Harel and R.E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13:338–355, 1984.

[53] P.E. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, IT-14(3):515–516, May 1972.

[54] Ordway Hilton. The relationship of mathematical probability to the handwriting identification problem. In *Proceedings of Seminar No. 5*, pages 121–130, 1958.

[55] Gary Holcombe, Graham Leedham, and Vijay Sagar. Image processing tools for the interactive forensic examination of questioned documents. In *IEE Conference Publication*, number 408, pages 225–228. IEE, 1995.

[56] Tao Hong, Stephen W. Lam, Jonathan J. Hull, and Sargur N. Srihari. The design of a nearest-neighbor classifier and its use for japanese character recognition. In *Proceedings of the Third International Conference on Document Analysis and Recognition(ICDAR '95)*, pages 370–377. IEEE, August 1995.

[57] Roy A. Huber and A. M. Headrick. *Handwriting Identification: Facts and Fundamentals*. CRC Press LLC, 1999.

[58] Thomas Kailath. The divergence and bhattacharyya distance measures in signal selection. *IEEE Trans. on Communication Technology*, COM-15(1):52–60, February 1967.

[59] M. Kallay. Convex hull algorithms in higher dimensions. unpublished manuscript, 1981.

[60] Moshe Kam, Babriel Fielding, and Robert Conn. Writer identification by professional document examiners. *Journal of Forensic Sciences*, 42(5):778–786, January 1997.

[61] Moshe Kam, Joseph Wetstein, and Robert Conn. Proficiency of professional document examiners in writer identification. *Journal of Forensic Sciences*, 39(1):5–14, January 1994.

[62] Baek S. Kim and Song B. Park. A fast k-nearest neighbor finding algorithm based on the ordered partition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):761–766, November 1986.

[63] S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:79–86, 1951.

[64] Gad M. Landau and Uzi Vishkin. Introducing efficient parallelism into approximate string matching and a new serial algorithm. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, pages 220–230, 1986.

[65] Huan Liu and Hiroshi Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers, 1998.

[66] Andres Marzal and Enrique Vidal. Computation of normalized edit distance and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):926–932, September 1993.

[67] Kameo Matusita. Decision rules, based on the distance, for problems of fit, two samples, and estimation. *Annals of Mathematical Statistics*, 26:631–640, 1955.

[68] Tom Mitchell. *Machine Learning.* McGraw Hill, 1997.

[69] Donald F. Morrison. *Multivariate statistical methods.* New York : McGraw-Hill, 1990.

[70] George Nagy. Twenty years of document image analysis in pami. *IEEE Transactions on Pattern Analysis and Recognition*, 22(1):38–62, January 2000.

[71] H. Niemann and R. Goppert. An efficient branch-and-bound nearest neighbour classifier. *Pattern Recognition Letters*, 7:67–72, 1988.

[72] Fathallah Nouboud and Réjean Plamondon. On-line recognition of handprinted characters. survey and beta tests. *Pattern Recognition*, 23(9):1031–1044, 1990.

[73] U.S. Department of Commerce. Population profile of the united states. Current Population Reports Special Studies P23-194, Semtember 1998.

[74] Lawrence O'Gorman and Rangachar Kasturi. *Document image analysis.* IEEE Computer Society, 1995.

[75] Albert S. Osborn. *Questioned Document.* Albany, N.Y. : Boyd Print. Co., 2nd edition, 1929.

[76] C. Papadimitriou and J. Bentley. A worst-case analysis of nearest neighbor searching by projection. In *In Automata, Languages and Programming LNCS*, volume 85, pages 470–482. Springer-Verlag, 1980.

[77] Marc Parizeau, Nadia Ghazzali, and Jean-Francois Hebert. Optimizing the cost matrix for approximating string matching using genetic algorithms. *Pattern Recognition*, 31(4):431–440, 1998.

[78] Greg Pass, Ramin Zabih, and Justin Miller. Comparing images using color coherence. In *ACM International Multimedia Conference*, pages 65–73. ACM, 1996.

[79] Ioannis Pavlidis, Rahul Singh, and Nikolaos Papanikolopoulos. On-line handwritten note recognition method using shape metamorphosis. In *International Conference on Document Analysis and Recognition*, pages 914–918. IEEE, 1997.

[80] Rejean Plamondon and Guy Lorette. Automatic signature verification and writer identification - the state of the art. *Pattern Recognition*, 22(2):107–131, 1989.

[81] Rejean Plamondon and Fathallah Nouboud. On-line character recognition system using string comparison processor. In *Proceedings of International Conference on Pattern Recognition*, pages 460–463. IEEE, June 1990.

[82] Réjean Plamondon and Sargur N. Srihari. On-line and off-line handwriting recognition: A comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):63–84, 2000.

[83] Franco P. Preparata and Michael Ian. Shamos. *Computational geometry: an introduction*. New York : Springer-Verlag, 1985.

[84] Harpreet S. Sawhney and James L. Hafner. Efficient color histogram indexing. In *International Conference on Image Processing*, volume 1, pages 66–70. IEEE, 1994.

[85] P.H. Sellers. The theory and computation of evolutionary distances: pattern recognition. *Journal of Algorithms*, 1:359–373, 1980.

[86] C.E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.

[87] John E. Shore and Robert M. Gray. Minimum cross-entropy pattern classification and cluster analysis. *Transaction on Pattern Analysis and Machine Intelligence*, 4(1):11–17, 1982.

[88] M. L. Simmer, C. G. Leedham, and A.J.W.M-Thomassen. *Handwriting and Drawing Research: Basic and Applied Issues*. Amsterdam: IOS Press, 1996.

[89] Rohini K. Srihari. On-line handwriting database. http://www.cedar.buffalo.edu/Linguistics/database.html, Jan 1997.

[90] Sargur N. Srihari. *Computer Text Recognition and Error Correction*. IEEE Computer Society, 1984.

[91] Sargur N. Srihari, Sung-Hyuk Cha, Hina Arora, and Sangjik Lee. Handwriting identification: Research to study validity of individuality of handwriting & develop computer-assisted procedures for comparing handwriting. *submitted to Journal of Forensic Sciences*, 2001.

[92] Sargur N. Srihari and E.J. Keubert. Integration of hand-written address interpretation technology into the united states postal service remote computer reader system. In *Proceedings of 4th International Conference on Document Analysis and Recognition (ICDAR'97)*, pages –, Ulm, Germany, 1997.

[93] Sargur N. Srihari and Stephen W. Lam. Character recognition. Technical Report CEDAR-TR-95-1, SUNY at Buffalo, Jan 1995.

[94] Sargur N. Srihari, Yong-Chul Shin, Vemulapati Ramanaprasad, and Dar-Shyang Lee. A system to read names and addresses on tax forms. Technical Report CEDAR-TR-94-2, SUNY at Buffalo, Feb 1994.

[95] Geetha Srikantan, Stephen W. Lam, and Sargur N. Srihari. Gradient-based contour encoding for character recognition. *Pattern Recognition*, 29(7):1147–1160, 1996.

[96] Geetha Srikantan, Dar-Shyang Lee, and John T. Favata. Comparison of normalization methods for character recognition. In *Proceedings of the Third ICDAR 95*, volume 2, pages 719–722. IEEE Computer Society Press, August 1995.

[97] Starzecpyzel. United states vs. starzecpyzel. 880 F. Supp. 1027 (S.D.N.Y), 1995.

[98] Graham A. Stephen. *String Searching Algorithms*. Singapore:World Scientific, 1994.

[99] Ching Y. Suen, Marc Berthod, and Shunji Mori. Automatic recognition of handprinted characters - the state of the art. In *Proceedings of the IEEE*, volume 68, pages 469–487, April 1980.

[100] Ching Y. Suen and Patrick S. P. Wang. *Thinning Methodologies for Pattern Recognition*. World Scientific, 1994.

[101] Michael J. Swain and Dana H. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, Nov 1991.

[102] Charles C. Tappert, Ching Y. Suen, and Toru Wakahara. The state of the art in on-line handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(8):787–808, 1990.

[103] Tamara Plakins Thornton. *Handwriting in America*. New Haven : Yale University Press, 1996.

[104] Godfried T. Toussaint. Bibliography on estimation of misclassification. *IEEE Transactions on Information Theory*, 20(4):472–479, July 1974.

[105] Jeremy Travis. Forensic document examination validation studies. Solicitation: http://ncjrs.org/pdffiles/sl297.pdf, October 1998.

[106] Scott E Umbaugh. *Computer Vision and Image Processing*. Prentice Hall PRT, 1998.

[107] Enrique Vidal, Héctor Rulot, Francisco Casacuberta, and José Benedí. Searching for nearest neighbors in constant average time with applications to discrete utterance speech recognition. In *Proceedings of International Conference on Pattern Recognition*, pages 808–810. IEEE, Oct 1986.

[108] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 6(1):168–178, January 1974.

[109] P. Weiner. Linear pattern matching algorithm. In *Proceedings 14th IEEE Symposium on Switching and Automata Theory*, 1973.

[110] Neil A. Weiss. *Introductory Statistics.* Addison-Wesley, 5th edition, 1999.

[111] P. J. Ye, H. Hugli, and F. Pellandini. Techniques for on-line chinese character recognition with reduced writing constraints. In *Proceedings of 7th ICPR*, pages 1043–1045. IEEE CS Press, 1984.

Vita

Sung-Hyuk Cha

1970 Born in Seoul, Korea on December, 29th.

1989 Graduated from YoungDong High School, Seoul, Korea.

1993 a member of Golden Key National Honor Society.

1994 B.S. with high honor in Computer Science, Rutgers, The State University of New Jersey, New Brunswick, New Jersey.

1994 High Honors in Computer Science.

1994 a member of Phi Beta Kappa.

1994-96 Graduate work in Computer Science, Rutgers, The State University of New Jersey, New Brunswick, New Jersey.

1995-96 Part-time Lecturer, Covered CS111 recitations and Grading for 71 students.

1996 M.S. in Computer Science, Rutgers, The State University of New Jersey, New Brunswick, New Jersey.

1998 Sung-Hyuk Cha, *Fast Image Template and Dictionary Matching Algorithms* ACCV '98 Proceedings, LNCS Computer Vision Vol. I, XXIV p370-377, 1997, Springer

1996-96 Summer Programming Position, Prof. Casimir Kulikowski, developed the web-based iconic radiology interface prototype system.

1996-98 Assistant Engineer, Information Technology R&D Center, Samsung Data Systems Co., LTD. Seoul, Korea. Specialized in Medical Information Systems.

1997-98 Seoul National University Telemedicine Project Contractor, developed the video conferencing and web-based medical information system.

1997 Sung-Hyuk Cha M.S. and Sang-Bok Cha M.D., *Iconic Communication Method for Liver Disease on Teleradiology* IMAC '97 Proceedings, Oct. 1997, p 238-246

1997 Sung-Hyuk Cha, *Medical Image Processing by using the Intensity Histogram* IMAC '97 Proceedings, Oct. 1997, p 233-237

1997 Sung-Hyuk Cha M.S. and Sang-Bok Cha M.D., *Iconic Communication Method for Liver Disease on Teleradiology* Korean PACS Journal, vol 3, yr 1997, p 17-25

1997 Sung-Hyuk Cha, *Medical Image Processing by using the Intensity Histogram* Korean PACS Journal, vol 3, yr 1997, p 53-57

1998 Sung-Hyuk Cha, *Fast image template and dictionary matching algorithms* In *Proceedings of ACCV '98 LNCS - Computer Vision*, volume 1351, pages 370–377. Springer-Verlag, Jan 1998

1998 a member of Korean PACS Soceity.

1998-2001 Graduate work in Computer Science and Engineering, The State University of New York, Buffalo, New York.

1998-2001 Research Project Assistant, CEDAR, The State University of New York, Buffalo, New York.

1999 Ph.D. Candidate in Computer Science

1999 NIJ research project award # 1999-IJ-CX-K010, Forensic Document Examination Validation Study $428,000

1999 Sung-Hyuk Cha and Sargur N. Srihari, *Handwriting Identification* Sigma Xi student poster competition, April, 1999

1999 Sung-Hyuk Cha and Sargur N. Srihari, *Handwriting Identification* poster presentation in CSE department conference, April, 1999

1999 Sung-Hyuk Cha, Yong-Chul Shin and Sargur N. Srihari, *Algorithm for the Edit Distance between Angular Type Histograms* Technical Report, CEDAR-TR-99-1, April, 1999

1999 Sung-Hyuk Cha, Yong-Chul Shin and Sargur N. Srihari, *Approximate Character Stroke Sequence String Mathing* In *Proceedings of Fifth International Conference on Document Analysis and Recognition*, pages 53–56, September 1999

2000 a student member of AAAI

2000 a student member of IEEE and its Computer Society

2000 Sung-Hyuk Cha, Yong-Chul Shin and Sargur N. Srihari, *Approximate string matching for stroke direction and pressure sequences* In *Proceedings of SPIE's Electronic Imaging 2000, Document Recognition and Retrieval VII*, pages 2–10, January 2000

2000 Sung-Hyuk Cha, *Efficient Algorithms for Image Template and Dictionary Matching* Journal of Mathematical Imaging and Vision, Vol 12, issue 1, February 2000, pages 81-90.

2000 Sung-Hyuk Cha and Sargur N. Srihari, *Writing Speed and Writing Sequence Invariant On-line Handwriting Recognition*, to appear in *Lecture Notes in Pattern Recognition*, edited by Sankar Pal and Amita Pal, World Scientific Puhblishing Co., October, 2000

2000 Sung-Hyuk Cha and Srikanth Munirathnam, *Comparing Color Images using Angular Histogram Measures* In *Proceedings of JCIS 2000*, pages 139–142, Feb. 2000

2000 Sung-Hyuk Cha and Sargur N. Srihari, *Convex Hull Discriminant Function and its Application to Writer Identification Problem* In *Proceedings of JCIS 2000*, pages 13–16, Feb. 2000.

2000 Sung-Hyuk Cha and Sargur N. Srihari, *Mapping the Many Class Problem into a Dichotomy using Distance Measures an international conference of Statistics*, in honor of Professor C.R. Rao on the occasion of his 80th birthday, 2000 San Antonio

2000 Sung-Hyuk Cha and Sargur N. Srihari, *Writer Identification* Sigma Xi student poster competition, April, 2000

2000 Sung-Hyuk Cha and Sargur N. Srihari, *Nearest Neighbor Search using Additive Binary Tree* In *Proceedings of CVPR 2000*, pages 782-787, June 2000

2000 Sung-Hyuk Cha, *Writer Identification using Distance Measures and Dichotomies* CEDAR Colloqquium presented on June 21, 2000

2000 Sung-Hyuk Cha and Sargur N. Srihari, *System that Identifies Writers* In *Proceedings of AAAI 2000*, July, 2000, p 1068

2000 Teaching Assistant, Department of Computer Science & Engineering, The State University of New York, Buffalo, New York.

2000 Sung-Hyuk Cha and Sargur N. Srihari, *Writer Identification: Statistical Analysis and Dichotomizer* In *Proceedings of SS&SPR 2000*, pages 123-132, August, 2000

2000 Sung-Hyuk Cha and Sargur N. Srihari, *Distance between Histograms of Angular Measurements and its Application to Handwritten Character Similarity* In *Proceedings of ICPR 2000*, pages 21-24, September, 2000.

2000 Sung-Hyuk Cha and Sargur N. Srihari, *Multiple Feature Integration for Writer Identification* In *Proceedings of 7th IWFHR 2000*, ranked 9, pages 333-342, September, 2000

2000 Teaching Assistant for CSE 474/574 Machine Learning.

2000 Sung-Hyuk Cha and Sargur N. Srihari, *Assessing the Authorship Confidence of Handwritten Items* In *Proceedings of WACV 2000*, December 2000.

2001 Sung-Hyuk Cha and Sargur N. Srihari, *Handwritten Document Image Database Construction and Retrieval System*, In *Proceedings of SPIE's Electronic Imaging 2001, Document Recognition and Retrieval VII*, pages 13–21, January 2001.

2001 a student member of IS&T (Society for Imaging Science and Technology).

2001 listed in *Strathmore's Who's Who* 2001-2002 edition.

2001 Sung-Hyuk Cha and Sargur N. Srihari, *A Fast Nearest Neighbor Search Algorithm by Filtration*, Accepted as a full length paper in *Pattern Recognition Journal*.

2001 Sung-Hyuk Cha and Sargur N. Srihari, *Writing Speed and Writing Sequence Invariant On-line Handwriting Recognition*, To appear in a book chapter of *Lecture Notes in Pattern Recognition* edited by Sankar Pal and Amita Pal, World Scientific.

2001 expecting Ph.D. in Computer Science and Engineering.

2001 appointed as an assistant professor of the department of computer science at Pace university in Westerchester.