THE CHINESE UNIVERSITY OF HONG KONG

DEPARTMENT OF INFORMATION ENGINEERING

MASTER OF SCIENCE DEGREE IN INFORMATION ENGINEERING

DESIGN AND IMPLEMENTATION OF MULTISTAGE TREE CLASSIFIER FOR CHINESE CHARACTER RECOGNITION

YEUNG LAP KEI

JUNE 1992

PREFACE

PREFACE

The objective of the project is to design and construct a practical off-line character recognition system for printed Chinese character. Two approaches i.e. unsupervised learning and supervised learning has been studied. Finally, supervised learning approach with the basis of the 3 corner code was adopted. The philosophy of the supervised learning is to make use of the similar properties of the corner feature and 3 corner input code. The corner features are first extracted and then classify based on the knowledge given from 3 corner code. The classification are done at various node and finally a binary decision tree is formed.

It has been shown in the study that the construction of the optimum decision tree belongs to the class of NP complete problems and thus cannot be built within polynomial time. As a result, a heuristic is introduced and suboptimal decision tree is built.

A two stage classification procedure has been introduced where a 3-tree protocol has been developed for the first stage discrimination process. the use of the context relationship among inputted characters is used to serve for the second stage classification. Random samples of both ideal type or having been corrupted with noise have been used to test the efficiency and accuracy of this classifier.

Simple testing has been carried out and the result shows that the performance of the classifier is rather sensitive to the noise of the input character. As a result, different techniques for improving the performance of the classifier have been studied which includes the introduction of overlapping, the technique of back tracking for holes, the use of a fuzzy decision making use of a tolerance limit and the idea of entropy reduction in other tree architecture. Some of these can actually improve the performance of the classifier to quite significant degree and among which the 1-tree protocol which employs the use of entropy reduction is found to be the most promising and worth future exploration in more depth. By the time, since only 500 characters are used in the training of the classifier, it is not reliable enough.

This thesis can be said is the combined effort of me and my project partner Mr. Sin Ka Wai. Part of the research is actually done by him. In order to maintain the completeness of the whole topics, the studies from Mr. Sin Ka Wai will be attached at the appendix for easy reference.

Last but not least, I would like to take this opportunity to thank my supervisor Dr. M. Chang for his invaluable guidance and assistance and my project partner Mr. Sin Ka Wai for his advise, support and coordination throughout the whole project.

Mr. Yeung Lap Kei

June 1992

ABSTRACT

DESIGN AND IMPLEMENTATION OF MULTISTAGE TREE CLASSIFIER FOR CHINESE CHARACTER RECOGNITION

YEUNG LAP KEI

ABSTRACT

Input has long been one of the most difficult problems faced by most scientists in the development of practical computer Chinese information processing systems. The intrinsic difficulties of the input of Chinese characters has hindered the advancement of these Chinese systems and Chinese character recognition is thought to be the ultimate solution. Classical recognition systems are either font dependent or not efficient enough in terms of the recognition speed. Usually the size of the character set under consideration is restricted to be just a small one and cannot comprises of all the Chinese characters that we commonly encountered.

Exploration has been made to construct a multistage tree classifier for printed Chinese characters based on the idea of supervised learning. The 3 corner code has been adopted as the basis in the construction of the tree classifier. A two staged classification procedure is used where in the first stage, a 3-tree protocol has been developed and used to discriminate 5400 commonly used Chinese characters. Improvement in its performance

can also be sought through the introduction of overlapping or the technique of back tracking for holes. The result is encouraging and an accuracy of up to 100% can be attained for ideal sample input and around 65% for noisy sample input. The classifier is also efficient where the average recognition time is around 0.5 second per character.

The second stage of classification explores the use of the context relationship among the inputted characters. Again, the performance of the classifier is good where for an ideal passage input, about 40% of the characters can be uniquely determined, 58% recognized as a group of characters and less than 3% are misclassified.

CONTENT

PREFACE

ABSTRACT

CONTENT

APPENDICES

REFERENCE

CHAPTER 1 -- INTRODUCTION

§1 INTRODUCTION

§1.1 THE CHINESE LANGUAGE

The Chinese language, being in use by billions of people in the world, is different from other languages in the western world in its representation method. They are stand alone characters and square shaped. Unlike most of other languages, Chinese words are not formed by sequence of alphabets or sequence of some other symbols. Basically one symbol will be one Chinese character. It has its own meaning and its own pronunciation. Of course, phrases can also be formed by combining separate characters just like other languages.

As a result of the above consideration, we note that since characters in Chinese exists by its own, and there is no basic group of symbols used in forming word, the total number of characters could theoretically be infinite. There is no upper bound for the number of distinct characters. Fortunately the Chinese culture tells us that there are altogether only around 20 thousands distinct Chinese characters and out of these, only about 5 thousands are commonly used. However, this is already an astonishing large number and has presented an intrinsic difficulties in Chinese character recognition system in comparison with the English recognition system where only tens of alphabets are to be distinguished.

The astonishing large number of characters in the Chinese language is, on one hand a problem which we have to face while on the other hand, one of the distinguished feature which enables a easier recognition process. It has been claimed that since there are many distinct characters, we can distinguish them more easily as the characters are usually very different from one another. People may just look at the corners or the edges before they can identify the characters correctly. Despite of such advantages, Chinese character recognition is still a difficult and challenging problem for most of the scientists and has attracted much research to be carried on this field [1].

§1.2 CHINESE INFORMATION PROCESSING SYSTEM

Chinese information processing system has been developed rapidly in the past decades both in Taiwan and Mainland China. Most of the commercially available Chinese systems in Hong Kong come from Taiwan.

Just like the coding system usually employed for English in most of the computer system, there is an internal code associated with every Chinese characters. This internal code will just resemble the role of ASCII in English alphabet. However, there are a large number of coding system and among which one of the most popular system, especially in Taiwan, is the use of Big 5 code. Basically every Chinese character is represented by a Big 5 code of two bytes in length. So as

to distinguish a Chinese character from the English alphabets, the most significant bit of all Big 5 codes will be 1. The Big 5 code will begin at hexadecimal a440 which is the internal code for "—". As a result, upon receiving an internal code of length one byte, the computer system will first access the most significant bit of that code. If the bit is found to be 0, then it is a usual ASCII code and will just be identified as usual characters. If the bit is found to be 1, then it will be the start of a Big 5 code. The other one byte of information will be taken and the two will be combined to become a Big 5 code of a certain Chinese character. In this way, Chinese characters and other usual characters can coexist in the system.

One fundamental problem in Chinese system is about input. To input an English alphabet is simple because the keyboard is primarily designed for this. How can we input a Chinese character to the computer system? In fact input has long been a difficult task in Chinese language processing by computer. Many different input methods have been developed. Typical examples include:

1. The use of a combination of keys to input a Chinese character
   a. By partial forms (e.g. the Chong Qi input scheme); or
   b. By the sequence of strokes in forming a character;
2. To input the phonemes which represent the sound of the character.

However, the most convenient method of input nowadays still require combination of keys to uniquely determine one single Chinese character. Most methods cannot uniquely determine a character. Usually they will seek the help from the user by asking the user to select for the computer among some possible characters displayed on the screen. These methods are slow and require appropriate training and practice before users can easily get along with. This intrinsic difficulty hinders further advancement in Chinese information processing system.

§1.3 CHINESE CHARACTER RECOGNITION

Different alternatives other than the conventional use of the keyboard have been proposed for inputting Chinese characters. Better solutions include the input in handwritten form, the input in preprinted form or the input in the form of voice. Among these input schemes, speech recognition should undoubtedly be our ultimate choice despite the intrinsic difficulties encountered in tackling the problem. More acceptable solution will be the use of the input in handwritten or preprinted forms. If handwritten character recognition is used and characters be inputted real time at the terminal, the system is an on-line recognition system. If preprinted character recognition is adopted, the system is said to be operating in off-line mode. There are recently quite a number of researches done

supporting the development of practical systems both of the type on-line and off-line.

It has been claimed that the input of on-line handwritten characters occurs less frequently and is in some sense less practical. Doubtless to say, Chinese characters are too difficult to write and it usually takes time for us to transcribe passages of Chinese characters. On the other hand, it is also one of our main reasons for using the computer in helping us to speed up the entire task of Chinese information processing which includes input of characters, process of information and output of the results.

The development of the off-line system for recognizing preprinted characters becomes more useful and important. It has been noticed that in many of the applications, we are just required to input pages of printed characters. Even though when we are writing our own passages, handwritten scripts can still be fed into the system for recognition if the discriminating power of our system is large enough. This means that the on-line system can actually be incorporated in the off-line system. As a result, the study of a practical off-line system is highly perferrable. However, one point we have to bear in mind is that an off-line system is not actually a "superset" of the on-line system. On-line systems usually possess extra discriminating power with the use of stroke order which off-line systems cannot have.

Traditional methods of thinning or stroke extracting are either not necessary or simple in on-line systems. Maybe the off-line system is more difficult to design due to the limited information available for the input.

§1.4 MULTI-STAGE CLASSIFIER VS SINGLE-STAGE CLASSIFIER IN CHINESE CHARACTER RECOGNITION

The design of a character recognition system is equivalent to designing a classifier which can successfully discriminate the characters. Classifier can either be single-staged or multistaged referring to classifiers which can successfully dicriminate a given character input to its target by one single step decision or by multiple steps of decision respectively.

Single-stage classifier is characterized by its simplicity and efficiency. Since only one step decision is needed, the output should inevitably comprise of a number of groups. In addition, the decision function must be complicated so that one single decision can complete the entire discrimination process. Besides the classifier will also be severely limited by the use of only some particular informations of the input. As a result, the discriminating power of such a classifier cannot be very good. Of course in practical situation such a single-stage classifier cannot attain the required level of performance.

Multistage classifier can be arranged in the form of a tree structure so that decisions are performed in a successive and hierachical manner. Since there are a number of steps to go before final decision can be made, different types of information and different discriminant functions can be used. As a result the discriminating power of a multistage tree classifier will usually be better than its single-stage counterpart. However, there is a problem of error propagation in multistage classifier which single-stage classifier does not have. We will discuss the problem in detail in later chapters.

§1.5 DECISION TREE

§1.5.1 Basic Terminology of a Decision Tree

Before we are going to describe the method of building the tree classifier in recognizing the Chinese character, some terminologies about the decision tree are mentioned here first. The decision tree here means the Direct Acyclic Graph as defined by S.Rasoul Safavian & David Landgrebe in [2]. As mentioned in [2], the Direct Acyclic Graph would satisfy the following properties

- There is exactly one node, called the root, which no edges enter. The root node contains all the classes.
- Every node except the root node has exactly one entering edge.

. There is a unique path from the root to each node. Refer to figure 1.1, several terms for describing tree are defined as follows

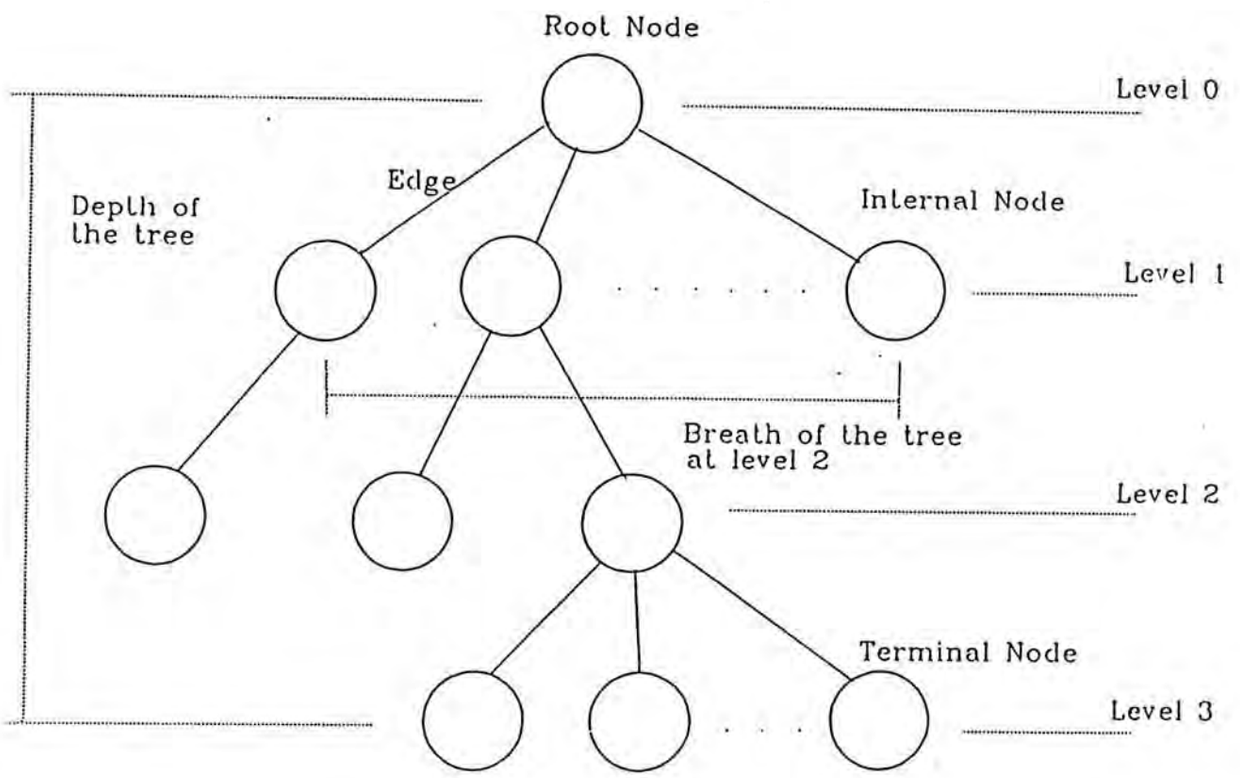


Figure 1.1 – Schematic Diagram of a Decision Tree

a. Edge

The edge is the order pairs (v,w) of the node. Node v is called the father of node w and node w is a son of v.

b. Root node

Root node is the node where no edge enters.

c. Leaf node (terminal node)

The node with no proper descendant.

d. Internal node

The nodes in the tree other than root node and leaf node.

e. Depth of a node

The depth of a node is the length of the path from root node to that node.

f. Ordered tree

An ordered tree is the tree in which the sons of each node are ordered.

g. Binary tree

A binary tree is an ordered tree such that

- Each son of a node is identified either as a left son or as a right son.
- No node has more than one left son or more than one right son.

h. Node with overlap classes

Two internal nodes contain a least one common class, then it is said that the node have overlap class.

i. Average depth of the tree

The average depth of the tree is the average number of layers from the root to the terminal nodes.

j. Average breadth of the tree

The average number of internal nodes in each level of the tree.

§1.5.2 Structure Design of a Decision Tree

Many methods about the optimal tree structure design has been proposed in [3] [4] [5] [6]. A summary has also been discussed in [2]. According to [2], the most common optimality criteria for tree design are

- Minimum error rate
- Min-Max path length
- Minimum number of nodes in the tree
- Minimum expected path length
- Maximum average mutual information gain

A basic problem with these methods is their computational unfeasibility, usually large amount of computational time and memory storage are required, which make it difficult to implement.

It is also shown by Hyerfil and Rivest in [7] that the problem of constructing optimal binary trees, optimal in the sense of minimizing the expected number of tests required to classify an unknown sample is an **NP complete** problem and thus very likely of non-polynomial time complexity.

It is classified in [2] that there are basically four categories for the construction of decision tree by heuristic method i.e. Bottom-up approach, Top-down approach, the Hybrid approach and Tree Growing-Pruning

approach. Each category is briefy described as follows:

a. Bottom-up Approach

In Bottom-up approach, a binary tree is constructed using the training set. Some distance measurement such as Mahalanobis distance between a priori defined classes are computed and in each step the two classes with the smaller distance are merged to form a new group. The mean and covariance matrix for each group are also computed from the training samples of classes in that group and the process is repeated until one is left with one group at the root. This tree building method makes use of the philosophy that more obvious discriminations are done first and more subtle ones at the later stages of the tree.

b. Top-down Approach

With the decision tree constructed in Top-down approach, an effective node splitting rule should be determined at each internal node of the tree to split the training set into two or more classes. A decision about which nodes are terminal is also being established. Each terminal node is assigned to a class label according to some criteria such as to minimize the mis-classification rate etc.

c. Hybrid Approach

In Hybrid approach, both Bottom-up and Top-down

approaches are used at the same time. The rationale for this method is that in Top-down approach such as hierarchical clustering of classes, the initial cluster centers and cluster shape information are unknown. These information can be provided by a Bottom-up approach. With this approach, the training set are considered using Bottom-up approach to come up with two clusters of classes. Then the mean and covariance for each cluster are computed. These information are then used in Top-down approach to come up with two new clusters.

d. Tree Growing - Pruning Approach

In Tree Growing - Pruning approach, the data set is divided into two approximately equal sized subsets and iteratively grow the tree with one subset and prune it with other subset. The role of the two subsets are interchanged successively.

A summary of some of the tree design methods in terms of the assumptions each approach makes, their performance criterion and special requirement can be found in table 1 at p.671 of [2].

§1.6 MOTIVATION OF THE PROJECT

We will concentrate on the development of a off-line printed character recognition system. We want to explore suitable

methods in contructing our multistage tree classifier. Our study will emphasize mainly on accuracy and efficiency. Since our target is only on printed characters, we are expecting to have a recognition system with high speed. In this way, classical recognition which involves processes of thinning and stroke extraction may not be appropriate in our study. Instead, we will treat the input character as a bitmap of 0 and 1 and try to detect suitable features which will successfully discriminate the character with one another. As a result, a number of questions has to be explored in our current study:

1. Which features should we use in the discrimination process?
2. Which methods should we adopt in constructing the multistage tree classifier?
3. What kind of tree architecture should we employ? Is there any distinguished advantages towards the use of such tree architecture?
4. How can we evaluate the effectiveness of the classifier? How can we quantify such effectivness?
5. Is there any possible improvement in the classifier thus developed?

In the chapters which follow, the above questions will be answered and practical system of Chinese character recognition system will be developed.

§1.7 OBJECTIVE OF THE PROJECT

The objective of the current study is to develop a practical off-line printed Chinese character recognition system with high recognition rate and reasonable speed. Multistage tree classifier will be used. We will focus on the methods employed in constructing the classifier. The methodology will be implemented in computer using suitable samples of Chinese characters as testing data. The system should be practical and hopefully it can be implemented in ordinary microcomputer system.

§1.8 DEVELOPMENT ENVIRONMENT

Since the system is to be practical and can be implemented in microcomputer system, the development environment is chosen to be in ordinary IBM 386 systems. The C programming language is used as the development tools and programs have been written to implement the entire idea of character recognition.

Besides, we have been continuously using the Eten Chinese system for reference. The Eten Chinese system has provided good resources of bit pattern of printed Chinese characters and lookup tables for input schemes. These are indispensable materials towards the development of the current system.

CHAPTER 2 -- APPROACH 1 UNSUPERVISED LEARNING

§2. APPROACH ONE - UNSUPERVISED LEARNING

Unsupervised learning actually refer to the case where no prior knowledge of the class membership of the objects in the training sample is known to the system. The objects are classified into clusters by some intrinsic likeliness of the objects themselves. There is no absolute measurement of likeliness, and the correctness of the classification. However, the nonparametric nature renders the process to be highly flexible and thus versatile.

The details study of the use of the unsupervised learning approach in printed Chinese character recognition is given by my project partner Mr. Sin Ka Wai and therefore will not be described in details here. In this chapter, a brief overview is being given in order to maintain the completeness of the thesis and the whole analysis can be found in appendix A2.

1. Several useful feature extraction methods for the input bit pattern of the character are first developed.

2. Different unsupervised clustering algorithms are derived. One clustering algorithm is chosen to cluster the given training set into different groups based on the use of one particular feature. This step will be repeated for all other features which have been extracted from the characters.

3. A decision table is then formed.

4. Based on the decision table just constructed, the optimum decision tree is then built.

There are some problems associated with this approach in the training up of the classifier since practically the algorithm descirbed in (4) cannot be implemented computationally. This belongs to the class of NP complete problems. Details of the discussion and reasons will be given in appendix A.2.

CHAPTER 3 -- APPROACH 2 SUPERVISED LEARNING

§3 APPROACH TWO - SUPERVISED LEARNING

§3.1 IDEA

As mentioned in section 2.6 that the tree classifier with unsupervised learning has several implementation difficulties. A lot of time is required before the decision functions can be built. It would make the classifier inflexible especially when some new characters are added to the training character set.

In view of this difficulty, a tree classifier with supervised learning approach is suggested. It can maintain the advantage of a tree classifier in reducing the complexity of the decision making and at the same time improve the flexibility and decrease the training time required.

The idea of the supervised learning approach is to make use of the similarity properties of 3 corner code input method and the corner feature extraction method (refer section 2.2.6 for details).

In 3 corner code input method, the Chinese character is divided into four equal parts i.e. 4 corners. Only 3 corners are considered i.e. top left, top right and bottom left. Based on the key strokes properties of that corner, an 3 corner code 00 - 99 would be assigned. In other words, those Chinese characters with the same corner key strokes

properties would have the same 3 corner code assigned. The details code assignment method would be described later in the section 3.2.

Similarly, with the corner feature extraction method as mentioned in section 2.2.6, those Chinese character with the same corner key stroke will have the same corner features extracted provided that the same character font is considered.

Because of the above mentioned similiarity between the 3 corner code and corner feature, the 3 corner code is used as a reference to assist the training of the classifier with the corner features.

There are a lot of tree structure such as binary tree and multi-path tree (refer section 1.5.2). Binary tree is suggested because it is much simpler. Linear discriminant function usually used in statistics can be used at each decision node to divide the feature space into two regions.

It seems that everything is fine. However, the 3 corner codes can only provide the 100 classes information about the training characters instead of two classes. The problem is how the 100 classes information that the 3 corner code provided can be used to determine the linear discriminant function? To tackle this problem, an Hybrid tree building approach described in section 1.5.2 is adopted. The details of the training and tree building will be described in

subsequent sections. The following is a brief description of the necessary steps.

For each corner,

Step.1: Group the training characters into 100 classes according to the 3 corner code.

Step.2: Determine the mean of each class based on the corner features.

Step.3: Determine the class discriminant functions based on the Mahalanobis distance

Step.4: Re-substitute the original training characters into the discriminant functions. The degree of misclassification will be summarized in a misclassification table.

Step.5: An heuristic will be introduced here to combine those 100 classes into two groups. The philosophy of the grouping is to minimize the degree of misclassification in step 4.

Step.6: Determine the corresponding resulting linear discriminant function after grouping.

Step.7: Determine the number of distinct 3 corner codes denoted by N for each group.

Step.8: Repeat step 1 - 7 for each decision node with N classes instead of 100.

Step.9: With the above steps, a decision tree for each corner of the character would be formed. There will be a total of three decision trees since we will

consider three corners.

Step10: A terminal code will be assigned for each terminal of the decision tree.

Step11: Repeat step 1 - 10 for the other corners.

There will be three terminal codes, one terminal code per decision tree. The terminal codes will be combined to form the character code for the training character. The philosophy of this character code formation is the same as that of the 3 corner codes mentioned in the next section.

§3.2 THE 3 CORNER CODE

The 3 corner code is developed from the traditional 4 corner code commonly used in Chinese dictionary and in library where Chinese books are categorized by the 4 corner codes. It is used as one of the traditional input scheme in some of the Chinese computer systems. The 3 corner coding scheme tries to exploit the shape at the three corners of a given Chinese character and encode these shapes using appropriate number codes. The first corner refers to the top left hand corner, the second corner refers to the top right hand corner while the third corner refers to the bottom left hand corner. There may be some deviation for the third corner in the circumstances if the shape at the bottom left hand corner has already been encountered and incorporated in the first corner code. In such case, the bottom right hand

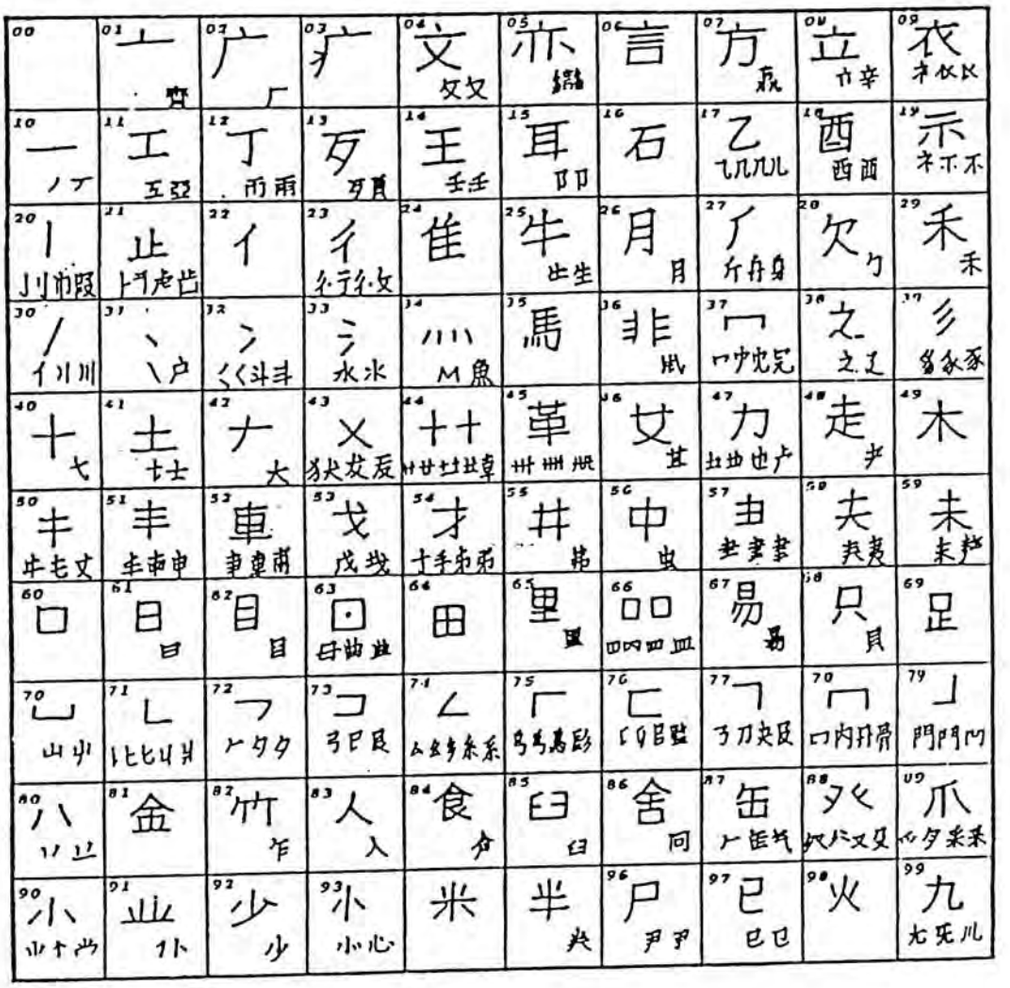


Figure 3.1 - Three Corner Coding Table

As a result the combined 3 corner code is a six digit number theoretically ranging from 000000 to 999999. Of course, not all the codes in this range are feasible and many of them does not exist. Besides, one 3 corner code may

not uniquely determine one single Chinese character. There may exist different characters having the same 3 corner code although such case does not occur very frequently.

The 3 corner code input scheme is not actually popular. Despite its unpopularity, the codes are useful in adopting it as the guidelines in the approach of supervised learning. Moreover, these codes are also commercially available in the lookup table for the 3 corner code input scheme in Eten Chinese system. This look up table has been decoded for our later use.

§3.3 FEATURE EXTRACTION AND SELECTION

As explained in section 3.1, the corner properties of 3 corner code will be used as the guideline to develop the tree classifier. Therefore the corner feature as mentioned in section 2.2.6 will be used.

§3.4 DECISION AT EACH NODE

§3.4.1 STATISTICAL LINEAR DISCRIMINANT ANALYSIS

Since statistical linear discriminant function will be used, a brief description of constructing this function from the training sample will be given. More detailed discussion on statistical discriminant analysis [14] will be included in the appendix.

Consider the problem of classifying an observation vector x into one of k groups (or populations) Π_1, Π_2, ..., Π_k. When the parameters of the distribution in the kth populations are unknown, the usual procedure in classical discriminant analysis is to estimate them from training samples x_{ij}, j=1,...,N from each of the populations Π_i, i=1,...,k. Let

$$\bar{x}_i = \sum_{j=1}^{n_i} \frac{x_{ij}}{Ni} \quad \text{and} \quad S_i = \left[\sum_{j=1}^{n_i} \left(x_{ij} - \bar{x}_i\right)\left(x_{ij} - \bar{x}_i\right)' \right] / n_i$$

where $n_i = N_i - 1$ be the sample mean and co-variance matrix corresponding to the training sample from Π_i, and

$$S = \left(\sum_{i=1}^{k} n_i S_i \right) / \left[\left(\sum_{i=1}^{k} n_i \right) - k \right]$$

be the pooled co-variance matrix. Here we estimate the population covariance matrix Σ by S. The usual, estimative, approach to discriminant analysis is to replace the parameters in the classification rules given in the appendix by their sample estimates. Applying this approach yields the sample based classification rule:

Assign x to Π_i if

$$(\bar{x}_i - x)'S^{-1}(\bar{x}_i - x) < (\bar{x}_j - x)'S^{-1}(\bar{x}_j - x)$$

$$\forall j = 1,...,p;\ j \neq i$$

The generalized discriminant function will then be given

by $$y_i = a_i'x + c_i, \quad \text{where}$$
$$a_i = S^{-1}\bar{x}_i \quad \text{and}$$
$$c_i = -\frac{1}{2}\bar{x}_i S^{-1}\bar{x}_i$$

The term $(\bar{x}_i - x)'S^{-1}(\bar{x}_i - x) = D^2(\bar{x}_i, x)$ is the sample Mahalanobis distance between $\bar{x}_i$ and x and is an estimate for the population Mahalanobis distance $\Delta^2(\mu_i, x)$. If unequal prior probability is assumed, then the corresponding c_i will be given by

$$c_i = \ln p_i - \frac{1}{2}\bar{x}_i S^{-1}\bar{x}_i$$

instead of the previous one. Of course p_i may not be known and is usually estimated by $N_i / \sum_{i=1}^{k} N_i$.

§3.4.2 Optimization of the number of misclassification

Optimum Grouping Analysis

The probability of misclassification under any classification rule is a measure of the expected performance of that rule when classifying observations of unknown origin. This probability can be estimated through classifying each member of the samples from Π_i according to the discriminant function developed in the last section. The classification table thus obtained forms our basis for further analysis.

We define optimum grouping to be that grouping under which the total number of misclassification to be resulted from

the discriminant analysis will be minimized. To start the analysis, a preliminary multi group discriminant analysis is performed based on the corner code values. Within this framework, the classification table will be obtained resembling the following:

To group / from group	1	2	3	4 . . . . .
1	2	0	0	1
2	0	1	2	5
3	0	1	5	0
4	1	4	5	8 . . . . .
.			.	
.			.	

Our idea of optimum grouping is to choose one grouping here such that when these groups are merged together and treated as one group while all the others remaining treated as another, the total number of misclassification is minimum. Take an example that the classification matrix is given as

2 0 0 1

0 1 2 3

0 1 5 0

1 4 5 8

where the group numbers are 1, 2, 3 and 4. Suppose group 1 and 2 are merged. Then the new classification table becomes

To group / from group	1 & 2	3 & 4
1 & 2	3	6*
3 & 4	6*	18

* misclassified

Here the total number of misclassification is 6 + 6 = 12. However, if groups 2, 3 and 4 are merged together, the new classification table becomes:

To group / from group	1	2, 3 & 4
1	2	1*
2, 3 & 4	1*	29

* misclassified

Hence the total number of misclassification decreases to 2. Our objective in the study is to find a way to obtain such an optimum grouping when we are given the classification table.

Formulation of Problem

Let N be the number of groups under consideration. Therefore the dimension of the classification table is N by N. Write the group label for the ith group as G[i]. Specifically, if the ith group represents the group of data having corner code equal to 60, then G[i] = 60.

Suppose $(c_{ij})_{NxN}$ = classification matrix

where c_{ij} = number of elements from group i classified to group j

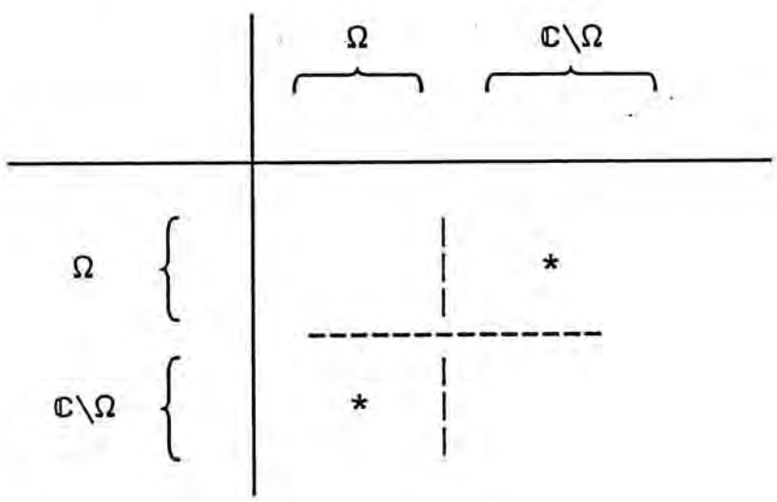
$\mathbb{C}$ = set containing all the corner codes

$$= \bigcup_{i=1}^{N} \{ G[i] \}$$

Ω = a certain grouping

Of course, $\Omega \subset \mathbb{C}$.

If we merge the data to two groups designated by Ω and $\mathbb{C}\backslash\Omega$, then after changing the rows and columns of the original classification matrix, a new matrix can be formed where the first appearing codes are from Ω while the rest are from $\mathbb{C}\backslash\Omega$, i.e.



The total number of misclassification will be represented by the sum of all the entries in regions marked with (*).

Denote $M(\Omega)$ = total number of misclassification obtained by the partition specified in grouping Ω

Then, $$M(\Omega) = \sum_{\substack{G[i]\in\Omega \\ G[j]\in\mathbb{C}\backslash\Omega}} c_{ij} + \sum_{\substack{G[j]\in\Omega \\ G[i]\in\mathbb{C}\backslash\Omega}} c_{ij}$$

$$= \sum_{G[i]\in\Omega} \sum_{G[j]\in\mathbb{C}\backslash\Omega} c_{ij} + \sum_{G[i]\in\mathbb{C}\backslash\Omega} \sum_{G[j]\in\Omega} c_{ij}$$

Hence our objective is to find Ω which will minimize $M(\Omega)$. There does not seem any possible methods in finding Ω and exhaustive search appears to be the only solution.

Analysis of the Complexity of Exhaustive Search

We do not know the cardiality of Ω and since $|\mathbb{C}| = N$, we have $0 < |\Omega| < N$. By the symmetry of the problem, $|\Omega| > N/2$ will imply $|\mathbb{C}\backslash\Omega| < N/2$, hence an equivalent problem will be obtained which aims at finding $\mathbb{C}\backslash\Omega$. As a result, we can restrict our search for $0 < |\Omega| < N/2$, i.e. $1 \leq |\Omega| \leq \lfloor N/2 \rfloor$ where $\lfloor x \rfloor$ denotes the largest integer smaller than or equal to x.

For each value of $|\Omega|$, we want to find the optimum Ω so that $M(\Omega)$ is minimized. Since there are N different elements in $\mathbb{C}$, the total number of different combinations will be amounted to C_k^N where k is the cardiality of Ω. Assuming a constant amount of computations α is necessary in each distinct combination for Ω, then the total number of complexity of the search will be given by

$$\sum_{k=1}^{\lfloor N/2 \rfloor} C_k^N \, \alpha = \alpha \sum_{k=1}^{\lfloor N/2 \rfloor} C_k^N$$

$$\simeq \frac{\alpha}{2} \sum_{k=0}^{N} C_k^N$$

$$= \frac{\alpha}{2} \cdot 2^N$$

$$\simeq \left(2^{N-1} \right) \cdot \alpha$$

which shows that the job is actually not a polynomial time problem and belongs to the class of NP complete problems. Rough estimation reveals that even if $\alpha = 10^{-6}$ sec, N = 100, the time required for finishing the search will be given by

$$\begin{aligned} 10^{-6} \times 2^{N-1} &= 10^{-6} \times 2^{99} \\ &\simeq 6.34 \times 10^{23} \text{ sec} \\ &\simeq 2 \times 10^{16} \text{ years} \end{aligned}$$

which is an astronomically long period of time.
The above analysis concludes that the finding of the true optimum is not really possible and other approaches should be adopted so that some "nearly" optimum values can be obtained. To tackle the above problem, the following heuristic has been proposed.

Heuristic

The basic assumption is to assume that if Ω is the optimum grouping thus far, then $M(\Omega \cup \{G[k]\})$ will be minimized where $M(\{G[k]\})$ is the minimum for all $G[i] \in \mathbb{C} \backslash \Omega$. This assumption is of course not really true. Although such heuristic cannot give the optimum solution to our problem, we can still arrive at a "nearly" optimum solution at a reasonably short period of time with the algorithm. The complexity analysis of this heuristic follows.

There are $\lfloor N/2 \rfloor$ passes for us to complete the algorithm

for $k = 1, 2, \ldots, \lfloor N/2 \rfloor$ where k is the cardiality of Ω. At each pass, our job is just to find the G[k] such that $M(\{G[k]\})$ is the minimum among all $G[i] \in \mathbb{C} \backslash \Omega$. Adopting our previous argument,

time complexity $\leq \lfloor N/2 \rfloor \cdot N \alpha \simeq \propto N^2/2$

which can be solved within polynomial time. Take N = 100, $\alpha = 10^{-6}$ sec, the time required will be

$$100^2 \frac{1}{2} 10^{-6} \text{ sec} = 5 \times 10^{-3} \text{ sec}$$

which is an extremely short period of time.

Implementation of the Heuristic

To formalize our discussion, we will first define some of our terminologies used. We define a k-grouping to be a combination of k groups of elements merged together. Adopting our previous notations, we have

N = total number of groups under consideration

G[i] = the group label of the ith group

$\mathbb{C} = \bigcup_{i=1}^{N} \{ G[i] \}$

Ω = set representing a certain k-grouping where $|\Omega| = k$.

C = $(c_{ij})_{NxN}$ = the classification matrix where c_{ij} is the number of elements from group i classified to group j

To facilitate further formulation, the following

definitions will also be included.

A[k] = the group added to the "optimum" k-grouping to make up the "optimum" (k+1)-grouping

Here "optimum" refers to the optimum claimed by the heuristic assumption. As a result, the "optimum" k-grouping will be given by the groups A[0], A[1], ..., A[k-1] and we will denote this as

$$A_k = \bigcup_{i=0}^{k-1} \{ A[i] \} \subset \Omega$$

$$G = \{1,2,\ldots,N\}$$

$E[A_k]$ = the optimum number of misclassification given by the k-grouping of groups as represented by A_k.

To begin, we have to calculate E[S] for all S where |S| = 1, i.e. for all singleton set S. For simplicity, if S = {i} then E[S] will be denoted as E[i] where

E[i] = the number of misclassification induced if we partition the group space into two subsets where one subset is the ith group while the other comprises of all the groups other than the ith one.

Notice that

E[i] = sum of all entries of the shaded part in matrix C of figure 3.2a.

$$= \sum_{\substack{j=1 \\ j\neq i}}^{N} c_{ij} + \sum_{\substack{j=1 \\ j\neq i}}^{N} c_{ji}$$

$$= \sum_{j=1}^{N} (c_{ij} + c_{ji}) - 2\, c_{ii} \qquad (3\text{-}1)$$

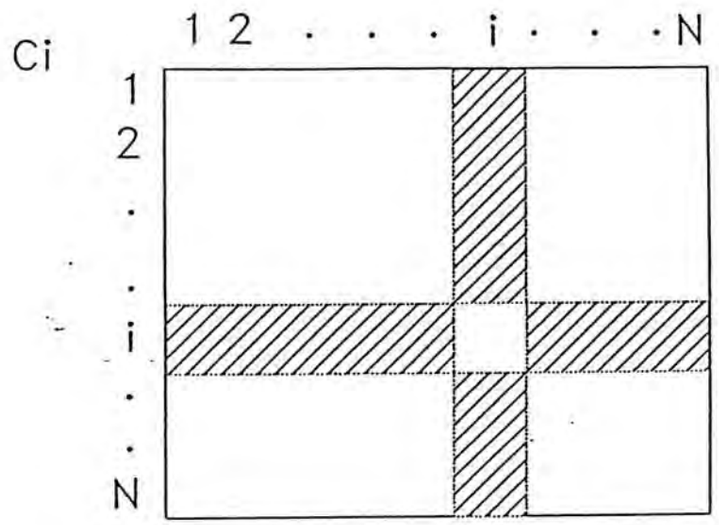


Figure 3.2a

To facilitate further analysis and subsequent calculation, the following lemma is formulated.

Lemma : $E[A \cup \{i\}] = E[A] + E[i] - \sum_{j \in A} (c_{ij} + c_{ji})$ where $i \notin A$.

Proof :

Suppose we rearrange the rows and columns of C so that all the groups represented in the set A are adjacent to one another and i is just next to the clusters of groups represented by A. The scenario can be illustrated by the figure 3.2b.

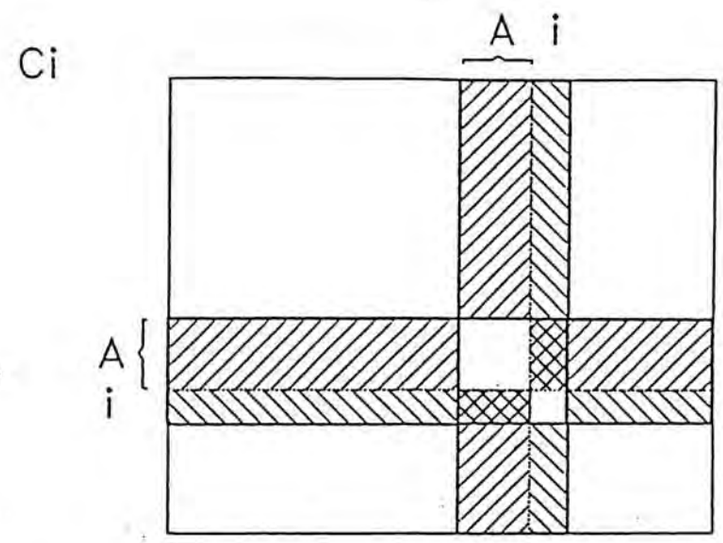


Figure 3.2b

E[A] = sum of all entries in the ▨ region

E[i] = sum of all entries in the ▨ region

When the groups in A and the ith group are merged,

E[A U {i}] = sum of all entries in the ■ region.

Notice that from the diagram,

$$E[A \cup \{i\}] = \text{▨ region} + \text{▨ region} - \text{■ region}$$

$$= E[A] + E[i] - \sum_{j \in A} c_{ij} - \sum_{j \in A} c_{ji}$$

$$= E[A] + E[i] - \sum_{j \in A} (c_{ij} + c_{ji})$$

Q.E.D.

The above lemma indicates how the total number of misclassification can be calculated when a particular

group is being added to the existing grouping represented by the set A. With this lemma, the entire problem of finding the optimum grouping can be formulated, using the approach of dynamic programming, as follows.

Dynamic Programming Formulation

Recall assumption:

If A_k is the optimum k-grouping then the optimum k-grouping should contain all the groups residing in A_{k-1}.

1. Calculate all E[i] for i ϵ G by the formula (3-1).
2. A[0] = j

   A_0 = {j}

   $E[A_0]$ = e

   where $e = \min_{i \epsilon G} E[i]$

   and j takes the value of i chosen which corresponds to that of e.
3. For k = 1, 2, ..., N-1

$$E[A_k] = \min_{i \epsilon G \backslash A_k} \left\{ E[A_{k-1}] + E[i] - \sum_{j \epsilon A_{k-1}} (c_{ij} + c_{ji}) \right\}$$

   A[k] = the value of i chosen

   $A_k = A_{k-1} \cup \{A[k]\}$

In principle, we can calculate all the optimum k-groupings from the above heuristic for all values of k ϵ G, i.e. k = 1, 2, ..., N. In practice, we want to arrive at a certain grouping which will partition our

data set into two halves with roughly the same size so the multistage tree developed will be more balanced. Then what value of k should we choose to stop our iteration. One obvious choice will be N/2. In such case, we are aiming at partitioning all the groups into two equal halves which will minimize the overall number of misclassification with such grouping. However such choice does not necessary guarantee that the data set will be divided into two exactly equal halves and will result in the formation of an unbalanced tree classifier. As a result, a better criterion is to stop the iteration at a value of k, with which although the partitioning of the groups will not be exactly two equal halves, the result of grouping will divide our data set into two approximately equal subsets.

Let C[i] = total number of elements in group i, $i \in G$

Then $$C[i] = \sum_{j=1}^{N} c_{ij} = \sum_{j=1}^{N} c_{ji}$$

Modified Algorithm

1. Calculate all E[i] for $i \in G$ by the formula (1).
2. A[0] = j

   A_0 = {j}

   $E[A_0]$ = e where $e = \min_{i \in G} E[i]$

   and j takes the value of i chosen which corresponds to that of e.

3. $k = 1$

T = total number of data = $\sum_{i=1}^{N} C[i]$

Repeat

$$E[A_k] = \min_{i \in G \setminus A_k} \left\{ E[A_{k-1}] + E[i] - \sum_{j \in A_{k-1}} (c_{ij} + c_{ji}) \right\}$$

$A[k]$ = the value of i chosen

$A_k = A_{k-1} \cup \{A[k]\}$

Until

$$\sum_{j \in A_k} C[j] > \frac{T}{2}$$

§3.5 IMPLEMENTATION

The idea of supervised learning approach has been described in section 3.1 and the theory of the corner feature extraction and clustering have also been illustrated in section 3.3 and section 3.4 respectively. In this section, the implementation details such as the creation of testing samples, the building of the decision tree and the determination of decision functions will be mentioned.

§3.5.1 Training Data

The 24 x 24 dots standard character set (stdfont.24) in Eten Chinese System is used to train up and test the performance of the tree classifier. There is no distinguished advantage of using this character set. The reasons of adopting this character set as the testing sample are

a. It is simple and easy to obtain. No extra character reading and prepossessing software are required and neither of them is the main objective of our project.
b. The standard character set consists of about 13,000 Chinese characters which has already covered the commonly used Chinese character and difficult Chinese character.
c. Big 5 internal code is adopted in Eten Chinese system which is very popular nowadays in Hong Kong and Taiwan. Tree classifier based on Big 5 internal coding can interface with Eten Chinese system more easily.
d. Many commonly used Chinese font types such as Ming, Sung etc are available in Eten Chinese system. These character fonts can be used to test the classifier's performance or used in multi font training and recognition.
e. The lookup table for the 3 corner coding system is available in Eten Chinese system which can directly match with the training character.

To prepare the training data, the corner features for each corner of the sample characters are extracted with the method described in section 2.2.6. As 3 corner code is used as a mean for supervised learning in our tree classifier (section 3.1), the corner features are then combined with the corresponding corner code (refer section 3.2 for details of 3 corner code) and Big 5 internal code.

Finally, three training data files, one data file per each corner, are formed. Each training file consists of around 13,000 character records. Each character record has 8 corner features, the corner code and the Big 5 internal code. The format of the character record is illustrated in figure 3.3.

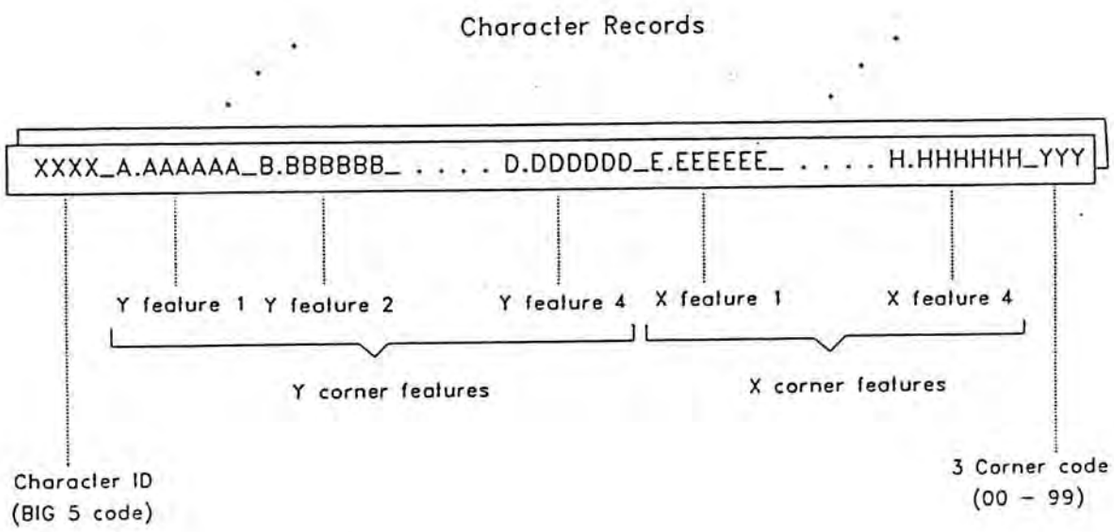


Figure 3.3 Format of the character records

§3.5.2 Clustering with the use of SAS

After the training data files have been prepared, they can be used to train up the classifier at each node of the tree with the clustering algorithm and linear discriminant function as described in section 3.4.

It will take a long time and effort to develop the software for clustering and tree building. In order to reduce the development effort during the early testing stage, a statistical package called SAS was used. This statistical package has built-in library routines to calculate the Mahalanobis distance and covariance matrix which are essential in distinguishing at each internal nodes.

To perform the clustering at each internal node with the corner file using SAS, the following procedures is adopted. The corner file here refers the training data set in each internal node as mentioned in section 3.5.1.

For each corner file

a. Group the characters according to the 3 corner codes of each character record. The number of classes would depend on the content of the corner file.
b. A SAS script program (CLUSTER.SAS) is written and to be run at SAS to carry out the following sub-tasks. The details of "CLUSTER.SAS" is shown in figure 3.4.

   - calculate the mean of each class and the covariance matrix between classes based on the corner features.
   - Determine the class discriminant functions calculat ed based on the Mahalanobis distance between classes

- Re-substitute the original training characters (corner file) into the discriminant functions. A statistic is made to determine the degree of misclassification within the classes.
- A table of misclassification is then printed out. Part of the misclassification report is listed in appendix.

```
data _cfont_;
  infile "sas_dat1.dat";
  input id $ f1-f8 group;
  run;
%macro cluster(u,l);
  %do i=&u %to &l;
    data test;
      set _cfont_;
      if not(group=%eval(&i)) then group=999;  * 999 = others;
      run;
    proc discrim data=test;
      class group;
      var f1-f8;
      id id;
      run;
  %end;
%mend cluster;
****;
*  Execute the macro;
****;
%cluster(0,99);
```

Figure 3.4 Listing of CLUSTER.SAS

c. Reformat the misclassification report manually so as to form a misclassification table between the classes. Part of the misclassification table is shown in figure 3.5.

d. A C program "SPLIT.C" is written to read in the modified misclassification table. The heuristic for splitting described in section 3.4.2 is then used to split the original training data set (corner file) into two groups.

e. Afterwards, another SAS script program "TREE.SAS" is written to determine the linear discriminant function for that internal node based on the splitting result in procedure d. Part of the discrimination report is listed in appendix for reference and the "TREE.SAS" is also shown in figure 3.6.

```
data _cfont_;
  infile "c:\tc\e-10.dat";
  input id $ f1-f8 g;
  flag=1;
  group=1;
  if g=54 or g=45 or g=14 or g=48 or g=59 or g=94  then flag=0;
  if g=55 or g=50 or g=58 or g=13 or g=36 or g=49 then flag=0;
  if g=45 or g=90 or g=25 or g=29 or g=20 or g=34 then flag=0;
  if g=74 or g=83 or g=88 or g=3  or g=12 or g=40 then flag=0;
  if g=71 or g=80 or g=93 or g=7  or g=37 or g=41 then flag=0;
  if g=32 or g=1  or g=77 or g=39 or g=46 or g=72 then flag=0;
  if g=84 or g=81 or g=87 or g=44 or g=52 or g=78 then flag=0;
  if g=86 or g=97 or g=23 or g=43 or g=24 or g=38 then flag=0;
  if g=92 or g=33 or g=6  or g=19 or g=8  or g=22 then flag=0;
  if g=28 or g=47 or g=95 or g=16 or g=70 or g=11 then flag=0;
  if g=10 or g=57 or g=73 or g=98 or g=21 or g=42 then flag=0;
  if g=30 or g=51 or g=75 or g=53 or g=9  or g=89 then flag=0;
  if g=67 or g=66 or g=64 or g=65 or g=85 then flag=0;
  if flag=0 then group=0;
  run;
  proc discrim data=_cfont_;
    class group;
    var f1-f8;
    id id;
    run;
```

Figure 3.6 Listing of the TREE.SAS

f. The discrimination report would be generated by SAS but it cannot be used directly. It will be reformatted manually to erase all the dummy information as shown in figure 3.7. Note that every nodes will be identified by an unique label during the character recognition stage.

-13.00569	-12.75579
8.31453	8.02564
11.59937	11.34153
9.95468	10.53099
6.12050	4.94885
14.16205	12.86861
2.04583	-0.71325
25.90327	31.95018
-0.30411	-0.08740

Figure 3.7 Example of the Linear Discriminant Function

§3.5.3 Building the decision trees

The clustering of the training data set at each internal node of the decision tree has already been mentioned in section 3.5.2. In this section, the method of building the decision tree is discussed. Binary tree structure is used as it is simpler and easier to implement and it is logically equivalent to the m-ary tree counterpart.

a. At the root node of the tree, we will use the training data set created in section 3.5.1. There will be 100 three corner classes at the root node.

b. Use the procedures described in section 3.5.2 to split the training data set into two subsets. These are the sons of the root node. The format of the training data subset is the same as the original training data set. The decision function (a linear discriminant function) for the root node is then stored and labeled according to its position.

c. Repeat procedure b for each internal node until the terminal node of the tree is reached. The number of 3 corner classes at each internal node would depend on the splitting of the training character.

d. For each terminal node of the tree, a code called corner code is assigned for identification.

e. Three separate trees protocol is adopted in this experiment. The entire procedure will be repeated for other two corner files.

f. As a result, there are three terminal codes (one terminal code per decision tree) for each training Chinese character. A code table with the corresponding corner code and BIG 5 internal code is formed.

g. To test the performance of the classifier, a C program "CHDIS_1.C" is developed to carry out the recognition process. The main function of the program includes

. The discriminant functions of each internal nodes will be stored and used in the recognition algorithm.

. Extract the corner features of an input unknown * To invoke this program in sas, just type character.

- The corner features extracted will be passed through the three decision trees to get the corresponding corner codes for that unknown character. The 3 corner codes will be combined and this single code is used to identify the corresponding BIG 5 code. In this way, the Chinese character is recognized.

Figure 3.8 summarizes the clustering and tree building procedure

In our classifier, only $(13,000)^{1/3}$ or approximately 32 terminals per decision tree is sufficient to identify the 13,000 training characters as 3 separate decision trees methodology is adopted. 32 or 2^5 terminals means 5 levels in binary decision tree structure. The tree is not perfectly balanced due to the structure of 3 corner codes.

It is noticed that three 32 terminals in the binary decision tree has a total of 32x32x32 character codes which is much greater than 13,000. As a result, many holes (i.e. character code which do not correspond to any Chinese character) are formed.

If 4 levels are used instead of 5 levels, a total of 2^4 = 16 terminals per decision tree which corresponds to 16x16x16 = 4096 distinct character codes outputted. It means that there will be an average of 3 characters for each character code.

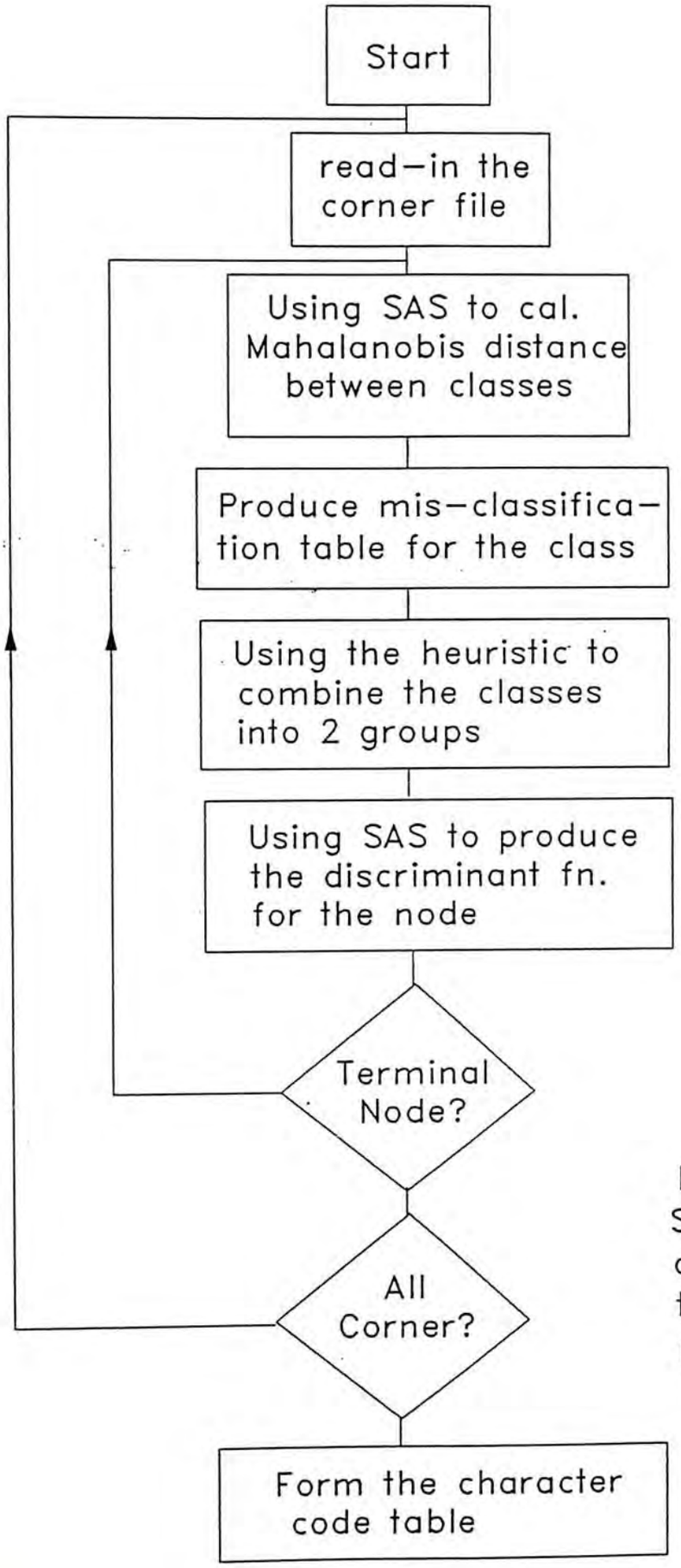


Figure 3.8
Schematic flow of clustering & tree building procedure

To solve the problem of the existence of holes, back tracking is introduced. The idea of back tracking will be discussed in section 5.4. In addition to back tracking method, the holes can also be reduced by using other tree architecture such as single decision tree instead of three decision trees. The details of this method is illustrated in section 5.6 and 5.7.

§3.5.4 Description of the classifier

After completing the training process, we will have obtained

1. the discriminant functions of all the nodes of the three separate trees which correspond to the three corners;
2. the code corresponding to every character in the training data set.

To facilitate the construction of the classifier, we will store all the discriminant functions in the program. The number of codes is very large and will be stored in a file in the format:

```
012011  <----  code of the first character in stdfont.24
020306  <----  code of the second character in stdfont.24
120314
   :
030915  <----  code of the ith character in stdfont.24
   :
```

The steps to be performed by the classifier for the purpose of recognizing an inputted passage of characters will be given as:

a. Read in the codes for each characters from file. These codes will be sorted in ascending order by the use of radix sort and stored in an array so that subsequent searching using binary search can be performed. Besides the big 5 codes corresponding to each characters have also been calculated for later reference.

b. The bit pattern of an inputted character is read into the system.

c. The three corner feature values will be extracted.

d. Using the discriminant functions stored and the feature values just calculated, the code for the character will be computed.

e. This code will be used as the key for searching in the array formed in (1). Binary search will be used so that only a few comparisons are needed even with a sample of size around 13000. If there is a match, the corresponding big 5 code will be taken. This is the big 5 code of the resulting character being recognized.

f. The character with this big 5 code will be displayed.

g. The entire process is repeated for another inputted character.

§3.6 EXPERIMENTS AND TESTING RESULT

§3.6.1 Performance parameters being measured

To access the performance of the classifier, several performance parameters are measured. For easy understanding, the tree classifier can be considered as a single decision node with many terminals. Each terminal node corresponds to a single character code with either one Chinese character, no Chinese character (i.e. holes) or more than one Chinese characters.

As illustrated in figure 3.9, a 5 level decision tree can have a maximum of 32x32x32 = 32768 terminal nodes. In our test, the upper bound for the number of terminal nodes will be 13,000. However, some characters may appear in groups residing in the same terminal node. As a result, the actual number of terminal nodes should be less than this upper bound. Refer to figure 3.9,

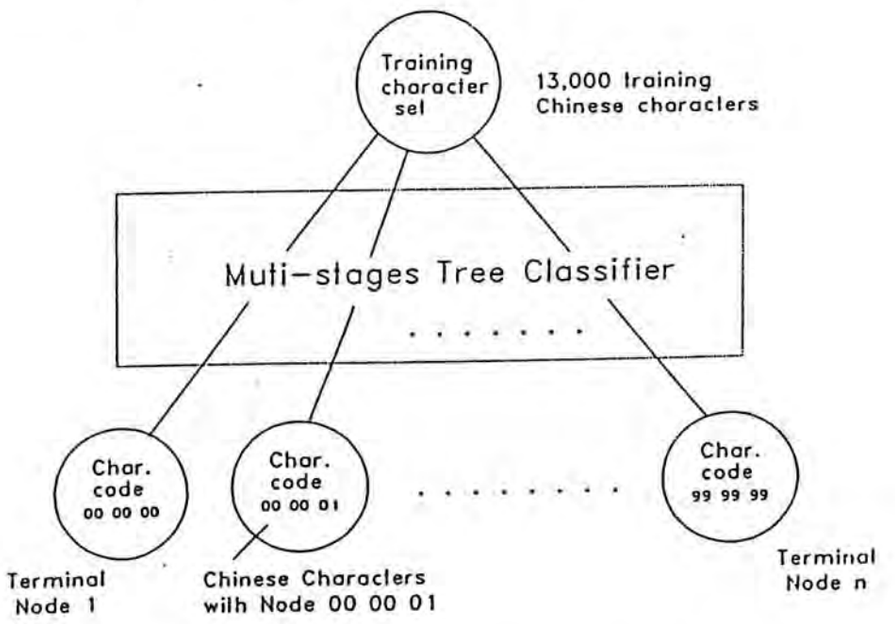


Figure 3.9 Multi-stages Tree Classifier

Let N be the number of terminal nodes where $N \leq 13000$

n be the total number of Chinese character to be recognized

n_i be the number of characters at terminal node i where $1 \leq i \leq N$

The following performance measures are proposed :

a. The mean and standard deviation of the number of Chinese character per character code

The mean number of the character per terminal node is

$$MEAN = \frac{1}{N} \sum_{i=1}^{N} n_i$$

The Standard Deviation (SD) of the number of characters per terminal node is

$$SD = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (n_i - MEAN)^2}$$

b. The coefficient of variation, i.e. MEAN / SD

c. The entropy reduction

Let P_j = the probability of occurrence of character j, where $1 \leq j \leq n$

$$\sum P_j = 1 \qquad 0 \leq P_j \leq 1, \quad \forall\ 1 \leq j \leq n$$

Based on the Shannon's entropy calculation as proposed by Wang and Suen in [15], the entropy of these n characters is

$$Hn(P1, P2, P3 \ldots Pn) = -\sum_{j=1}^{n} P_j \log_2 P_j$$

For N terminal nodes, the reduction of entropy due to the classifier is

$$\Delta H = -\sum_{k=1}^{n} P_k \log_2 P_k - \left(-\sum_{i=1}^{N}\left(-\sum_{j\in\Omega_i} P_j\right)\log_2\left(-\sum_{j\in\Omega_i} P_j\right)\right)$$

d. The accuracy of the classifier

An input unknown character is said to be accurately recognized if the expected Chinese internal code can be found at the resulting terminal node. The accuracy of the classifier is the percentage of the characters correctly recognized over the total no. of characters being tested.

$$\text{Accuracy} = \frac{\text{Number of characters correctly recognized}}{\text{Total number of characters being tested}}$$

Note that even when the internal code of the testing character is not uniquely determined, it is also counted as accurately recognized if the expected internal code is residing in the group of codes in the terminal node.

e. Recognition Speed

The recognition speed measured in our experiments is the total time required to give the deduced internal code from an unknown character bit pattern. It includes the time for corner features extraction, character

codes formation and the time for other improvement such as context consideration, back tracking etc. The average recognition speed per character for the testing sample will be calculated.

§3.6.2 Testing by resubstitution method

One direct and easy way to estimate the performance of the classifier is the resubstitution method. In this method, the original design/training samples are resubstituted into the classifier for testing. This method has the advantage of maintaining the size of the design set. However, the independent issues between the design character set and test character set is ignored and is often criticized for being biased.

A program has developed to read in the corner features of the training characters, have these feature values fed into the decision trees for each corner and finally output character codes thus formed. The character code is then used as an key to search for the corresponding Chinese internal code. If this character code corresponds to a group of internal codes, all these internal codes will be displayed.

Several test parameters related to the performance of the tree classifier is measured (refer section 3.6.1). The result can be tabulated as follows

(A) General Statistic

- Total number of distinct codes = 8759
- Mean number of characters per code = 1.49
- Standard deviation for the number of characters per code = 1.10
- Coefficient of variation = 1.36
- Original entropy = 13.68
- Final entropy = 0.83
- Reduction in entropy = -12.85

(B) Testing of efficiency and accuracy

As will be described in section 5.1, it is not necessary that all the data from the training set should be resubstituted to the classifier for testing. Random sample of size 500 is chosen for testing and the results are as given.

. Average processing time per character = 0.54 second

. Accuracy = 100%

The recognition speed and accuracy of this classifier is quite attractive. Very high accuracy is guaranteed just because we use the training character set for testing. However, the use of the training character set for testing may not too appropriate in real life situation. No input device can read in the ideal bit pattern of the character that matches perfectly with the training character without any discrepancy or noise.

In section 3.6.3, the performance of the classifier will also be tested with the bit pattern of the noisy character instead of the training bit pattern.

§3.6.3 Noise Model

In real life, there is no perfectly printed Chinese character that can be obtained through the input device such as OCR, scanner etc. So, it is not quite realistic to use the original ideal sample characters to test the performance of the classifier (resubsitution method as described in section 3.6.2).

To simplify the analysis, the noisy Chinese character bit pattern is not directly generated from the OCR or the scanner. It is generated randomly from the ideal character bit pattern using a noise model proposed in [16].

Assume the ideal Chinese character bit pattern is represented by a 0-1 matrix. The noise model proposed in [16] can be briefly described as follows.

Let A be the 0-1 matrix (24x24 bit pattern in our case) of the ideal Chinese character.

B be another matrix in such a way that

B(i, j) = random number uniformly distributed between 0 and 1 for each (i,j).

N be the noise model matrix such that

$$N(i, j) = \begin{cases} 1 & \text{if condition is satisfied} \\ 0 & \text{otherwise} \end{cases}$$

Noise is added to different points (i,j) of A in a way that

$$\begin{cases} A(i, j) \text{ is changed either from 0 to 1 or from 1 to 0} & \text{if } N(i, j) = 1 \\ \text{No change} & \text{if } N(i, j) = 0 \end{cases}$$

Two noise models are described in [16], they are

I. WHITE NOISE MODEL

In white noise model, the noise matrix is represented by

$$N(i, j) = \begin{cases} 1 & \text{if } B(i, j) \leq \alpha \\ 0 & \text{otherwise} \end{cases}$$

where α is some preassigned threshold value which determine the noise level. The noise points generated by this noise model are independent from each other. Character of this type of noise can easily be removed by simple prepossessing.

II. PRINTED MATERIAL NOISE MODEL

As mentioned in [16] the white noise model do not totally reflect the noise characteristic of printed character. The noise generated in printed material is

not independent to the neighboring point. They cannot be easily removed at the prepossessing stage. The noise matrix in this noise model is represented by

$$N(i, j) = \begin{cases} 1 & \text{if } n \leq \alpha \\ 0 & \text{otherwise} \end{cases}$$

where $n = \beta B(i, j) + B(i-1, j) + B(i+1, j) + B(i, j-1) + B(i-1, j-1) + B(i+1, j-1) + B(i, j+1) + B(i-1, j+1) + B(i+1, j+1)$

β = Threshold constant

The degree of noise affecting the character bit pattern depends on the values of α and β in the formulas above.

The white noise model is adopted and testing samples with extra noise incorporated have been generated. By the reason to be given in section 4.1, only 500 characters with appropriate noise added are randomly selected and used for testing. The results for different values of threshold is tabulated below.

Noise Level	Accuracy with noisy sample
0.005	0.474
0.01	0.396
0.02	0.282
0.03	0.230
0.04	0.132
0.05	0.070

Table 3.1

CHAPTER 4 -- POSSIBLE IMPROVEMENT

§4 POSSIBLE IMPROVEMENT

The idea of supervised learning with the help of 3 corner codes has been discussed in chapter 3 and a multistage decision tree classifier has also build. The result shows that a high accuracy (≈ 100%) can be obtained if the ideal sample is used for testing. However, the accuracy of the classifier is rather sensitive to noise (refer section 3.6.2 for the performance of the classifier). To improve the accuracy of the classifier, several possible ways of improvement are suggested by my project partner Mr. K W Sin, the details are depicted in the appendix A.4. In order to maintain completeness for the presentation, a summary of the possible improvement is described in the following paragraphs.

a. Reduce the training and testing samples

   5000 commonly used Chinese character is used instead of the 13,000 training samples. The reduction of the number of Chinese characters with similar shape would reduce the training time required and at the same time increase the discriminating power of the classifier.

b. Filter the noise of the input characters

   Two noise filtering algorithms has been recommended. These algorithms can eliminate the single bit noise and double bit noise occurs at the character bit pattern. However, the noise bit adhere to the character key stroke cannot be

eliminated as there is no knowledge to clarify whether it is noise bit or the actual bit pattern of the character.

c. Decision with overlapping

During the construction of the decision tree, the training characters will be classified at each node of the tree. Any error occurs during the classification may resulting in a wrong character being recognized. As the depth of the tree grows, such misclassification error will be propagated and magnified.

Such misclassification error can be eliminated by duplicating those uncertain items in the decision. At a result, they will be classified to both of the sub-nodes of the parent node. Overlapping will delay the committing of errors to subsequent stages of classification but increases the number of data items residing in the subsequent nodes in the tree. If the depth of the tree classifier is large, such technique is not feasible and a relatively large number of data items will be associated with all the terminal nodes.

d. Backtracking for holes.

"Holes" here refer to those terminal nodes of the decision tree without any training characters assigned. If an ideal character is inputted to the classifier, the corresponding corner code will be obtained. However, if there is some error occurs during the recognition, a

corner code without any character i.e. holes will result and no character can be recognized.

As errors has been occurred, we may wish to fix the errors so that codes can be corrected. The search for such mistake can easily be achieved through back tracking. Assuming that errors committed at greater depth are more likely than errors committed at the top levels, we can devise a back tracking algorithm which will back track on the three trees successively for an increasing number of levels until a realistic code is obtained. Refer to appendix A.4 if you want to understand the whole details of back tracking algorithm.

e. Fuzzy decision function with tolerance limit

To tackle the problem of unnoticeable errors which may have committed during the process of classification, fuzzy decision with the tolerance limited is being studied. As we have pointed out previously, committing error is inevitable. So as to reduce the chance of obtaining unnoticeable errors, it will be better if we do not provide such a definite decision for the internal nodes. If the distance between an object and the hyperplane is smaller than a preassigned threshold value which we will call the tolerance limit, then decision of classification to which region is not made. Rather the decision will be delayed.

f. Different tree architecture

Another way to improve the performance of the classifier is to explore other type of tree architecture. One of the possible alternatives is to combine the three trees to a single one. Suppose the tree for the first corner has been constructed. We can further develop at the terminal nodes of the first tree for a few levels of depth by the consideration of the second corner feature. Similarly the final tree will also be further extended at the terminal nodes by the consideration of the third corner feature. The 1-tree protocol is simple and straightforward although there is no guarantee that it is a good one.

g. Building decision tree by entropy reduction method

It makes use of the 1-tree protocol as describe in item f. Instead of just allowing the use of one particular feature successively at some levels of the tree classifier, the best corner feature is used at every internal nodes of the tree. The best feature here means the corner feature among the three corner features that can gives the most drop in the system entropy or in other words the most negative value of ΔE.

CHAPTER 5 -- EXPERIMENTAL RESULTS

§5 EXPERIMENTAL RESULTS & THE IMPROVED MULTISTAGE CLASSIFIER

§5.1 EXPERIMENTAL RESULTS

Experiments have been carried out to test the performance of the classifier of different protocols as described in chapter 4. Parameters of those described in section 3.6.1 are measured and the results are given as follows.

I) 3-trees protocol

- trained with 13,000 Chinese characters
- without overlapping

This original protocol as developed in section 3 is tested with filtered noisy samples. Two different case are considered, one with back tracking while the other does not.

(a) Without back tracking

Table 5.1 shows the accuracy of the classifier without back tracking under different noise level threshold α.

Noise level	Noisy Sample	with filtering by algorithm 1	with filtering by algorithm 2
0.05	0.436	0.506	0.262
0.1	0.354	0.424	0.220
0.2	0.248	0.332	0.190
0.3	0.184	0.310	0.142
0.4	0.094	0.174	0.108
0.5	0.050	0.140	0.102

Table 5.1

(b) With back tracking

Table 5.2 shows the accuracy of the classifier with back tracking under different noise level threshold α.

Noise level	Noisy Sample	with filtering by algorithm 1	with filtering by algorithm 2
0.05	0.474	0.534	0.296
0.1	0.396	0.464	0.264
0.2	0.282	0.360	0.220
0.3	0.230	0.346	0.178
0.4	0.132	0.222	0.138
0.5	0.070	0.170	0.120

Table 5.2

II) 3-trees protocol

- trained with 5400 commonly used Chinese characters
- without overlapping

A. General Statistic

- Total number of distinct codes = 2507
- Mean number of characters per code = 2.15
- Standard deviation for the number of characters per code = 1.60
- Coefficient of variation = 1.35
- Original entropy = 12.40
- Final entropy = 1.42
- Reduction in entropy = -10.98

B. Testing of efficiency and accuracy

1. with ideal sample -

(a) without back tracking

. Average processing time per character = 0.052 sec

. Accuracy = 0.998

(b) with back tracking

. Average processing time per character = 0.052 sec

. Accuracy = 0.998

2. with noisy samples -

(a) without back tracking

Table 5.3 shows the accuracy of the classifier without back tracking under different noise level threshold α.

Noise level	Noisy Sample	with filtering by algorithm 1	with filtering by algorithm 2
0.5	0.482	0.544	0.296
0.1	0.418	0.476	0.258
0.2	0.284	0.390	0.206
0.3	0.222	0.336	0.184
0.4	0.142	0.252	0.168
0.5	0.082	0.182	0.118

Table 5.3

(b) with back tracking

Table 5.4 shows the accuracy of the classifier with back tracking under different noise level threshold α.

Noise level	Noisy Sample	with filtering by algorithm 1	with filtering by algorithm 2
0.05	0.488	0.550	0.302
0.1	0.424	0.480	0.260
0.2	0.288	0.392	0.210
0.3	0.224	0.336	0.186
0.4	0.148	0.258	0.170
0.5	0.084	0.186	0.120

Table 5.4

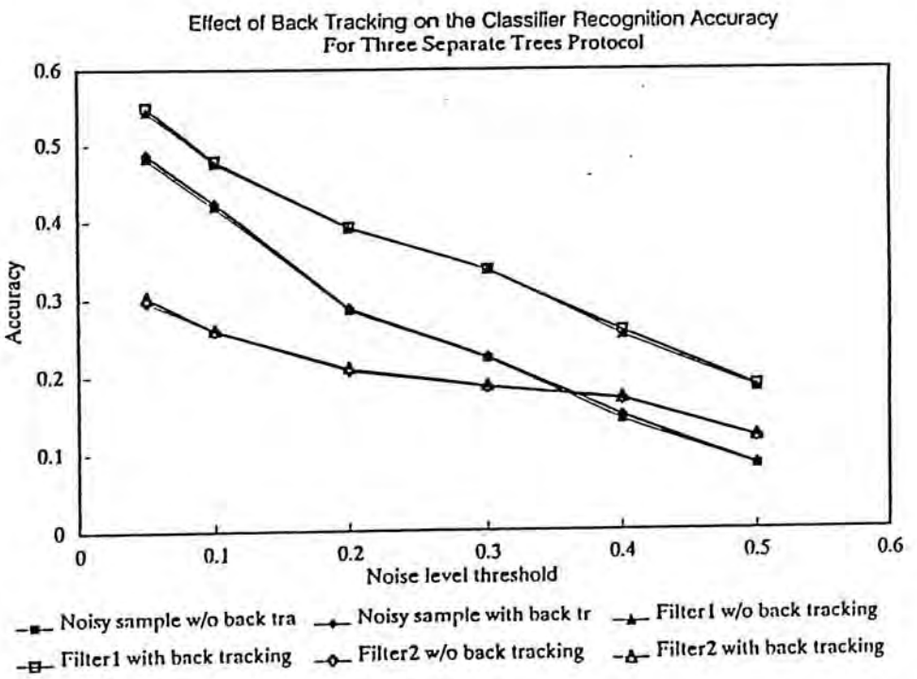


Figure 5.1

Comparing the results of (a) and (b) (refer to figure 5.1, it is noticed that the introduction of backtracking can slightly improved the performance of the classifier. It is also noticed that the filtering algorithm one has continuously performed much better than algorithm two and is useful in

improving the accuracy of the classifier. It may due to the fact that some bits belongs to the character key strokdes are being eliminated by this filter algorithm. Later results will also confirm with such finding.

3. with the introduction of tolerance limit ε and tested with noisy sample
   - no back tracking is introduced
   - Different values of ε are used and tested. The performances of the classifiers are as follows.

     a. ε = 0.001

        ideal sample : accuracy = 0.992

        the accuracy of the classifier is shown in table 5.5

Noise level	Noisy Sample	with filtering by algorithm 1	with filtering by algorithm 2
0.05	0.480	0.542	0.296
0.1	0.420	0.478	0.258
0.2	0.284	0.390	0.208
0.3	0.224	0.336	0.184
0.4	0.140	0.250	0.168
0.5	0.082	0.180	0.118

Table 5.5

     b. ε = 0.005

        ideal sample : accuracy = 0.964

        Table 5.6 shown the accuracy of the classifier under different noisy level

Noise level	Noisy Sample	with filtering by algorithm 1	with filtering by algorithm 2
0.05	0.464	0.522	0.288
0.1	0.410	0.470	0.252
0.2	0.278	0.380	0.202
0.3	0.218	0.328	0.178
0.4	0.136	0.248	0.162
0.5	0.084	0.180	0.118

Table 5.6

c. $\varepsilon = 0.01$

ideal sample : accuracy = 0.930

Table 5.7 shown the accuracy of the classifier

Noise level	Noisy Sample	with filtering by algorithm 1	with filtering by algorithm 2
0.05	0.446	0.502	0.274
0.1	0.394	0.446	0.238
0.2	0.268	0.368	0.196
0.3	0.214	0.318	0.170
0.4	0.130	0.244	0.154
0.5	0.082	0.178	0.118

Table 5.7

4. $\varepsilon = 0.05$

ideal sample : accuracy = 0.654

Table 5.8 shown the accuracy of the classifier

Noise level	Noisy Sample	with filtering by algorithm 1	with filtering by algorithm 2
0.05	0.324	0.346	0.226
0.1	0.266	0.310	0.186
0.2	0.204	0.272	0.162
0.3	0.162	0.246	0.146
0.4	0.094	0.206	0.148
0.5	0.066	0.158	0.102

Table 5.8

In figure 5.2, the accuracy is plotted against the noise level for different values of ε. It is noticed that the smaller the value of ε, the higher the value of the accuracy. Since when ε = 0, the case will just degenerate to the case where no tolerance limit is added and is just the original protocol. As a result, we can conclude that the introduction of tolerance limit does not improve the accuracy of the classifier.

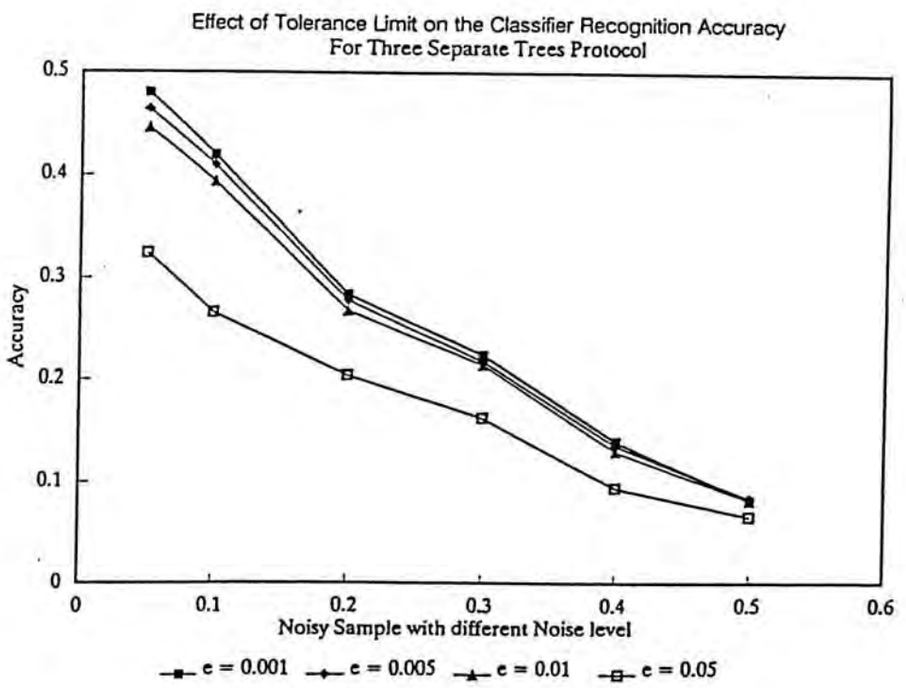


Figure 5.2

III) 3-trees protocol

. trained with 5,400 commonly used Chinese characters

. with overlapping

A. General Statistic

- Total number of distinct codes = 4,096
- Mean number of characters per code = 13.63
- Standard deviation for the number of characters per code = 7.22
- Coefficient of variation = 1.89
- Original entropy = 15.79
- Final entropy = 3.96
- Reduction in entropy = -11.81

B. Testing of efficiency and accuracy

1. with ideal sample

(a) without back tracking:

. average processing time per character = 0.404 sec

. accuracy = 100%

(b) with back tracking

. average processing time per character = 0.394 sec

. accuracy = 100%

As mentioned in section 4.3 that due to the limitation of memory area, the sorting and searching during the character recognition is

done through file in disk. As a result the recognition speed in overlapping case is much larger than that of no overlapping case. In fact, the recognition speed should for these two cases should be closed as the number of decision nodes is more or less the same.

2. with noisy samples

(a) without back tracking

Table 5.9 showing the accuracy of the classifier

Noise level	Noisy Sample	with filtering by algorithm 1	with filtering by algorithm 2
0.05	0.592	0.646	0.378
0.1	0.542	0.602	0.362
0.2	0.434	0.536	0.316
0.3	0.382	0.504	0.266
0.4	0.260	0.394	0.234
0.5	0.194	0.314	0.210

Table 5.9

(b) with back tracking

Table 5.10 showing the accuracy of the classifier

Noise level	Noisy Sample	with filtering by algorithm 1	with filtering by algorithm 2
0.05	0.592	0.646	0.378
0.1	0.542	0.602	0.362
0.2	0.434	0.536	0.316
0.3	0.382	0.504	0.266
0.4	0.260	0.394	0.234
0.5	0.194	0.314	0.210

Table 5.10

Again, the results show that filtering algorithm one is helpful in improving the performance of the classifier but it is not much contribution in back tracking case since back tracking is not necessary in overlapping case.

Comparisons on the accuracy of the classifier in (II) and (III) with the introduction of back tracking and filtering algorithm one are made. The accuracy has been plotted against different values of noise level as shown in figure 5.3. From the graph, it is noticed that the introduction of overlapping has added extra merits to the performance of the classifier.

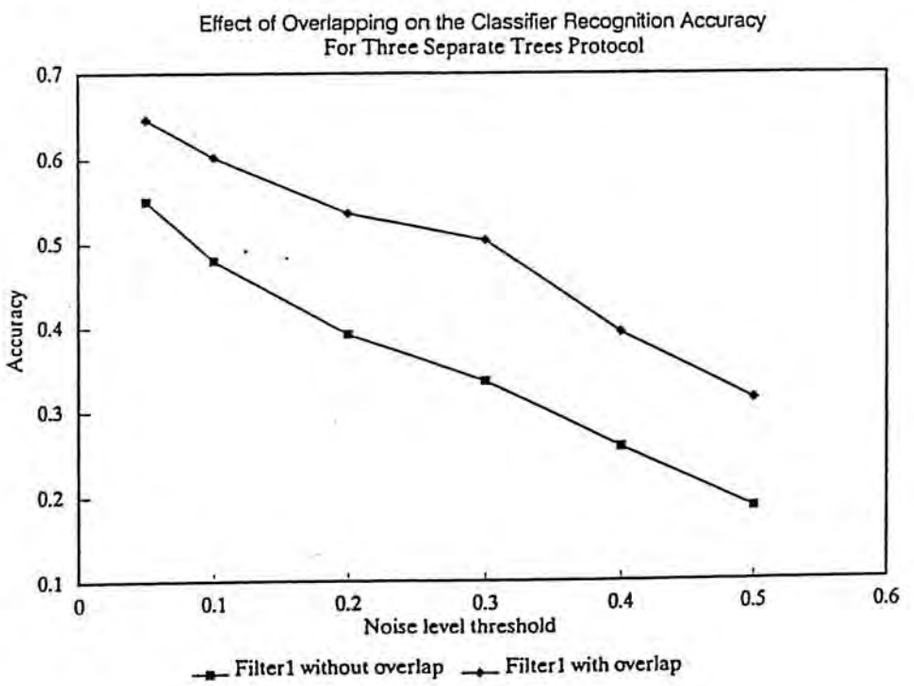


Figure 5.3

IV) 1-tree protocol

. trained with 500 randomly chosen commonly use Chinese characters

. use entropy reduction to select corner features at every node

. with overlapping

. no back tracking for holes is necessary since there is merely no holes with one single tree classifier

A. General Statistic

- Total number of distinct codes = 64
- Mean number of characters per code = 42.83
- Standard deviation for the number of characters per code = 15.33
- Coefficient of variation = 2.79
- Original entropy = 11.42
- Final entropy = 5.24
- Reduction in entropy = -6.18

B. Testing of efficiency and accuracy

1. with ideal sample

. average processing time per character = 0.038 sec

. accuracy = 0.998

2. with noisy sample

Table 5.11 showing the accuracy of the classifier

Noise level	Noisy Sample	with filtering by algorithm 1	with filtering by algorithm 2
0.05	0.788	0.812	0.634
0.1	0.758	0.776	0.612
0.2	0.664	0.720	0.562
0.3	0.652	0.724	0.540
0.4	0.550	0.618	0.488
0.5	0.528	0.612	0.500

Table 5.11

§5.2 CONCLUSION

Based on the above experimental results, the following points can be concluded.

1. Back tracking for holes can improve the performance of the classifier to a certain extent. At the same time, we have also noticed that the increase in the average processing time for each character is only negligible.
2. The introduction of the tolerance limit does not help in improving the performance of the classifier. It has been noticed that when ε is set to zero, this protocol will be degenerated to the ordinary protocol where there is no fuzzy decision by tolerance limit.
3. The use of overlap can improve the performance of the three separate trees protocol. Perhaps the extra costs to pay for such an improvement will be the increased average processing time per character (from approximately 0.05 second to 0.4 second). Besides,

the mean number of characters per distinct code has also increased from 2.15 to 13.63. Of course, if this classifier serves only as the first stage of our discrimination process, such an increase in the mean number of characters will not bother us much. Rather the increase in the accuracy should be our major concern. As a result, despite all these drawbacks, the use of overlapping should still be adopted.

4. The one single tree protocol with overlap has found to be quite promising in its discriminating power as compared with other protocols. However, since only 500 characters are used in training up the classifier, the good performance of such classifier will not be guaranteed for a larger training set (say with 5400 commonly used Chinese characters). Experiments should be performed to confirm the above finding.
5. Concerning the filtering algorithms, we have noticed that the filtering algorithm one has continuously better performance than the algorithm two. As a result, we can conclude that inputted characters should be filtered by algorithm one before they are fed into the recognition system.

CHAPTER 6 -- IMPROVED MULTI-STAGE TREE CLASSIFIER

§6 IMPROVED MULTISTAGE TREE CLASSIFIER

§6.1 THE OPTIMAL MULTISTAGE TREE CLASSIFIER

The results discussed at chapter 5 reveal that the three separate tree protocol should be adopted. The idea of backtracking for holes should also be incorporated. As a result, the improved multistage classifier recommended is:

- trained with 5,400 commonly used Chinese characters
- idea of the backtracking is incorporated
- with overlapping

The classifier will follow the following sequence of steps in classifying the characters inputted.

1. The codes for all the 5,400 characters are inputted through a file. These codes will be sorted in ascending order with the use of radix sort. The sorted codes, together with the corresponding Big 5 codes, will be stored in an array for later reference.
2. The bit pattern for the characters are inputted to the system for classification. They will be inputted to the system one by one. For every character inputted, the corner features at the three corners will be extracted first.
3. The discriminant functions at every nodes of the tree classifier have already been stored in the program. With these functions, the code for the character will be

calculated. Of course, if it is a character from the ideal training sample, the code computed should exactly match with actual code for that character. There may be deviation if this character is from some noisy sample.

4. The code will be used as the key to search for the corresponding Big 5 code for that character inputted. Binary search will be used.
5. The inputted character will be recognized as that character having the Big 5 code as that found in (4).
6. The process will be repeated for another character inputted.

§6.2 PERFORMANCE ANALYSIS

Classifier based on the improved multistage tree idea has been constructed and being tested with ideal samples, noisy samples and samples which have been filtered using filtering algorithm one. Again, only random samples of size 500 are generated for the purpose of testing. The results are given as follows.

(A) The Recognition Rate

Four categories can be identified in terms of the recognition result. They are the percentage of characters

1. correctly recognized as a single unique character;
2. correctly recognized as a group of characters;
3. incorrectly recognized as another character or

another group of characters;

4. having not been recognized as any characters..

Both (1) and (2) belongs to the class of success while cases (3) and (4) lead to errors. In particular, case (3) will incur unnoticeable error and is most undesirable. The results of the classification is given in table 6.1.

%	ideal sample	Noisy sample at noise level 0.005	Filtered sample at noise level 0.005
1	0%	0%	0%
2	79.6%	59,2%	64.6%
3	0.2%	40.8%	35.4%
4	0%	0%	0%

Table 6.1

(B) Recognition speed

Since idea of overlapping has been incorporated and the total number of codes is large, we have to use file to hold the sorted during run time. This has significantly increased the average processing speed of the classifier. However, the speed is still small enough to be implemented in the microcomputer environment. Upon testing, the average processing time per character is found to be.

	Ideal Sample	Nosiy sample at noise level 0.005	Filtered sample at noise level 0.005
Processing Speed (sec)	0.5	0.5	0.5

Table 6.2

Besides, the performance of other protocol has also been studied for referece and the recognition rates can be tabulated below:

I. 3-trees protocol

- trained with 5400 commonly used characters
- without overlapping
- with back tracking

%	ideal sample	Noisy sample at noise level 0.005	Filtered sample at noise level 0.005
1	20.2%	8.8%	10.8%
2	79.6%	40%	44.2%
3	0.2%	51%	45%
4	0%	0.2%	0%

Table 6.3

II. 1-tree protocol

- with overlapping

%	ideal sample	Noisy sample at noise level 0.005	Filtered sample at noise level 0.005
1	0%	0%	0%
2	99.8%	78.8%	81.2%
3	0.2%	21.2%	18.8%
4	0%	0%	0%

Table 6.4

CHAPTER 7 -- FURTHER DISCRIMINATION BY CONTEXT CONSIDERATION

§7 FURTHER DISCRIMINATION BY CONTEXT CONSIDERATION

§7.1 IDEA

The multistage tree classifier developed in the previous chapters only provide a partial discrimination of all the Chinese characters into a number of clusters. Since global feature (the corner feature) is used, we cannot expect discrimination to be complete just with the tree classifier developed. Although we can increase the depth of the trees so that data in each clusters can be further discriminated into smaller clusters. However, this increase in depth is not justified in light of the additional error encountered for a tree with greater depth as pointed out in [15]. As a result, further exploration should be sought so as to completely discriminate the characters.

It has been suggested that since only global features have been used in the first stage of our decision, local feature may help in further classification in our second stage of decision. Local features including the search for a dot around certain position, search for a stroke at the corner, a slanted stroke at the right edge and the like, have been recommended. In this way, rules can be set up for this second stage discrimination. However, there are some drawbacks to this approach. Since we have left with quite a large number of clusters from our first stage of work, may be up to hundreds or thousands of rules are needed so as to

further discriminate these clusters of characters. This is time consuming and laborious. Moreover, there is no general strategy for the setting up of such rules. We cannot decide whether one setting is better while compared with another setting.

To tackle the entire problem, we decided to use the semantic of the language. It has been noticed that in every application of Chinese character recognition, the input to the system should be in the form of a passage. This means that characters will be grouped in phrases and be fed into the recognition system successively. Suppose a certain character has been recognized. Then we can guess the next character to come and have them be confined to only tens of characters which are semantically related to the previous character. For instance, " 段 ", " 切 ", " 次"may follow the character " 一 ". Hopefully, based on such consideration of context, we can discriminate every character uniquely when the whole passage is fed into the system.

There is a difficulty in grouping all the semantically related words in Chinese since such grouping done manually is time consuming and laborious. Fortunately, help can be sought from commercially available Chinese system. The Eten Chinese system is one which provides an input method by the consideration of context. The lookup table for this input method actually groups together related words of a character so as to facilitate subsequent searching more easily.

Converting this file to a file with coding in Big 5, we can store the lookup table in a particular file, the format of which is

<Big 5 of 1st character> <list of semantically related words>
<Big 5 of 2nd character> <list of semantically related words>
<Big 5 of 3rd character> <list of semantically related words>

.....

The first character on each line will be served as the key during subsequent searching. A partial listing of this table has been included in the appendix for reference.

§7.2 DESCRIPTION OF ALGORITHM

The implementation of the idea in computer is simple. All we have to do is to store the previously recognized character and look for the corresponding list of semantically related characters which are to be stored in an array. The bitmap of the next character will then be inputted to the system. With the use of the multistage tree classifier developed previously, the first stage discrimination is performed and a group of characters is obtained. To choose the most likely character among this particular group of words, the second stage discrimination process will be performed by context consideration. The list of semantically related characters corresponding to the previously recognized character is sought and be compared with each of the character in the

group of characters left after the first stage of classification. If a match can be found, then the character has been uniquely recognized. If a match cannot be obtained, decision cannot be made at this moment. Further decision has to be made and there are various methods in tackling the situation which are to be discussed later in this section.

To summarize the above discussion, the following algorithm has been suggested.

1. Store the previously recognized character to variable pre_char.
2. Get the list of semantically related characters of pre_char.
3. Read in bitmap of next character.
4. Multistage tree classifier is used and a group of characters is concluded. These possible characters are stored in array pos_char[] and the total number of characters is stored in variable max_pos_char
5. For i=0 to max_pos_char-1
   5.1 Check whether pos_char[i] is in the list of the semantically related characters of pre_char.
   5.2 If found,
       k ⟵ i
       Report the finding
       pre_char ⟵ pos_char[k]
   5.3 Otherwise, Uncertainty Tracking algorithms
6. Go to step (1).

UNCERTAINTY TRACKING ALGORITHM

Our final task is to find methods to deal with the case of uncertainty. Obviously there exists a number of algorithms to tackle the problem.

The first method tries to defer the decision in later stages of classification. When any uncertainty is encountered we simply do not decide and try on every possibilities until any one such path, upon completion of subsequent stages of classification, gives a definite decision to the classification. In other words, we wait until one such path gets a match. This approach, although sounds sensible, is not practical for implementation. Obviously the number of possible paths will be booming due to the large number of semantically related characters at each level of classification. Such depth first search of solution will be very time consuming. On the other hand, there is no guarantee that a feasible solution should be obtained even after prolonged search.

Breadth first search may be another solution but still the exponentially growing number of possible paths also hinders the implementation of the algorithm. Searching does not seem to be a good method. May be the introduction of a suitable heuristic function will render selective search to be a better way. But how can we choose the heuristic function? There is simply no easy method.

The next method is simply to ignore that character. Whenever uncertainty arises, we simply admit that our system fails to recognize correctly this character and leaves the answer untouched. The system will then proceed to recognize the next character coming into the system. However, such method is definitely not good since our recognition system depends very much on the previously recognized character if context is to be used as a classification criterion. Any characters having left to be unrecognizable will make the entire system to fail in subsequent stages of classification for other characters.

Then we see that the most straightforward way to tackle uncertainty should be to let the user to make the decision for us. In such case, when uncertainty arises, all possibilities will be displayed to the user and the user will make the decision for the system before the system goes on. This method, though a little subtle, should be the simpliest and the best that we can have.

§7.3 PERFORMANCE ANALYSIS

Adopting the improved multistage tree classifier as developed in chapter 6 as the first stage of discrimination and the idea of context consideration as the second stage, a final multistage tree classifier can be constructed.

We cannot test the performance of this classifier by inputs which are not realistic and cannot suit the requirement for context consideration. Characters used in the testing samples of our previous work are merely not semantically related in any way. As a result, we have to choose a passage to serve as the testing sample. Again ideal sample, noisy sample and filtered noisy sample are fed into the system and tested successively.

Similar to the four categories for the recognition result discussed in section 6.2, there will also be categories of outcome in our case.

1. Characters to be correctly recognized as a single unique character.
2. The first stage of classification leaves a group of characters and the second stage of classification cannot successfully select the correct character from this group. In this case, the result will still correspond to a group characters.
3. Character has been incorrectly recognized either as another character or as residing in another group of characters.
4. Characters cannot be recognized as any known character even at the first stage of classification. Again, we have also assumed that the character has been uniquely recognized so as to facilitate further classification

Adopting the four categories, the performance of this classifier can be demonstrated as follows.

Percentage being classified to	Ideal Sample
category 1	38.44%
category 2	58.75%
category 3	2.81%
category 4	0%

Table 7.1

Despite of the lower accuracy of the 3-tree protocol without overlapping, it has been found that the percentage of characters which can be uniquely recognized, that is. those being classified to category 1, is to some extent greater than that with overlapping. The following table summarizes the performance of such classifier.

Percentage being classified to	Ideal Sample
category 1	51.56%
category 2	46.88%
category 3	1.56%
category 4	0%

Table 7.2

CHAPTER 8 -- CONCLUSION

§8 CONCLUSION

§8.1 Advantages of the classifier

The advantages of the classifier thus developed can be summarized as follows.

1. It is fast and efficient when passage of moderate size is inputted for recognition.
2. If ideal sample is used, the accuracy of the classifier is very high and is approaching 100%.
3. The use of three separate decision trees has make possible the use of distributed processing where one processor will be responsible for one decision tree.
4. Practically we have included all commonly used Chinese characters in our system and there is little chance of getting a character input which is unknown to the system.
5. If a new character is added to the system, we can define a new code for this character based on the existing classifier by allowing this character to go through our system and have this result be written on our code file. However since our code file has already been sorted, it will take some time for this new code to be inserted at a proper position in the file.
6. Since we have been making use of Big 5 as the internal coding system, the chaaracters being recognized will all be coded in Big 5. As a result, the output from our recognition system can easily be interfaced with all Chinese systems commonly used in Hong Kong,

§8.2 Limitations

There are some limitations to our classifier and they are listed below.

1. The performance of the classifier degenerates with noisy sample input at larger noise levels. This is not desirable since noisy sample input is more realistic than the ideal one.
2. The classifier is font sensitive. The performance will degenerate when tested with characters of other font type, e.g. Ming font.
3. This is not fast enough when long passages are inputted to the system for recognition. This is a problem originated from the large number of codes generated as a result of overlapping.
4. There is still a small probability of unnoticeable error which is highly undesirable since these errors can hardly be detected and located.

CHAPTER 9 -- AREA OF FUTURE RESEARCH AND IMPROVEMENT

§9. AREA OF FUTURE RESEARCH AND IMPROVEMENT

§9.1 DETAILED ANALYSIS AT EACH TERMINAL NODE

In the tree classifier recommended in chapter 6, the number of levels in the decision tree is quite large in order to make the number of Chinese characters at each terminal node reasonably small. To fulfil this requirement, a lot of mis-classification occurs.

The mis-classification error can be reduced by several techniques which includes overlapping the critical characters (refer section 4.3), back tracking for holes (refer section 4.4) and use of fuzzy decision function with tolerance limit (refer section 4.5). However, all the technique mentioned cannot make a great improvement due to the nature of corner feature which can only discriminate the Chinese characters with different corner styles. It is quite obvious that the recommended tree classifier can only be used as the first stage classification which roughly classifies the character set into groups with similiar corner styles. In this way, the number of levels in the decsion tree can be reduced in order to reduce the error incurred.

The actual identification is done at the second stage classification where special discrimination techniques on particular set of character(s) are used. One example of this second stage classifier is the use of context

consideration as that used in Eten Chinese System (refer chapter 7 for details). However, this method is only useful if a large database with context relation is maintained. Furthermore, the Chinese character which has no context relation with the other Chinese character or is the first character of the sentenses may not be recognized.

§9.2 IMPROVING THE NOISE FILTERING TECHNIQUE

It is shown in section 4.2 & 5 that a good noise filtering technique can improve the recognition rate of the tree classifier. In section 4.2, two noise filtering algorithms has been recommended. However, both of the algorithms can only eliminate a single noise bits and two consecutive noise bits. The noise bits that are closed to the character key stoke or closed to other noise bits cannot be filtered.

It is recommended to derive an effective noise filtering technique that can eliminate the unnecessary noise bits especially in the feature extraction regions. This noise filtering algorithm can be included into the feature extraction program at which more analysis at the feature extraction region is done before extracting the corner features.

§9.3 THE USE OF 4 CORNER CODE

In approach 2, three corner codes are used to assist the training of the tree classifier in recognizing the Chinese character. It can shorten the training time and improve the flexibility of the tree classifier especially when the training character set is changed. However, several characteristics of three corner codes also degrades the performance of the classifier i.e.

a. In three corner code, there are 100 classes (i.e. 00 - 99 classes) per each corner. For supervised learning, the training character set is divided into 100 classes according to the three corner code. It is then combined into two groups based on the minimum clasification error in the covariance matrix. However, to minimize the global classification error of an 100 classes decision tree is a very difficult task. A lot of calculation is required which makes the implemenation impossible. As a result, making use of the heuristic proposed in section 3.4, we only got the local minimum at each level of the tree classifier instead of the global minimum of the whole tree.

   Furthermore, the 100 classes requirement would increase the number of levels for each decision tree. The mis-classification error accumulate from level to level and the overlapping requirement will then drastically

increase.

b. Only the features of the three corners are used for the training of the tree classifier which is insufficient to distinguish a large character set.

The problems mentioned greatly increase the error of the tree classifier or reduce the performance of the classifier. However, it can be improved with the use of 4 corner code instead of 3 corner code.

Four corner code is similiar with three corner code as have been mentioned since section 3.2. But, in 4 corner code, only 10 classes (i.e. 0 - 9) for each corner is used instead of 100 classes. The computational requirement in getting the global optimum decision tree is geatly reduced when compare with 3 corner code method.

With 10 classes per each corner, the number of levels, and thus the error accumulated in each level and the number of overlapping are greatly reduced.

Furthermore, with the features at four corners of the character instead of three corners, the power of discrimination will be improved.

However, the following implementation difficulties will be encountered.

a. In the existing Chinese Systems available in the market like Eten Chinese System, KC Chinese System, there is no

four corner code table available. A lot of implementation effort will be required to build this four corner code table as there are over 13,000 Chinese characters.

b. In four corner code Chinese character input method, one four corner code may corresponding to more than one Chinese characters.

c. If four corner code is adopted, four decision trees are required instead of three decision tree. The increase in the number of decision trees would increase the probability of classification error and complicate the back traking effort. However, it can reduce the number of levels in each decision tree.

To conclude, it is worthwhile to implement with four corner code instead of three corner code. A more practical approach is to train up the classifier with reduced training character set and compare the peformance with that of the three corner code.

§9.4 INCREASE IN THE DIMENSION OF THE FEATURE SPACE

As described in section 2.2.6, there are a total of 24 feature points (8 feature points i.e. 4 at the x axis and 4 at the y axis for each corner of the character) used in the training of the tree classifier to discriminate up to 13,000 Chinese characters. However, at each internal node of the corner decision tree, all the 8 feature points of

the corner are used up in determining the leaves of the node. It is not preferrable according to the argument of C.Y. Suen & Q.R.Wang in [15].

According to Suen's argument in [15], those features which have been used in the parent node will not be used in the child nodes again because these features become less informative. It is obvious since similiar classes are assigned to the same child node after clustering using these features, thus they have less discriminative power in this child node than that of the parent node.

Because of this philosophy, it is recommended to increase the number of feature points at each corner, say 16 points instead of 8 points. At each decision node, only the best feature points are used. Different feature points may be used at different decision nodes. To select the best feature points, the feature merit measures like information content measure or Fisher's criterion are suitable for feature selection in a multiclass problem.

§9.5 1-TREE PROTOCOL WITH ENTROPY REDUCTION

The idea of single tree protocol with overlap and entropy reduction has already explored in section 4.7, test with small training character set (500 training samples) have also been done and the result is quite promising (refer chapter 5). Due to the limitation of the memory and

processing power in PC environment, this approach was not further elabolate in our project. However, it is worthwhile to have some further research in this area.

One of the most significant advantage of 1-tree protocol over the 3-trees protocol is the reduction of the overlapping element at the terminal nodes. The details has already mentioned in section 4.3.

§9.6 THE USE OF HUMAN INTELLIGENCE

It has been mentioned in the section 10.1 that two stage tree classifier is better than one stage classifier in high speed, large character size, character recognition process. However, mis-classification still happen in two stage classifier, a lot of effort such as overlapping or back tracking method should be used to reduce the error. To further improve the classifier performance, a most straight forward way is to display all the possible Chinese character when an unlogical Chinese character is detected. The unlogical Chinese character here mean that there is no context relation with the neighbour characters. Human intelligence has to be used to select the right character.

APPENDICES

A.1 K-MEANS

The original K-MEANS algorithm uses the arithmetic mean (i.e. the centroid) of the data as the cluster centres and based on a group of preassigned k initial cluster centres as seeds to classify the entire data set into these k clusters. The usual Euclidean distance is used as the distance measure between individual object and a given object will be classified to any particular cluster if its distance to that cluster centre is the minimum. During the classification process, the cluster centres will be continuously updated and the number of objects in the clusters incremented. The algorithm can be summarized as follows.

ORIGINAL K-MEANS

1. Choose x_1 , x_2 , ..., x_k as the initial k cluster centres.

   Let n_i = number of data in the ith cluster

   Set $n_i = 1 \quad \forall\ i = 1,2,\ldots,k.$

2. Take x from the data set, while not end of file do

   2.1 Calculate $d(x,x_i) \quad \forall\ i=1,2,\ldots,k$

   where $d(x,y)$ = Euclidean distance between x and y

   2.2 Assign x to the jth cluster such that

   $$d(x,x_j) = \min_i d(x,x_i)$$

   2.3 Update cluster centre x_j of the jth cluster by

   $$x_j = \frac{n_j x_j + x}{n_j + 1}$$

2.4 Update the number of data in the jth cluster by

$$n_j = n_j + 1$$

3. Repeat the classification process by feeding the data into the system based on the k cluster centres just calculated in (2).

The above K-MEANS algorithm has been criticized that there seems do not have an easy way of obtaining the initial k cluster centres as the seeds. One possible improvement is then to repeat the above algorithm until the cluster centres become stable. In that case, the algorithm is said to have converged. As a result, the modified K-MEANS algorithm is:

MODIFIED K-MEANS

1. Choose $x_1, x_2, \ldots, x_k$ as initial k cluster centres.
   Let n_i = number of data in the ith cluster
   Set $n_i = 1 \quad \forall\ i=1,2,\ldots,k.$
2. Repeat
   2.1 Save $x_i^0 = x_i$ for $i=1,2,\ldots,k$
   2.2 Take x from the data set, while not end of file do
      2.2.1 Calculate $d(x,x_i) \quad \forall\ i=1,2,\ldots,k$
         where d(x,y) = Euclidean distance between x and y
      2.2.2 Assign x to the jth cluster such that

$$d(x,x_j) = \min_i d(x,x_i)$$

2.2.3 Update cluster centre x_j of the jth cluster by

$$x_j = \frac{n_j x_j + x}{n_j + 1}$$

2.2.4 Update the number of data in the jth cluster by

$$n_j = n_j + 1$$

Until $| x_i^0 - x_i | < \varepsilon$ for all i where ε is some preassigned tolerance value.

The modified K-MEANS algorithm, though more robust to the initial choice of cluster centres, is complicated and takes time for it to converge. Since convergence is not guaranteed and it usually takes quite a long time for us to notice the convergence or divergence of the Algorithm, the above modified algorithm is usually not recommended. Practically the original K-MEANS algorithm is more preferable.

Apart from the criticism for the difficulties in choosing the initial k cluster centres, another major problem associated with the algorithm is the choice for the value of k. What value of k should we choose? This is actually a dilemma for us. One possible solution is the K-MEANS with coarsening and refining parameters.

In this new algorithm, the number of clusters is not fixed and will be changed during the training period. There are two possible changes in the number of clusters, one for cluster splitting and the other for cluster merging. When

the radius of a certain cluster is too large, that cluster will be split into two individual clusters, thus increasing the total number of clusters by one. On the other hand, when the distance between any two clusters is too small, the two clusters will be merged together to form one single cluster, thus decreasing the total number of clusters by one. Such splitting and merging processes can help to stabilize the final groupings of the clustering result in a more preferable way.

How can the "large" and "small" in the process of splitting and merging be characterized? Here we will use a coarsening parameter C and a refining parameter R specified before the start of the training. For simplicity, the splitting rule will only be applied when a certain object is being assigned to one particular cluster. If the distance between that object to that particular cluster is greater than C, then group splitting occurs and that object will individually form a new cluster. For merging of clusters, we will merge any two clusters if the distance between the two cluster centres is less than R. The entire algorithm has already been summarized as the K-MEANS with coarsening and refining as discussed in section 2.3.

A2. APPROACH ONE - UNSUPERVISED LEARNINGS

§2.1 IDEA

In choosing the approach towards the construction of a classifier, we seek for different ways to classify our objects. Of course, training samples are provided. Depending on the nature of the samples given, two different approaches exist, here, we will call unsupervised learning and supervised learning.

By supervised learning, we mean that the actual class membership of the objects are known and we can base on such knowledge of class membership to group the objects into some clusters. In this way, the classifier can then be constructed. More details of this approach will be given in the next section.

By the term unsupervised learning, we actually refer to the case where no prior knowledge of the class membership of the objects in the training sample is known to the system. We can classify the objects into clusters by some intrinsic likeliness of the objects themselves. For instance, in classifying a given basket of fruit, the likeliness may be measured by the external colour of the object so that all objects appeared red come together, so do green objects, blue objects and the like. Of course there is no absolute measurement of likeliness. Colour is one way in the above example while weight may be another. Then how can we decide

whether to use colour or weight as the measurement? This forces us to define clearly the goodness of the measurements, which is actually quite a difficult task. Besides, there is also no absolute measurement in the correctness of the classification since we merely do not know which object belong to which class as class membership is unknown to the system.

The two problems stated above present the intrinsic difficulties in the implementation of unsupervised learning. However, unsupervised learning still has its own distinguished advantages. Since no prior knowledge is required in the entire process of the classification, this provides a suitability for most of our everyday life problems since usually such prior knowledge is not known beforehand. This non-parametric nature renders the process to be highly flexible and thus versatile. As a result, most of the research in pattern recognition are done based on this approach.

In this chapter, we are going to explore how the technique of unsupervised learning can be used in the training of a chinese character recognition system. Like the usual step in most of other recognition systems, features are first extracted from input characters. After going through a clustering algorithm, objects will be clustered into groups which separate them into one another. The final stage will be to construct the tree classifier and form the optimum

decision tree. The most distinguished differences between supervised and unsupervised learning are in the use of the clustering algorithm and the method of constructing the decision tree classifier. All these details will be discussed in detail in the sections following. Here, an overview is given.

1. Several useful feature extraction methods from the input bit pattern of the character will be discussed.
2. Different unsupervised clustering algorithms will be described. We will choose one clustering algorithm to cluster the given training set into different groups based on the use of one particular feature. This step will be repeated for all other features which have been extracted from the characters.
3. A decision table as will be formed, details of which will be discussed.
4. From the decision table just constructed, we will show how the optimum decision tree can be built.

There are some problems associated with this approach in the training up of the classifier since practically the algorithm described in (4) cannot be implemented computationally. This belongs to the class of NP complete problems. Details and reasons will be given in section 2.6.

§2.2 FEATURE EXTRACTION

The idea of the unsupervised learning has already been mentioned in section 2.1. To start with the classifier development, feature selection pay a very important role in determining the performance of the classifier. Feature is

the measurement of likeliness we have mentioned in section 2.1. If a good feature is selected, the unknown object can easily be identified. In this section, the feature selection criteria will first be discussed. Different feature extraction methodologies that are favourable for this approach will also be described.

§2.2.1 Feature Selection Criteria

The most distinguished advantage of the tree classifier recommended in section 2.1 is to break down the most complicated decision into a tree of many simple, easy to obtain decisions.

Based on this philosophy, the feature extracted should satisfy the following feature selection criteria

a. The feature should be as simple as possible so that the time required to extract the feature is reduced to minimum.
b. The feature can effectiveity classify a particular set of Chinese characters.

c. The feature selected should provide maximum separability.
d. The feature should be insensitive to noise, rotation, shift etc.
e. The feature should require as minimum memory as possible.

It is the fact that no single feature can satisfy all of the above selection criteria, which means that measures such as the probability of error, the reduction in entropy etc should be calculated in order to determine which feature is the best choice. The one with the maximum advantage like separability, number of resulting classes etc. will be selected.

Furthermore, it is also mentioned in [2] that the size of the feature to be used at each node should be limited to be much smaller than the total number of available features. The sections follow will describe some of the most common features used in recent research of Chinese character recognition. Of course, it is by no means a enhaustive list but only provides an overview of the methods used in the extraction of useful features from the character.

§2.2.2 4C Code

"4C" actually refers to the four corners of a Chinese character. It has been argued that in the Chinese language, the shape of the corners can provide us much information. As a result the 4C code can be used as a feature for classification.

The 4C code is defined by encoding the four corner square zones of a character according to the size of the black points. To make the feature value more robust to the total number of pixels in the given character, the four black to white ratios are calculated. The respective order of the codes taken is illustrated in figure 2.1.

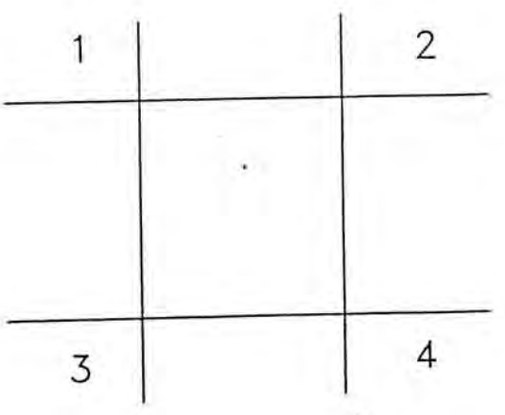


Figure 2.1 The Order of the 4 Corner Code taken from a Chinese Character

For a 24 x 24 bitmap, each corner consists of a 6 x 6 bitmap which will be amounted to 36 dots. We refer black dot as "1" and white dot as "0". Therefore the 4C code will be defined as a vector x of dimension four where

$$x = (x_1 \quad x_2 \quad x_3 \quad x_4)$$

and x_k = black-to-white ratio of the kth corner.

To formalize the calculation, denote

$B = (b_{ij})_{24x24}$ = bitmap of the Chinese character

where $b_{ij} = 0$ or $1 \quad \forall\ i,j$

Then

$$x_1 = \frac{\sum_{i=1}^{6} \sum_{j=1}^{6} b_{ij}}{36}$$

$$x_2 = \frac{\sum_{i=1}^{6} \sum_{j=19}^{24} b_{ij}}{36}$$

$$x_3 = \frac{\sum_{i=19}^{24} \sum_{j=1}^{6} b_{ij}}{36}$$

$$x_4 = \frac{\sum_{i=19}^{24} \sum_{j=19}^{24} b_{ij}}{36}$$

Such ratios calculated are insensitive to noise and is robust with respect to the change in the pixel size of the character.

§2.2.3 Regional Code

The regional code is defined by encoding the four corner square zones and the two central strips of a character according to the size of the black points. Similar to the 4C code discussed previously, the six black to white ratios are calculated so as to make the feature more robust to the total number of pixels in the character.

The first four values of the feature vector are just those of the 4C code. The fifth value corresponds to the vertical central strip while the sixth value corresponds to the horizontal central strip. Just like the 4C code, each corner consists of a 6 x 6 bitmap. The two central strips will be a 5 x 24 bitmap which is just 120 dots.

Specifically, if $x = (x_1 \; x_2 \; x_3 \; x_4 \; x_5 \; x_6)$ and $B = (b_{ij})_{24x24}$ = bitmap of the Chinese character where $b_{ij} = 0$ or $1 \; \forall \; i,j$.

Then

$$x_1 = \frac{\sum_{i=1}^{6} \sum_{j=1}^{6} b_{ij}}{36}$$

$$x_2 = \frac{\sum_{i=1}^{6} \sum_{j=19}^{24} b_{ij}}{36}$$

$$x_3 = \frac{\sum_{i=19}^{24} \sum_{j=1}^{6} b_{ij}}{36}$$

$$x_4 = \frac{\sum_{i=19}^{24} \sum_{j=19}^{24} b_{ij}}{36}$$

$$x_5 = \frac{\sum_{i=1}^{24} \sum_{j=10}^{14} b_{ij}}{120}$$

$$x_6 = \frac{\sum_{j=1}^{24} \sum_{i=10}^{14} b_{ij}}{120}$$

§2.2.4 Walsh Transform

Walsh transform has been claimed to be a simple, fast and reliable method for separating complex Chinese characaters [8]. Two dimensional Walsh transform will be discussed here.

Walsh transform involves only 1 or -1 operation and thus is a fast computation. It distinguishes from the conventional Fourier transform in that the former has "sequency" property and the latter has "periodicity" property. It is suitable for the analysis of the central portion of the characters.

Walsh function [9] $WAL(n,\theta)$, $-\frac{1}{2} < \theta \leq \frac{1}{2}$, are defined recursively as follows:

$$WAL(0,\theta) = \begin{cases} 1 & \text{if } -\frac{1}{2} < \theta \leq \frac{1}{2} \\ 0 & \text{otherwise} \end{cases} \qquad \text{and}$$

$$WAL(2j+q,\theta) = [-1]^{\lfloor \frac{j}{2} \rfloor + q} \left(WAL(j,2(\theta + \tfrac{1}{4})) + [-1]^{j+9} WAL(j,2(\theta - \tfrac{1}{4})) \right), \quad q = 0 \text{ or } 1;\ j=0,1,..,\ n.$$

If θ is not in $(-\frac{1}{2}, \frac{1}{2})$, then $WAL(j,\theta) = 0$. The ordering of Walsh functions is sequency (Walsh) ordering: $WAL(0,\theta)$, $WAL(1,\theta)$, $WAL(2,\theta)$, ...

As usual, suppose the input character is represented by an

M x M bitmap $(y_{ij})_{MxM}$ i = 1,2,..., M; j = 1,2,.., M and let the origin of the coordinate system be translated to the centre of the character. The central $\frac{M}{2} \times \frac{M}{2}$ submatrix centred at the origin is extracted .

Let $N = \frac{M}{2}$.

1. When N is odd, say N = 2k+1, we extract N equally spaced points from interval $[-\frac{1}{2}, \frac{1}{2}]$ and they are

$$-\frac{1}{2}, -\frac{k-1}{2k}, -\frac{k-2}{2k}, \ldots, -\frac{1}{2k}, 0, \frac{1}{2k}, \ldots, \frac{k-1}{2k}, \frac{1}{2}$$

2. When N is even, say N = 2k, then the N equally spaced points are

$$-\frac{2k-1}{4k}, -\frac{2k-3}{4k}, \ldots, -\frac{3}{4k}, -\frac{1}{4k}, \frac{1}{4k}, \frac{3}{4k}, \ldots, \frac{2k-1}{4k}$$

Substituting these N points into θ of WAL(n,θ) given above, we get sequence of values of WAL(n,i), i=0,1,2,..., N-1.

Now rename the extracted central portion as $(x_{ij})_{NxN}$, i=0,1,2,...,N-1; j=0,1,2,...,N-1. Then the two dimensional Walsh transform of (x_{ij}) is

$$C_{mn} = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} x_{ij} \, WAL(m,i) \, WAL(n,j)$$

where C_{mn} is called the Walsh coefficient.

We have noticed that since x_{ij} and Walsh functions have values 1 or -1 so that the above calculation can be

performed in "and or" circuit which is very fast. resides it has been suggested that in Chinese recognition system, only those C_{mn}'s with $2 \leq m \leq 6$ and $2 \leq n \leq 6$ are necessary for consideration of feature selection.

§2.2.5 Black Dot Density Projection Profile

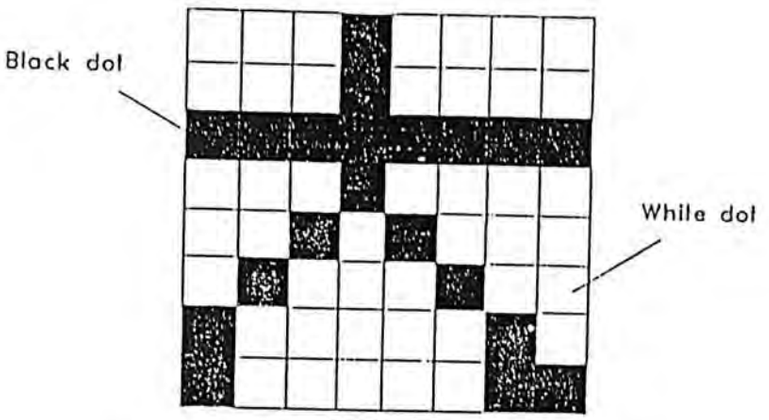
The bit pattern of a printed Chinese character can be represented by black dots and white dots matrix as shown in figure 2.2. The black dot density projection profile over the x and y axis shows the complexity of the character and can be used as an feature for character recognition.

To determine the black dot density projection profile, the total number of black dots projected to x axis and y axis are first calculated (figure 2.3). The mean, 1st, 2nd and 3rd moment of the black dot density projection profile are also calculated. The calculation is formulated as follows

If P(i,j) is the bit map of the Chinese character such that

$$P(i,j) = \begin{cases} 1 & \text{for black dot} \\ 0 & \text{for white dot} \end{cases}$$

where $1 \leq i \leq 24$ and $1 \leq j \leq 24$ for a 24 x 24 dot Chinese character bit pattern.



a. The black dot & white dot of a Chinese character

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

b. The bit pattern of a Chinese character

Figure 2.2 Matrix Representation of a Chinese Character

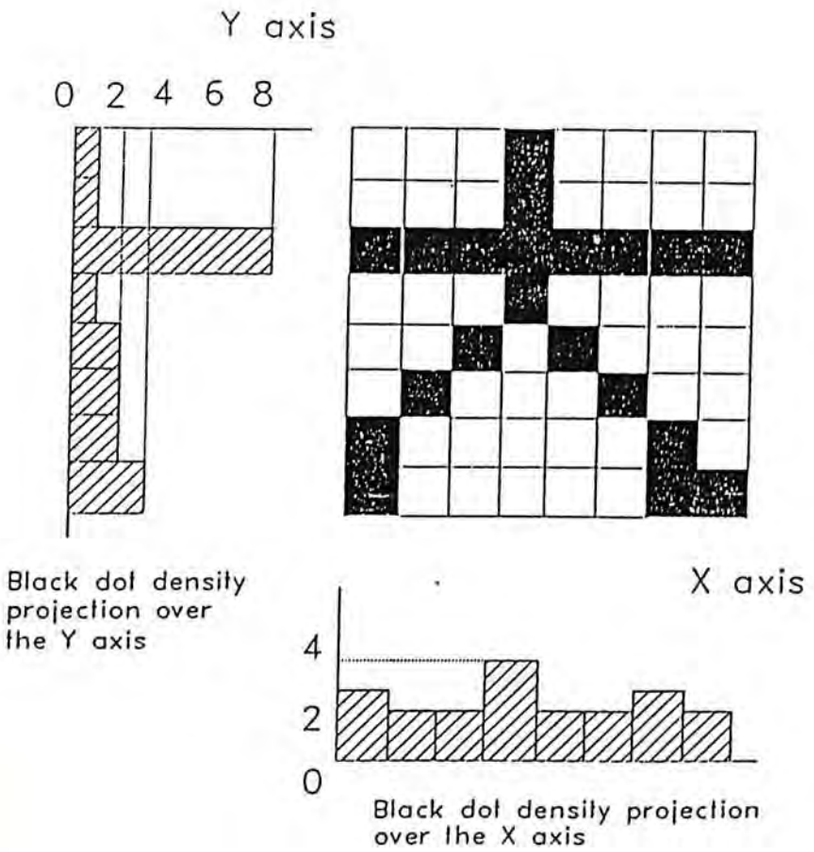


Figure 2.3 The black dot density projection profile over the X & Y axis

For x projection

the projection profile over x axis is

$$X(i) = \sum_{j=1}^{24} P(i,j) \qquad i = 1 \text{ to } 24$$

the mean value of the black dot density is

$$\bar{X} = \frac{1}{24} \sum_{i=1}^{24} X(i)$$

the kth moment of the projection profile are

$$m_k = \sum_{i=1}^{24} (X(i) - \bar{X})^k \qquad k=1,2,3 \text{ moment}$$

Similarly, for y axis projection

the projection profile over y axis is

$$Y(j) = \sum_{i=1}^{24} P(i,j) \qquad j = 1 \text{ to } 24$$

the mean value of the black dot density is

$$\bar{Y} = \frac{1}{24} \sum_{j=1}^{24} Y(j)$$

the kth moment of the projection profile are

$$m_k = \sum_{j=1}^{24} (Y(j) - \bar{Y})^k \qquad k=1,2,3 \text{ moment}$$

§2.2.6 Corner Feature

One of the characteristic of a Chinese character is its outline figure which can be used as a preliminary classification of the printed Chinese characters. The peripheral feature is to represent this outline figure for an input pattern.

There are many types of peripheral feature extraction methodology suggested in [10] [11] and each has their own advantage for a particular application. The 4C code described previously is one of an example.

In this project, another type of peripheral features called Corner Feature is suggested. The features are extracted in the following way

a. Determine the boundary of the character in the character bit map (figure 2.4).
b. Within the character boundary, dividing the character frame into four equal parts i.e. corner part.
c. For each corner part, sub-divide the frame into four line parts both horizontally and vertically from the character boundary. Since the distance of the character boundary (both horizontal and vertical) is not the same for each character, the following calculation should be adopted to determine the suitable feature positions. Refer to figure 2.4.

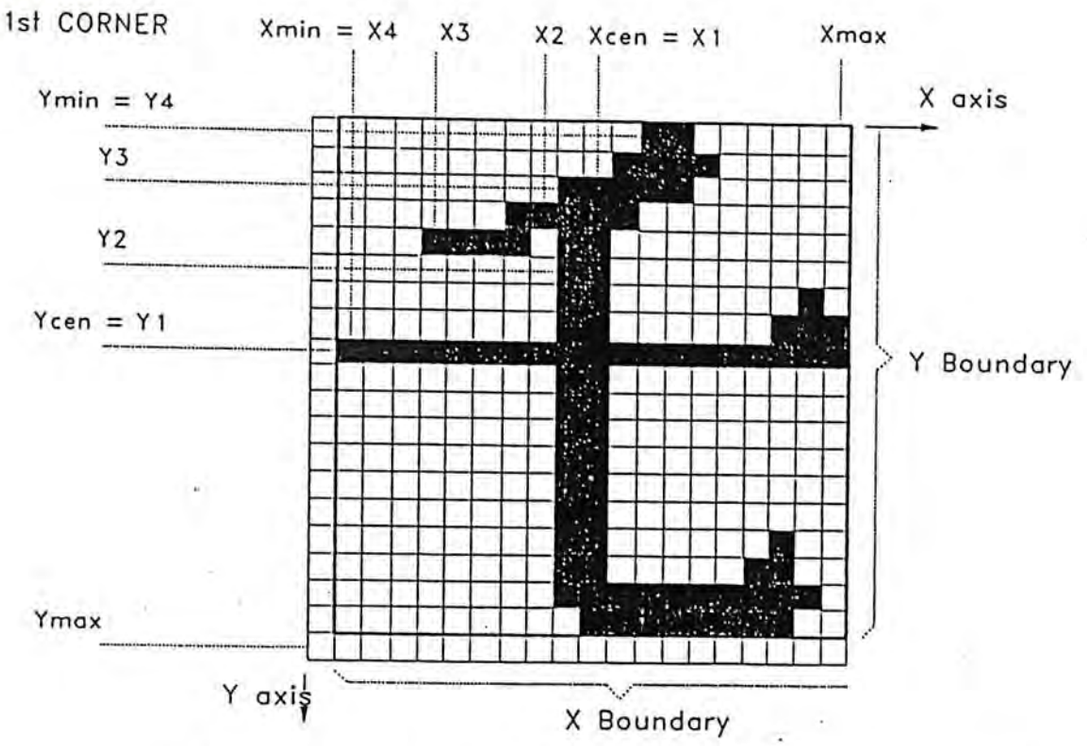


Figure 2.4 Corner Feature Extraction of a Chinese Char

If P(i, j) is the bit pattern of the Chinese character, then let

Xmax the distance from the edge of the character bit pattern to the right boundary of the character.

Xmin the distance from the edge of the character bit pattern to the left boundary of the character.

Ymax the distance from the top edge of the character bit pattern to the bottom boundary of the character.

Ymin the distance from the top edge of the character bit pattern to the upper boundary of the racter.

Therefore,

$$Xcen = INT\left[\frac{(Xmax + Xmin)}{2} + 0.5\right]$$

$$Ycen = INT\left[\frac{(Ymax + Ymin)}{2} + 0.5\right]$$

$$Xoffset = INT\left[\frac{(Xcen - Xmin)}{3} + 0.5\right]$$

$$Yoffset = INT\left[\frac{(Ycen - Ymin)}{3} + 0.5\right]$$

For each corner part, the position of each feature point can be calculated by means of Xmax, Xmin, Xcen, Ymax, Ymin and Ycen point. To illustrate the calculation, the first corner is determined by the formulas shown below

At x axis, the position of feature extraction points are

X1 = INT (Xcen + .5)

X2 = INT (Xcen - Xoffset + 0.5)

X3 = INT (Xcen - 2 x Xoffset + 0.5)

X4 = INT (Xcen - 3 x Xoffset + 0.5)

At y axis, the position of feature extraction points are

Y1 = INT (Ycen + 0.5)

Y2 = INT (Ycen - Yoffset + 0.5)

Y3 = INT (Ycen - 2 x Yoffset + 0.5)

Y4 = INT (Ycen - 3 x Yoffset + 0.5)

d. Calculate the distance (number of dots) from the boundary to the character (i.e. the first change from white to black) in the direction from top to bottom and from left to right as shown in figure 2.4.

e. Normalize the feature value with respect to the actual size of the character. This is used to cater for the different size of character over the same bitmap.

For each corner part, there will be eight corner features at which four features are taken at the x axis and four features are taken over the y axis. The following formulas shows the first corner features of a character.

$$F1 = \sum_{i=1} P(i, Y1) / (Xcen - Xmin)$$

$$F2 = \sum_{i=1} P(i, Y2) / (Xcen - Xmin)$$

$$F3 = \sum_{i=1} P(i, Y3) / (Xcen - Xmin)$$

$$F4 = \sum_{i=1} P(i, Y4) / (Xcen - Xmin)$$

$$F5 = \sum_{j=1} P(X1, j) / (Ycen - Ymin)$$

$$F6 = \sum_{j=1} P(X2, j) / (Ycen - Ymin)$$

$$F7 = \sum_{j=1} P(X3, j) / (Ycen - Ymin)$$

$$F8 = \sum_{j=1} P(X4, j) / (Ycen - Ymin)$$

The corner feature is particularly useful in our tree classifier because of the following merit

a. The calculation is simple and fast.
b. It is insensitive to the peripheral variation and thus can be applied to both printed and non-printed Chinese character.
c. It is insensitive to the shift of the character over the x or y axes.

However, the corner feature has also the following drawbacks

a. It is sensitive to the rotation of the character.
b. Characters with similar peripheral features cannot be identified.

§2.3 CLUSTERING METHOD - K-MEANS & OTHER ALGORITHMS

After appropriate features have been extracted from the inputted character, our next task is to find a way to cluster the characters into various groups based on the feature values just computed. Restating the problem in another way, our task here is, with the basis of the features values obtained, how can we define the likeliness of the objects so that individual characters can be grouped together? The answer to the above question is an algorithm we usually refer as clustering algorithm.

One of the most popular clustering algorithms used is the K-MEANS algorithm [12] first suggested by Macqueen in 1967. This can be improved by incorporating a coarsening and a refining parameter. Define C to be the coarsening parameter for group splitting and R be the refining parameter for group merging. The algorithm is summarized below.

K-MEANS (with coarsening and refining)

1. Choose $x_1, x_2, \ldots, x_k$ as initial k cluster centres.
   Let n_i = number of data in the ith cluster
   Set $n_i = 1 \quad \forall\ i = 1,2,\ldots,k.$
2. Take x from the data set, while not end of file do
   2.1 Calculate $d(x,x_i) \quad \forall\ i = 1,2, \ldots, k$
   2.2 Find $d(x,x_j)$ such that
   $$d(x,x_j) = \min_i d(x,x_i)$$
   2.3 Group Splitting
   if $d(x,x_j) > C$ /* group splitting */
   $$k \longleftarrow k+1$$
   $$x_{k+1} \longleftarrow x$$
   $$n_{k+1} \longleftarrow 1$$
   else /* assign x to the jth cluster */
   $$x_j \longleftarrow \frac{n_j x_j + x}{n_j + 1}$$
   $$n_j \longleftarrow n_j + 1$$

2.4 Group Merging

Check whether merging between groups is possible.

Repeat for each pair of groups

if $d(x_i, x_j) < R$ /* group merging */

$$x_i \longleftarrow \frac{n_j x_j + n_i x_i}{n_j + n_i}$$

$$n_i \longleftarrow n_i + n_j$$

$$k \longleftarrow k - 1$$

Relabel all cluster centres and the number of data in the cluster

Until no merging has occurred.

3. The classification process is repeated for the entire data set with the cluster centres and values of k just calculated.

This algorithm can also be improved by repeating the entire algorithm until convergence of the number of cluster and the value of the cluster centres is achieved. Again such modification is not justified by the tremendous increase in the training time of the modified algorithm.

Another implementation concern in the above algorithm is the choice of the values of C and R. Careful choices of C and R are important and one possible suggestion is to take C and R to be some fraction and multiple of the standard deviation. In our study, since the given data are of multivariate in nature, several values of the standard deviation have to be

combined as one single measure. If P is the dimension of the feature space and σ_i is the standard deviation of the ith variate, then we may takes

$$\sigma = \sum_{i=1}^{P} \sigma_i \; / \; m$$

$$C = 0.1\,\sigma$$

$$R = 2\,\sigma$$

Beside K-MEANS, classically there are still quite a number of clustering algorithm which also have the characteristic of self adjustment in the number of clusters formed. These include the maximium-distance algorithm and the ISODATA algorithm [13]. They will be discussed in the appendix.

§2.4 Pros and Cons

All of the above mentioned clustering methods are examples of the algorithms we commonly used in unsupervised learning. There are a number of advantages. Firstly, these algorithms are relatively simple and are thus easier to put to practical implementation by computer. Their algorithmic nature and simple mathematical computation required render the coding of the algorithms fast and simple. Besides, no prerequisite knowledge is necessary. We need not basically know anything about the class membership of all the data before the training process begins. This distinguished merit is practically important since in a lot of cases nothing is known about the given training data set.

Despite of the advantages we have just mentioned, there are quite a number of disadvantages preventing us from using the approach of unsupervised learning. Inevitably the clustering result depends very much on the choice of the initial parameters such as the initial cluster centres, the initial number of clusters, the coarsening and the refining parameters. Besides, the order of the data arranged in the input stream will also affect the result since undoubtedly when the order of data is changed, different situations of merging and splitting may be resulted during every step of the algorithm. As a result, these algorithms are too sensitive and the result will be subject to fluctuation.

The other disadvantage of these algorithms is that the processing time for these algorithms is usually large if they are to run after convergence is achieved, despite the relative simplicity in coding. Of course, there is also another difficulty as mentioned previously that there is no guarantee about the convergence of the algorithm.

The most important disadvantage of the unsupervised approach is that the clustering result is usually not good enough when compared with that of the supervised approach. The lack of prerequisite knowledge is on one hand a favourable condition for implementation while on the other hand, renders training more difficult. The more you know, the more you get. Hardly can we derive extra merits if we

are only given very limited information and knowledge. Also the unpredictable nature in the resulting number of clusters present an additional difficulty towards the implementation of all these unsupervised learning algorithms.The above discussion reveals that the approach of unsupervised learning, although easy to put to implementation, is not reliable in its performance and is therefore not preferable if comparison is to be made with the approach of supervised learning which will be discussed in the next chapter.

§2.5 DECISION TABLE

Adopting the approach of unsupervised learning and the use of feature extracted from the Chinese character, we are going to explore in this section, how a multistage classifier can be built from the clustering results.

Suppose we have adopted to use one particular clustering algorithm (say the K-MEANS with coarsening and refining) to do the clustering. Quite a large number of features can be extracted from a character, each may be specialized in distinguishing characters of some particular nature. If for each of the features, we perform the clustering process for one time, then we can develop an algorithm for the building up of a multistage classifier by gathering all the clustering results recorded.

Let m be the number of features used and in particular we designate these m features by F_1, F_2, ..., F_m. For each of these features, one particular clustering result is provided. By exploring the clustering result provided, we can determine which cluster is one particular character being classified to. If k_i is the number of resulting cluster of the ith feature, i.e. F_i, and we label all these clusters as $F_i[1]$, $F_i[2]$, ..., $F_i[k_i]$, then each character can associate with one particular cluster for every feature under consideration. Viewing these cluster labels as some codes, a decision table can be formed which will summarize all the clustering results of the features in association with the input data. Each line in the decision table will just give the coding associated with the clustering results of all the features of one particular Chinese character. As a result, the table obtained will depict the following form.

char_id	codes
a440	$F_1[2]$ $F_2[3]$ $F_3[4]$ .... $F_m[2]$
a441	$F_1[2]$ $F_2[4]$ .... $F_m[1]$
.	.
.	.
.	.

After the decision table is formed, our next task is to search for a classifier with best performance from this table.

§2.6 THE OPTIMUM CLASSIFIER AND ITS IMPLEMENTATION DIFFICULTIES

We want to construct a tree classifier from the decision table where every internal node will denote a decision with one particular feature among all available features we have been using. How can we quantify the optimality of such a classifier?

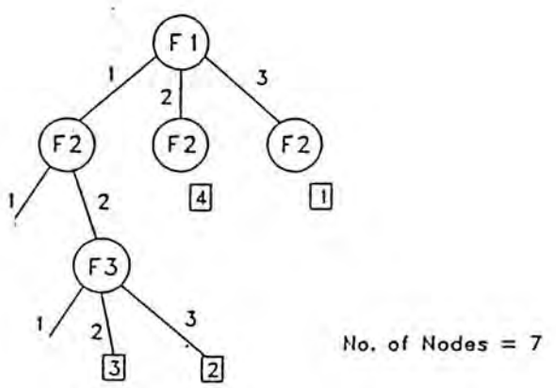
Based on which cluster the datum is belonging, each character can be determined using a code which represents the cluster membership of the datum if all the features are used. Such codes have already be formed in the decision table. If m is the total number of features used, then the length of the codes will also be m. This situation can be pictured as a balanced tree with depth m and each terminal node will uniquely determine a code for characters. Of course, there may be redundancies in such a tree. Suitable rearrangement of the nodes can significantly prune unnecessary branches and as a result an optimal decision tree can be obtained. In such an optimal decision tree, terminal nodes will not necessarily have a depth of m from the root node and the length of code will become shorter. Optimum code length can then be obtained.

Take an example, consider the case of classifing 4 characters with 3 different features. Suppose the following decision table which summarize the clustering results has been found.

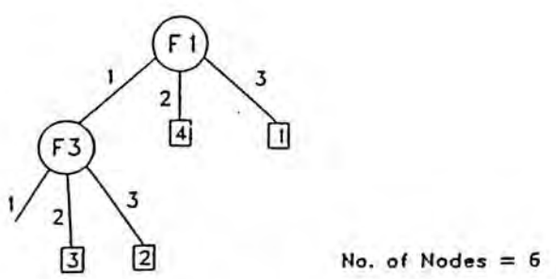
character id	code
1	$F_1[3]$ $F_2[2]$ $F_3[1]$
2	$F_1[1]$ $F_2[2]$ $F_2[3]$
3	$F_1[1]$ $F_2[2]$ $F_3[2]$
4	$F_1[2]$ $F_2[1]$ $F_3[2]$

As shown in figure 2.5a, feature 1 (F1) is being considered first and then feature 2 (F2) and feature 3 (F3) and the resulting decision tree is constructed as shown in the figure, the total number of nodes will be 7.

However, if feature 3 is considered before feature 2, then the structure of the decision tree will be different as shown in figure 2.5b. It is noticed that the total number of node in this case is reduced to 6.



a. F1 -> F2 -> F3



b. F1 -> F3 -> F2

Figure 2.5

Following the above argument, we can claim that the optimum tree classifier is one which has the least number of nodes, both internal and terminal added together. The smaller the number the nodes, the least amount of computation and the least number of features are required in the discrimination procedure. To facilitate the search for the optimum, the following dynamic programming model is constructed.

Dynamic Programming Formulation

Let n = the total number of features used

$\bar{S}$ = initial data set for classification

$A = \{0,1\}$

$\bar{v} = (0,0, \ldots, 0) \in A^n$

= initial status of the features used

where the ith entry $= \begin{cases} 1 & \text{ith feature has not been used} \\ 0 & \text{ith feature has aleady used} \end{cases}$

$e_i = (0,0, \ldots, \underset{\text{ith position}}{1}, \ldots, 0) \in A^n$

indicates that the ith feature has been used while all others still remain unused.

$\Omega_S^{\,i}$ = set of clusters formed by using the ith feature on $S \subset \mathcal{P}(S)$ which is the power set of S

Assumption:

If every subtree contains the minimum number of nodes (both internal and terminal), then the whole tree contains the minimum number of nodes.

Note that this assumption is actually valid.

Let f(S,v) minimum number of nodes both internal and terminal, needed in the optimal decision tree for classifying data set S given that the status of features already used is represented in v.

Then $\forall\ S \subset \bar{S}\ ,\quad v \in A^n$

$$f(S,v) = \min_{i \in \{i : e_i \wedge v = 0\}} \left(1 + \sum_{T \in \Omega_S^i} f(T, e_i \vee v) \right)$$

where $\wedge$ denotes the logical and operator and

$\vee$ denotes the logical or operator.

Boundary conditions:

$$f(S,v) = 1 \quad \text{if } |S| = 1 \text{ or } v=(1,1, \ldots, 1)$$

Our target is to find $f(\bar{S},\bar{v})$.

Although the problem of obtaining the optimal decision tree can be formulated in the above format, the time complexity for arriving at the optimum requires exponential time and belongs to the class of NP complete problems. As a result, practically the above problem is actually unsolvable.

Due to the abovementioned implementation difficulties especially when all the character set (i.e. 13,000) are to be recognized, another approach i.e. Supervised learning approach will be studied in more detail.

A.3 Other Algorithms (maximum-distance & ISODATA)

MAXIMUM-DISTANCE Algorithm

The maximum (maximum-minimum) distance algorithm is another simple heuristic procedure based on the Euclidean distance concept. In the first step, we arbitrarily choose one object to be the first cluster centre. Next, we determine the farthest sample from this cluster centre and let it be the second cluster centre. In the third step we compute the distance from each remaining objects to these two clusters. For every pair of these computations we save the minimum distance. Then we select the maximum of these minimum distances. If this distance is an appreciable fraction of the distance between original two cluster centres, the object will form a new cluster centre. Otherwise, the algorithm will be terminated.

In the next step, we compute the distance from each of the three established cluster centres to the remaining objects and save the minimum of every group of three distances. Then, we again select the maximum of these minimum distances. A new cluster centre will be formed if this distance is an appreciable fraction of the "typical" previous maximum distances. Otherwise the algorithm is terminated. A useful measure of the typical previous distances is the average of these distances. The entire process will be repeated again until the condition for termination is reached.

ISODATA Algorithm

The Isodata (abbreviation of Iterative Self-Organizing Data Analysis Techniques A) algorithm presented in this section is similar in principle to the K-MEANS procedure in the sense that cluster centres are iteratively determined sample means. Unlike the latter algorithm, however, Isodata represents a fairly comprehensive set of additional heuristic procedures which have been incorporated into an interactive scheme.

Before executing the algorithm it is necessary to specify a set N_c of initial cluster centres $z_1, z_2, \ldots, z_{N_c}$ and K to be the number of desired clusters. The algorithm can be described as follows.

1. Distribute objects among the present cluster centres by choosing that cluster which corresponds to the minimum Euclidean distance.
2. Discard clusters with fewer than θ_N members where θ_N is predetermined.
3. Update all cluster centres.
4. Compute the average distance $\bar{D}_j$ of data from their corresponding cluster centres.
5. Compute the overall average distance $\bar{D}$ of data from their respective cluster centres.
6. If this is the last iteration, set lumping parameter θ_c = 0 and go to step 9.

If $N_c \leq K/2$, go to step 7.

If this is an even-numbered iteration or if $N_c \geq 2K$, go to step 9; otherwise continue.

7. Find the standard deviation vector σ_j for each clusters and find the maximum component of each σ_j and denoted it as σ_{jmax}.

8. If $\sigma_{jmax} > \theta_s$ where θ_s is a prespecified standard deviation parameter and

   a. $\bar{D}_j > \bar{D}$ and $N_j > 2(\theta_N+1)$ or

   b. $N_c \leq K/2$

   then cluster j is splitted.

   If splitting took place in this step, go to step 1; otherwise continue.

9. Compute pairwise distances between all cluster centres.

10. Compare all pairwise distances against lumping parame θ_c. Arrange the L (L being a preassigned value) smallest distances which are less than θ_c in ascending order.

11. Pairwise lumping is performed for the smallest dista in step 10.

12. If this is the last iteration, the algorithm terminates. Otherwise, go to step 1 for the next iteration.

It should be noted that Isodata is an extremely complex algorithm and in general requires extensive experiment-ation before we can arrive at any meaningful conclusion.

A4 POSSIBLE IMPROVEMENT

§4.1 TRAINING AND TEST SAMPLE REDUCTION

One possible improvement for the multistage tree classifier developed is to reduce the number of training sample items. We have been using the entire dictionary of the Eten Chinese system which composes of around 13,000 Chinese characters. However it has been noted that many of these characters are actually very seldom used. Common chinese characters only amounts to about 5,000 chinese characters. As a result, it would be better for us to choose a smaller set of characters in our stage of training. The use of a smaller training data set will significantly reduce our training time required. On the other hand, it also increases the discriminating power of the classifier since the number of characters of similar shape decreases at the same time.

Besides, the testing sample can also be reduced so that statistic on the performance of the classifier can be evaluated more easily in a shorter period of time. In fact, if the testing sample items are randomly selected, the result will still be reliable and the performance of the classifier can accurately be evaluated.

§4.2 NOISE FILTERING

As mentioned in section 3.6.3, the performance of the recommended tree classifier under noisy sample is rather

poor when compared with the original training character. One of the possible reasons may be due to the sensitivity of the corner feature to noise (both random noise and printed noise). Therefore, reduce the noise of the noisy sample may improve the performance of the classifier.

To reduce the noise, two noise filtering algorithms has been derived. They make use of the fact that the probability of white noise generated in the character bit pattern is independent to its neighbourhood bits. With the noise generated of this nature, there will be three possible cases.

Case_1: The noise is generated near the character key stroke. In this case, one or more black dot/s will occur at the neighbour of the noise bit as shown in figure 4.1.

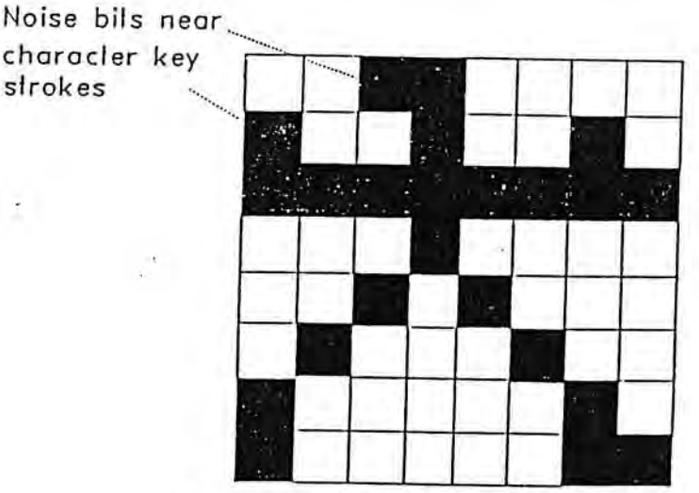


Figure 4.1 Case 1 : Noise bil near lhe charocler key strokes

Case_2: The noise is generated at the area other than the key stroke in a way that no other noise bit occurs at the neighbour of the noise bit as shown in figure 4.2.

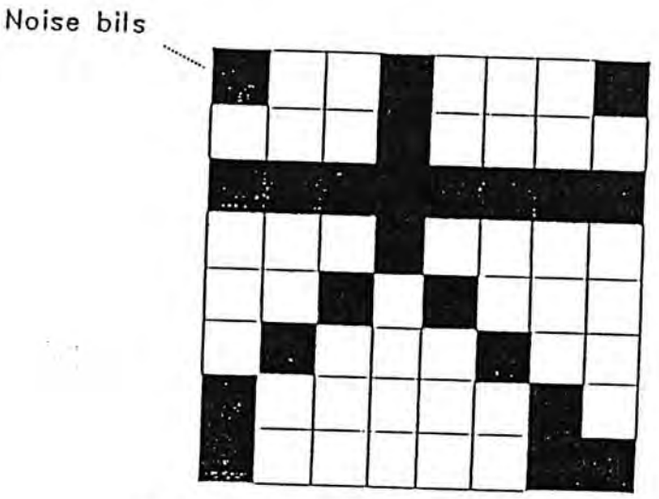


Figure 4.2 Case 2 : Noise bil generale al lhe area olher lhan key slrokes

Case_3: The noise is generated at the area other than the key stroke. At the same time, another noise bit/s occurs near the noise bit as shown in figure 4.3.

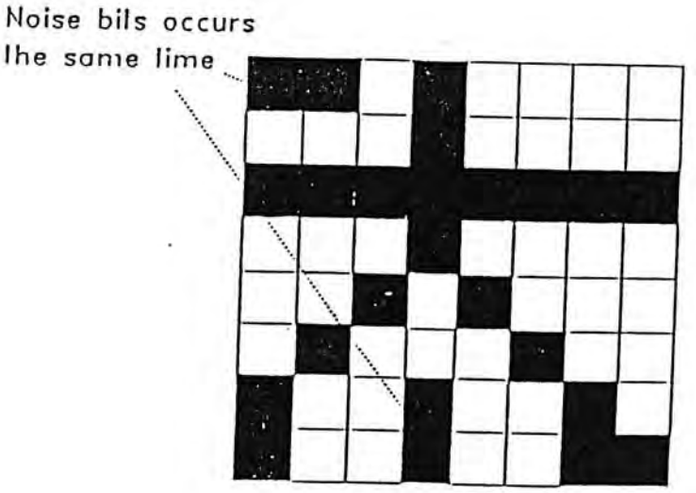


Figure 4.3 Case 3 : Noise bil occurs al lhe same lime

It can be shown that the noise generated in case 1 and case 3 are difficult to remove as there is not enough information to determine whether the neighbouring black dots are the actual key stroke or just another noise bit(s).

ALGORITHM 1

This filtering algorithm provides the capability of eliminating a single bit random noise (case 2) over the character bit pattern. By adjusting the values of the threshold δ, some special noise in case 3 can also be eliminated. But error may also occur in removing the black dot of the key stroke which is not desirable in the 24 x 24 character dot pattern.

The details of the filtering algorithm is explained as follows.

Let A(i,j) be the character bit pattern such that

$$A(i,j) = \begin{cases} 0 & \text{for white dot} \\ 1 & \text{for black dot} \end{cases}$$

where $1 \leq i, j \leq 24$.

For each value of i and j (refer figure 4.4), if A(i,j) = 1, then the black intensity over its neighbour is

$$\begin{aligned} I(i,j) = 1/9 \{ & \alpha A(i,j) + A(i-1,j) + A(i+1,j) \\ & + A(i,j-1) + A(i-1,j-1) + A(i+1,j-1) \\ & + A(i,j+1) + A(i-1,j+1) + A(i+1,j+1) \end{aligned}$$

A(i−1, j−1)	A(i, j−1)	A(i+1, j−1)
A(i−1, j)	A(i, j)	A(i+1, j)
A(i−1, j+1)	A(i, j+1)	A(i+1, j+1)

Figure 4.4

If $I(i,j) \leq \delta$, A(i,j) may be the noise bit and will be eliminated. Otherwise, A(i,j) remain unchanged.

ALGORITHM 2

As mentioned in previous paragraphs, algorithm 1 can only filter the single bit noise (case 2). To filter the noise bit in case 3 with algorithm 1, there will be a probability of filtering the character key stroke instead of the noise bit. To improve the suitation, this algorithm is suggested.

With this algorithm, the consecutive noise bits and single noise bit as shown in figure 4.3 can be filtered.

In this algorithm, two change counters i.e. horizontal change counter and vertical change counter are introduced. A change here refer to the change from white dot (bit 0) to black dot (bit 1) or vice versa. For each bit in the character bit pattern, the surrounding bits in figure 4.5

are being considered. The horizontal change counter will store the number of changes in the horizontal direction and the vertical change counter will store the number of changes in the vertical direction.

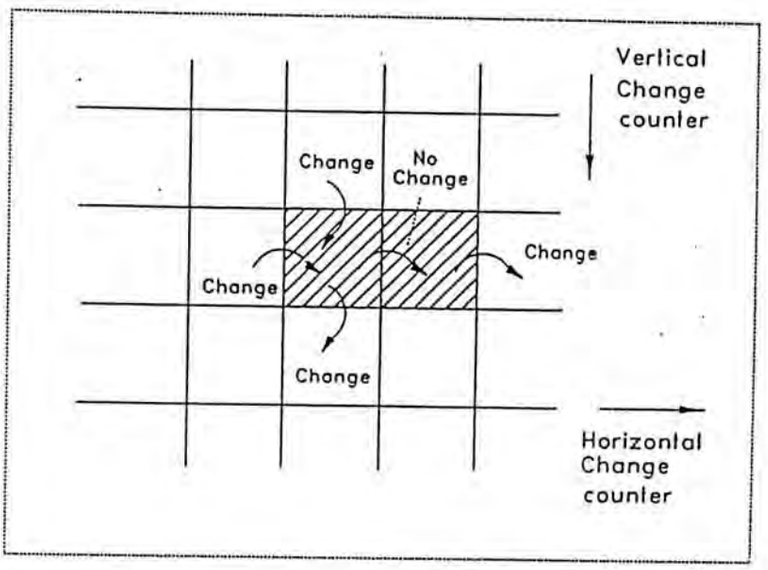


Figure 4.5 Noise Filtering Algorithm 2

To filter the mentioned noise bits, the following conditions has to be satisfied.

a. The horizontal change counter ≥ 1 and the vertical change counter ≥ 2.
b. The horizontal change counter ≥ 2 and the vertical change counter ≥ 1.

Even with this noise filtering algorithm, the noise coherent to the character key stroke cannot be eliminated.

The performance parameters described in section 3.6.1 will be measured with the noisy samples being filtered by each algorithm so that the degree of improvement after the noisy sample has been filtered can be determined.

§4.3 DECISION WITH OVERLAPPING

Our previous approach in clustering the Chinese character set has been relying heavily on the corner codes of the character. Basically classification to and from groups are based on the corner codes and misclassification actually refers to the state of misclassifying the character with some other corner code other than its own one. Errors are measured in terms of such misclassifications and it has been pointed out that as the depth of the tree grows such errors will be propagated and magnified. How such errors are tackled is the theme of this section.

Although errors will be propagated to the terminal nodes of our tree classifier, such errors are not actually real errors. Notice that our objective here is only to cluster the original data sets into some numbers of groups. Corner codes have been helping in the process of building the classifier but we are not using the corner codes in our future discrimination process. Characters at each of our terminal nodes do not necessarily have the same corner code but they should be similar in the sense that the feature

values are very close to each other. As a result to the above arguments, no error handling procedures have to be taken.

Based on the above approach, tree classifier has been constructed which shows that Chinese characters associated with each terminal node are really very similar in shape. However we can still explore some error handling steps so that the similarity provided by the corner code can be maintained and even improved. As a result the idea of overlap has been proposed. With the idea of overlap, we will simply duplicate those data items which have been found to be uncertain in the decision and are sources of errors. They will be classified to both of the subnodes of the parent node.

Consider a scenerio designated by figure 4.6.

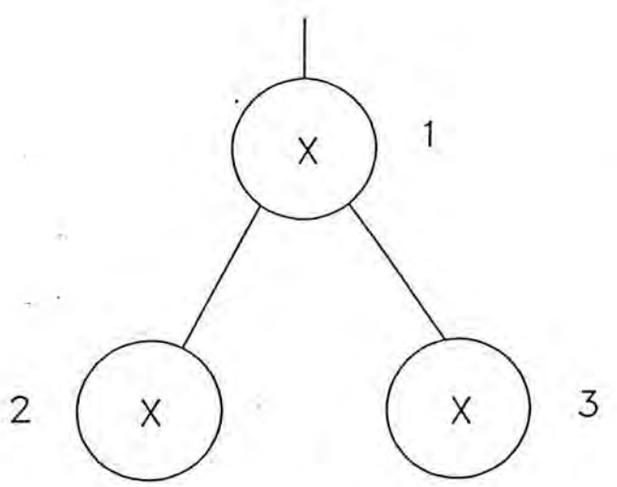


Figure 4.6

Suppose x has to be classified to either node 2 or node 3. Suppose the corner code of x belongs to the grouping revealed in node 2, then x should be classified to node 2 if no error is to be committed in the classification. However if it just happens that the use of the discriminant function classifies x to node 3, misclassification occurs. In such case we will duplicate x to node 2 since the position of x is uncertain. It can be in node 2 or in node 3 reflecting the situation of overlapping as illustrated by figure 4.7.

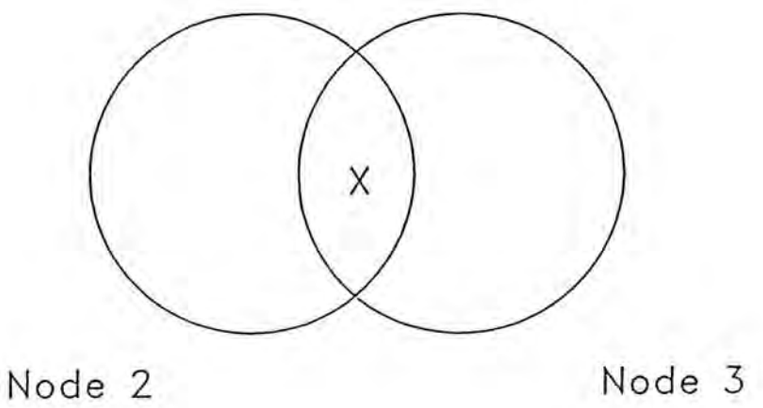


Figure 4.7

Overlapping will delay the committing of errors to subsequent stages of classification and will on the other hand increases the number of data items residing in the subsequent nodes in the tree. If the depth of the tree classifier is large, such technique is not feasible and a relatively large number of data items will be associated

with all the terminal nodes. However, as we have been adopting a tree classifier with depth less than ten which is actually not quite a large value, the use of the idea of overlap is justified and has been implemented.

IMPLEMENTATION REMARKS

It has been noticed that when overlapping is introduced, one single character can exist at more than one terminal node. Since each terminal node will be designated by a unique code, this single character will be represented by more than one code. Such increase in the number of distinct codes in the system will not be too difficult to manipulate when only one single tree is under consideration. When three separate trees are used, if for one particular character which has n distinct codes in the first tree, n in the second tree and n in the third tree, then the total number of distinct codes corresponding to this character will become n x n x n which will be too large for easy manipulation. Take an example when there is 2 codes (say 01 and 02) in the first tree, 2 codes (say 01 and 03) in the second tree and 1 code in the third tree (say 04). The possible combined codes include:

010104 010304 020104 020304

which amounts to $2 \times 2 \times 1 = 4$. As have been pointed out previously in section 3.5.4, the lookup code table will be read in during run time and the codes will be stored in an

array and sorted in ascending order so that subsequent searching by binary search can be done. However, since in our case here the number of codes is too large, we cannot store all of them in a single array. As a result, we have to use a file in the hard disk as the storage. The codes will be sorted and stored in an file permanently before the program is executed. Constant lookup from this file is required for searching. Of course, such constant lookup from file in the hard disk will increase the processing speed significantly. The experimental results will be given in chapter 5.

§4.4 BACK TRACKING FOR HOLES

We have been using three separate trees in our training process, each of which animating one of the corner of the three corner code. Suppose we are only interested in discriminating the around 5,400 commonly used Chinese characters. In our study tree architecture of three separate trees each with a depth level of four has been adopted. We can calculate the total number of distinct codes which can be formed within such framework. Follow similar argument given in section 3.5, the total number of distinct codes can be amounted to $16 \times 16 \times 16$ which is 4,096. Of course, if each code is only associated with one character, we can notice that all these 4,096 codes are not enough and theoretically they should all be used up and some codes

should even have to represent more than one character. To represent more than one character by one code is really not a problem since as we have suggested in previous chapters, the decision associated with this multistage tree classifier is only our first stage of classification and further discrimination is required. What really concern us is that not all the 4,096 codes will be consumed by the 5,400 Chinese characters. Instead some codes may be left untouched by the characters which we can refer them as holes". It can easily be noticed that holes are inevitable since similar characters will be clustered together and get the same code. In Chinese character recognition system, there are quite a number of different groupings of similar characters. As a result, the actual codes used will drop drastically resulting in the emergence of a large number of holes.

Hole is both a good and a bad feature in our discrimination process. The bad thing of hole is that if errors have been made in determining the code in each of the three tree classifier, the outcoming code may not be a realistic one. In other words, the code calculated may not exist and correspond to any particular group of characters. As a result, no decision can be made and it seems that the entire classification process will be a failure. On the other hand, the existence of such holes is beneficial to us since it provides a channel for trapping errors which are unavoidable in usual discrimination system. Whenever unrealistic code is

computed, errors must have been committed in our previous stages of classification. This is regarded as a good feature since it provides a signal for us to signify the committing of errors.

One should note that errors can still be committed and remained unnoticeable to us. Codes for the three tree protocol may be incorrectly computed independently while the final combined code still remains realistic and corresponds to some particular characters. The presence of holes cannot capture this type of error and we can by no means correct such mistake since we do not simply reckon its presence. In section 4.5, we will describe one method which will reduce the probability of the emergence of such mistake. Meanwhile, let's concentrate ourselves on the problem of holes.

We have claimed that hole is a good feature, but it will still not be good if the bad feature of uncertainty in terms of the recognition result has not been solved. As errors has been occurred, we may wish to fix the errors so that codes can be corrected. The search for such mistake can easily be achieved through the idea of back tracking. Since three separate trees have been used, back tracking on all these three trees is required. Assuming that errors committed at greater depth are more likely than errors committed at the top levels, we can devise a back tracking algorithm which will back track on the three trees successively for an increasing number of levels until a realistic code is

obtained. To simplify the actual searching time of the algorithm, only one error is assumed. The complete algorithm can be described as follows.

1. Repeat for back track level i = 1, 2, 3, 4
   /* only 4 levels in the tree*/
2. Repeat for corner j = 3, 2, 1
   2.1 Back track for i levels of the tree corresponding to corner j by going up for i levels and assuming decision has been made wrongly at that node.
   2.2 Tree code for other corners remain unchanged.
   2.3 Get the new code and determine if this code is realistic.
   2.4 If it is a hole, continue by going to step (2) again.
   2.5 Otherwise, exit the algorithm and report the new code.

The above idea has been implemented and tried out in the computer.

§4.5 FUZZY DECISION FUNCTION WITH TOLERANCE LIMIT

This section wants to tackle the problem of unnoticeable errors which may have committed during the process of classification. The entire idea lies in the use of a fuzzy decision based on the introduction of tolerance limit.

Our previous discussion has been completely based on the use

of a multistage tree classifier. By a multistage classifier, complete decision is to be made by a number of successive decision which in our case is represented as the linear discriminant function at each of the internal nodes of the tree. For simplicity, we will consider only the case of one single tree and the problems associated with the three separate tree protocol will follow in the same way.

The decision at each node has been characterized by a linear discriminant function which is actually representing hyperplane dividing the feature space into two regions, one representing the left subnode of the parent node while the other the right subnode. Here the decision is definite and clear cut and object has to be classified to either one of these two regions even when the distance of the object from the hyperplane is very small. As we have pointed out previously, committing error is inevitable. So as to reduce the chance of obtaining unnoticeable errors, it will be better if we do not provide such a definite decision for the internal nodes. If the distance between an object and the hyperplane is too small, preferably smaller than a preassigned threshold value which we will call the tolerance limit, then decision of classification to which region is not made. Rather the decision will be delayed. But when can we decide and how should we decide?

Denote the tolerance limit by ε. The previous argument claims that decision cannot be finalized if the object lies

within a distance of ε from the hyperplane. Three regions will be resulted as shown in figure 4.8. The shaded region corresponds to the region where decision will be delayed. To determine the decision in this region, we will choose to quantify an error measure. The object will be classified to the region if the subsequent total error accumulated is a minimum. We will define error to be the distance between the object and the hyperplane if the object is residing in the "not-yet-determined" region. However, if the object is just inside one of the "decided" regions, no error will be taken.

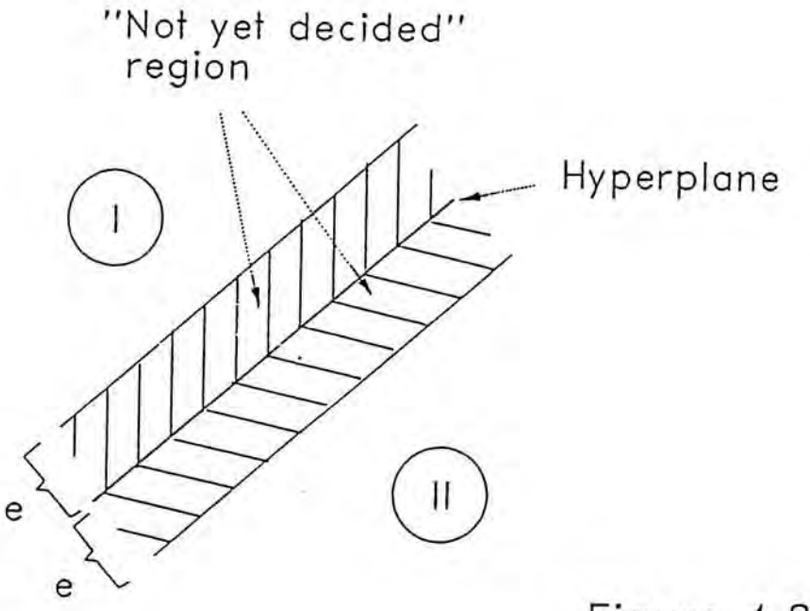


Figure 4.8

With the basis of the above discussion, the following algorithm can be formulated.

Let any particular node be designated as n. Suppose

$y_n(x)$ = linear discriminant function associated with node n

x_o = object vector

E(n) = error measure associated with the node n.

We will define a recursive procedure named Error(.) which takes a node as input and returns the error associated with this node.

```
procedure Error(n)
{
  If n is a terminal node then
  return 0
  else
  {
     If  | y (x ) | < ε  then   /* not-yet-decided region */
         left   ←——  left subnode
         E_left ←——  Error(left)
         right  ←——  right subnode
         E_right ←——  Error(right)
     return  min(E_left, E_right) + | y (x ) |
     else  if  y (x ) > ε  then   /* left subnode */
         left   ←——  left subnode
         return  Error(left)
```

```
        else  if  y (x ) < -ε  then  /* right subnode */
              right  ←——  right subnode
             return  Error(right)
        }
    }
```

§4.6 DIFFERENT TREE ARCHITECTURE

By the way, we have been using the three separate tree protocol in our analysis. In this section, we will explore some other tree architecture which may be more favourable in our current situation.

One of the possible alternatives to our present three separate tree protocol is a tree architecture which will combine the three trees to a single one. Suppose the tree for the first corner has been constructed. We can further develop at the terminal nodes of the first tree for a few levels of depth by the consideration of the second corner feature. Similarly the final tree will also be further extended at the terminal nodes by the consideration of the third corner feature. This 1-tree protocol is simple and straightforward although there is no guarantee that it is a good one. Figure 4.9 shows the configuration of the 1-tree protocol.

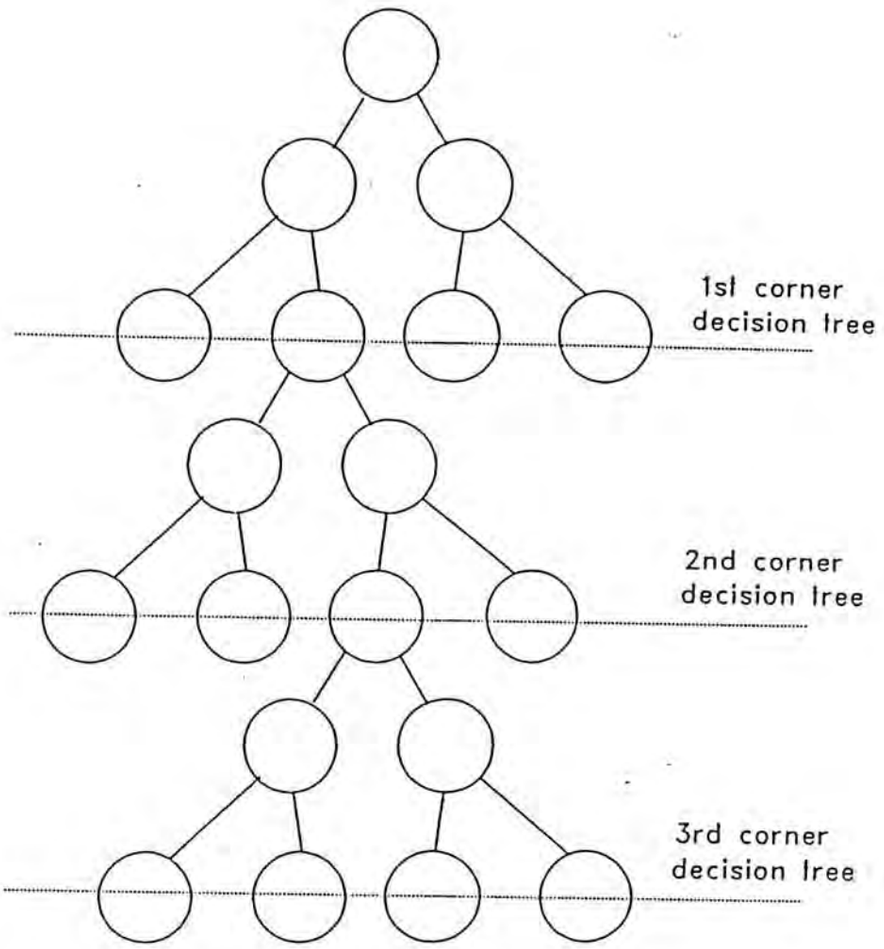


Figure 4.9 1–Tree Protocol

To improve the 1-tree protocol, we can also consider the case where the use of the second corner features can be varied and the order which corresponds to the best discriminating power is chosen. Despite its intrinsic simplicity, 1-tree protocol is prone to error since such an increase in the depth of the tree will inevitably increase the chance of committing error.

There is another tree architecture using the idea of entropy as given in the next section.

§4.7 BUILDING DECISION TREE BY ENTROPY REDUCTION METHOD

The 1-tree protocol discussed in previous section can be further improved if the best corner feature is used at every internal nodes of the tree instead of just allowing the use of one particular feature successively at some levels of the tree classifier. How can we quantify the "best" and how can we select the "best"? The question can be answered by defining the concept of the Shannon's entropy measure.

SHANNON'S ENTROPY MEASURE

Shannon's entropy is basically a measurement of the information level for a particular group of objects within a cluster. If a certain cluster consists of n objects where each object has a probability of p of being residing in the cluster, then the Shannon's entropy, denoted by E, is defined as

$$E = \sum_{i=1}^{n} -p_i \log_2 p_i .$$

The performance of a classifier can then be measured in terms of the difference between the entropy level before and after the classification process. It is usually referred as the information gain by the classifier. In symbol, the difference in entropy is

$$\Delta E \equiv E_{new} - E_{original} \ .$$

What does the value of the entropy tell us? Is a large value or a small value favourable? To answer the above questions,

we will consider an example. In a scenerio where a classifier classifies n distinct objects into only one cluster.

Then $$p_i = \frac{1}{n} \qquad \forall\ i=1,2,\ldots,n\ .$$

Therefore $$\text{Entropy} = \sum_{i=1}^{n} -p_i \log_2 p_i$$

$$= \sum_{i=1}^{n} \frac{1}{n} \log_2 n$$

$$= n \cdot \frac{1}{n} \log_2 n$$

$$= \log_2 n\ . \qquad (4\text{-}1)$$

In another scenerio where a classifier can classify n distinct objects into k clusters with n objects residing in each of them.

Then $$\sum_{i=1}^{k} n_i = n.$$

For the ith cluster, since there are n distinct objects and p_i for each object will be $\frac{1}{n_i}$. Therefore the corresponding entropy by (4-1) will be $E_i = \log_2 n_i$.

Hence, the total entropy, being weighted by the corresponding fractions of population shared, is

$$E_2 = \sum_{i=1}^{k} \frac{n_i}{n} \log_2 n_i. \qquad (4\text{-}2)$$

Obviously, $E_2 = \sum_{i=1}^{k} \frac{n_i}{n} \log_2 n_i$

$$= \frac{1}{n} \sum_{i=1}^{k} \left(\log_2 n_i^{n_i} \right)$$

$$= \frac{1}{n} \log_2 \left(n_1^{n_1} \, n_2^{n_2} \, \ldots \, n_k^{n_k} \right)$$

$$\leq \frac{1}{n} \log_2 \left(n^{n_1} \, n^{n_2} \, \ldots \, n^{n_k} \right)$$

since $n_i < n \quad \forall i$

$$= \log_2 \left(n^{n_1+n_2+ \ldots +n_k} \right)^{\frac{1}{n}}$$

$$= \log_2 \left(n^n \right)^{\frac{1}{n}}$$

$$= \log_2 n$$

$$= E_1$$

i.e. $E_2 \leq E_1$

The final scenerio shows n distinct objects being classified successfully to n distinct clusters with only one object residing in each of them.

Then $k = n$ and $n_i = 1 \quad \forall\, i=1,2, \ldots, n$.

By (4-2), the entropy $E_3 = \sum_{i=1}^{k} \frac{1}{n} \log_2 1 = 0$.

Hence, $0 = E_3 \leq E_2 \leq E_1 = \log_2 n$.

The above inequality shows that the smaller the value of the

entropy, the higher the information level the cluster possesses. It also reveals that the upper bound and the lower bound for the entropy measure are $\log_2 n$ and 0 respectively.

As a result of the above discussion, ΔE should be chosen as negative as possible so that the classifier is favourable.

TREE CLASSIFIER WITH ENTROPY REDUCTION

We want to use the idea of entropy reduction to improve our 1-tree protocol. As we have already mentioned, at each internal node of the tree we are free to choose among the three corner features any one particular feature. Decision is to be made so that there is the largest increase in the information gained. With the concept of entropy at hand, this actually refers to a classifier which gives the most negative value of ΔE.

In the current situation, characters are to be classified according to the three corner codes. Since all these three corner codes may be used at every step, three different classifications are possible. Suppose the first corner feature is selected, the entropy gain is reflected by the value of ΔE based on the classification given by the first corner feature. As a result, ΔE may not reflect the actual information gain very well since the original entropies, being calculated for the three different classifications

based on the three different corner features, are all different. In view of this, the entropy, which is the actual information level of a particular scenerio, will be adopted. The smaller the value the entropy, the higher is the information level. All three corner features will be tried at each internal node and the resulting entropy values are compared. The smallest among these three entropy values will be chosen and based on such corner feature the group of objects will be classified into smaller subgroups. The entire algorithm can be formulated as follows.

Since it takes a long time to train up this classifier, only 500 commonly used chinese characters are randomly chosen for training so that we can arrive at meaningful results at a relatively short period of time before we can proceed further. As a result, a depth of seven levels is assumed in our 1-tree protocol.

Repeat for tree level i = 0, 1, 2, ..., 6
Repeat for every internal node n at level i
1. Repeat for corner value j = 1, 2, 3
   1.1 Cluster the node n using the jth corner feature
   1.2 Calculate the entropy of the clustering result
2 Choose the corner feature c which corresponds to the smallest entropy values.
3 Repeat the classification process using the cth corner feature.

A.5 THEORIES ON STATISTICAL DISCRIMINANT ANALYSIS

I. The General Method of Classification

Consider the problem of classifying an observation (vector) x into one of k groups (or populations) $\Pi_1, \Pi_2, \ldots, \Pi_k$ where Π_i is characterized by a probability density function $f_i(x)$. Suppose further that the observation has a prior probability p_i of coming from Π_i, where $\sum_{i=1}^{k} p_i = 1$, and that the cost associated with classifying it into Π_i when it has actually come from Π_j is c_{ij}.

Within this framework, the conditional probability of classifying x to Π_i given the observation x is thus

$$\frac{p_i f_i(x)}{\sum_{i=1}^{k} p_i f_i(x)} \qquad i = 1, 2, \ldots, k$$

and hence the expected costs of misclassification are

$$\sum_{i\neq 1} p_1 f_1(x) c_{i1} \Big/ \sum_{i=1}^{k} p_i f_i(x), \ldots, \sum_{i\neq k} p_k f_k(x) c_{ik} \Big/ \sum_{i=1}^{k} p_i f_i(x)$$

where the term $\sum_{i\neq j} p_j f_j(x) c_{ij} \Big/ \sum_{i=1}^{k} p_i f_i(x)$ is the expected cost when misclassifying rule chosen should be one which minimizes the expected cost of misclassification. Thus, we assign observation x to Π_i if $\sum_{r\neq i} p_r f_r(x) c_{ir} \Big/ \sum_{i=1}^{k} p_i f_i(x)$ is minimum, i.e. if $\sum_{r\neq i} p_r f_r(x) c_{ir}$ is minimum. This is

equivalent to assigning observation x to Π_i if

$$\sum_{r=1}^{k} p_r f_r(x) c_{ir} < \sum_{r=1}^{k} p_r f_r(x) c_{jr}$$

$$\forall j = 1, 2, \ldots, k;\ j \neq i \qquad \text{(A5.1)}$$

since $c_{jj} = 0 \quad \forall j = 1, 2, \ldots, k.$

In the situation where the costs of misclassification are all equal, this rule reduced to assigning x to Π_i if

$$p_i f_i(x) = \max_{j=1,2,..,k} p_j f_j(x) \qquad \text{(A5.2)}$$

Assignment rules (A5.1) and (A5.2) have been derived by considering the discriminant analysis problem from a decision theoretic viewpoint. Viewing it form a purely probabilistic viewpoint instead, the optimal rule is to assign x to that group Π_i for which the posterior probability is the greatest. Now, using the Bayes theorem, the posterior probability rule is also (A5.2). So when the costs of misclassification are all equal, the optimal decision theoretic and probabilistic classification rule are equivalent.

In practice, the probability density function $f_i(x)$, i = 1, 2, ..., k are seldom known. Usually one assumes that they have some particular parametric form (e.g. multivariate normal distribution) which depends on some unknown parameters. Usually, random samples consisting of observations known to have come from each specific one of the k populations are used to construct sample based

classification rules corresponding to (A5.1) and (A5.2) above.

II. Classification using Distance Measure

We are now going to devise another set of classification rules and will see later, under certain circumstances, these rules and our previous ones are actually equivalent. For simplicity, let's treat the case when k=2 first and assume also that the underlying probability distributions of the two populations are multivariate normal with common covariance matrices. As a result, we are in the situation where we have an individual with observation vector $x = (x_1, x_2, \ldots, x_p)'$, and we wish to classify it into Π_1: $N_p(\mu_1, \Sigma)$ or into Π_2: $N_p(\mu_2, \Sigma)$ on the basis of x, where $\mu_i = (\mu_{i1}, \mu_{i2}, \ldots, \mu_{ip})'$, for i=1, 2.

In order to develop the classification theory, we first assume that the parameters μ_1, μ_2 and Σ are known. Here Σ is a p x p symmetric matrix represented as $\Sigma = (\sigma_{ij})$, i,j = 1,2,...,p. Intuitively, it seems reasonable to find a linear combination of the observations, called a discriminant function, given by

$$y = a_1x_1 + a_2x_2 + \ldots + a_px_p = a'x \qquad \text{(A5.3)}$$

where a_i's are some constants and to classify x into Π_1 if $y \geq c$ and into Π_2 if $y < c$ where c is another constant. The

problem then reduces to determining the values of a_i's and c which minimize the probabilities of making an incorrect classification.

If x is from Π_1, then y will be univariate normal with mean $\mu_1^* = \sum_{j=1}^{p} a_j \mu_{ij} = a'\mu_1$ and variance $v^2 = \sum_{i=1}^{p}\sum_{j=1}^{p} a_i \sigma_{ij} a_j = a'\Sigma\, a$. Similarly if x is from Π_2, then y will be univariate normal with mean $\mu_2^* = a'\mu_2$ and the same variance v^2. An intuitive criterion for choosing a is that to separate μ_1^* as far from μ_2^* as possible, relative to v^2. To achieve this, we define the Mahalanobis distance between μ_1 and μ_2 to be

$$\Delta^2(\mu_1, \mu_2) = \frac{(\mu_1^*, \mu_2^*)^2}{v^2}$$

and this quantity is supposed to be a measure of the "distance" between the two populations. Thus, we are going to find a so that in this y-scale where $y = a'x$, Δ^2 will be maximized.

$$\Delta^2(\mu_1, \mu_2) = \frac{(\mu_1^*, \mu_2^*)^2}{v^2} = \frac{[a'(\mu_1, \mu_2)]^2}{a'\Sigma\, a}$$

Using the Cauchy-Schwartz inequality, we have

$$\frac{a'(\mu_1, \mu_2)}{(a'\Sigma\, a)^{1/2}} \leq [(\mu_1 - \mu_2)'\, \Sigma^{-1}\, (\mu_1 - \mu_2)]^{1/2}$$

and equality holds if and only if $a \propto \Sigma^{-1}(\mu_1 - \mu_2)$

i.e. $\Delta^2 < (\mu_1 - \mu_2)'\Sigma^{-1}(\mu_1 - \mu_2)$.

Therefore, the maximum of Δ^2 is attained when $a = \Sigma^{-1}(\mu_1-\mu_2)$ and in such case,

$$\Delta^2(\mu_1, \mu_2) = (\mu_1 - \mu_2)'\Sigma^{-1}(\mu_1 - \mu_2) \qquad \text{(A5.4)}$$

Once the a_i's have been found, evaluation of (A5.3) for an individual whose measurements are $x_1, \ldots, x_p$ yields the discriminant score y for that individual.

To determine the constant c, we examine the following figure

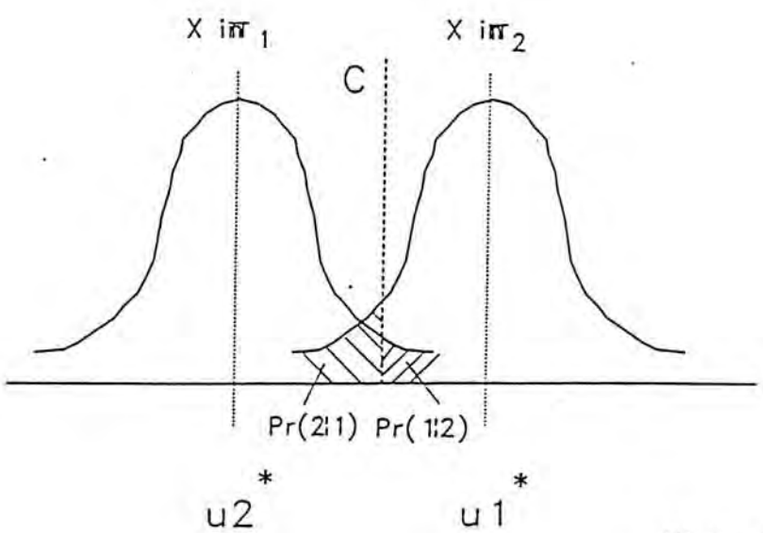


Figure A

which shows the two distributions of y along with an arbitrary constant c. If x is from Π_2 but $y \geq c$, then we would classify x into Π_1, thus committing an error. The probability Pr(1|2) of making this error is shown in the figure. Similarly Pr(2|1) is also shown. Intuitively, we would like to find c such that the sum of these probabilities Pr(1|2)+Pr(2|1) is minimized. In this simple case, it is obvious that this will be achieved by choosing c half way between the two means, that is,

$$\begin{aligned} c &= \frac{1}{2}(\mu_1^* - \mu_2^*) \\ &= \frac{1}{2}(a'\mu_1^* - a'\mu_2^*) \\ &= \frac{1}{2}\Sigma^{-1}(\mu_1 - \mu_2)(\mu_1 + \mu_2) \end{aligned}$$

Thus the empirical rule of classification is

Assign x to Π_1 if

$$\Sigma^{-1}(\mu_1 - \mu_2)x > \frac{1}{2}\Sigma^{-1}(\mu_1 - \mu_2)(\mu_1 + \mu_2)$$

i.e $(\mu_1-\mu_2)'\Sigma^{-1}x > \frac{1}{2}(\mu_1-\mu_2)'\Sigma^{-1}(\mu_1+\mu_2)$ (A5.5)

or equivalently (after some strict forward manipulation),

if $(\mu-x)'\Sigma^{-1}(\mu-x) > (\mu-x)'\Sigma^{-1}(\mu-x)$ (A5.6)

The last assignment rule simply says that we will classify x to Π_1 if the Mahalanobis distance between μ_1 and x is smaller.

To extend this approach to the case when $k > 2$, it should sound obvious then that the classification rule is to assign x to Π_i if the Mahalanobis distance between μ_i and x is the minimum among all the others, that is, if

$$\Delta^2(\mu_i, x) < \Delta^2(\mu_j, x) \qquad \forall j = 1,\ldots,p; \quad j \neq i$$

$$\text{i.e.} \quad (\mu_i - x)'\Sigma^{-1}(\mu_i - x) > (\mu_j - x)'\Sigma^{-1}(\mu_j - x)$$

$$\forall j = 1,\ldots,p;\ j \neq i \qquad \text{(A5.7)}$$

or equivalently if

$$(\mu_i - \mu_j)'\Sigma^{-1}x > \tfrac{1}{2}(\mu_i - \mu_j)'\Sigma^{-1}(\mu_i + \mu_j) \quad \forall j \neq i \quad \text{(A5.8)}$$

Notice that

$$(\mu_i - x)'\Sigma^{-1}(\mu_i - x) < (\mu_j - x)'\Sigma^{-1}(\mu_j - x)$$

$$\Leftrightarrow \quad \mu_i'\Sigma^{-1}\mu_i + x'\Sigma^{-1}x - \mu_i'\Sigma^{-1}x - x'\Sigma^{-1}\mu_i > \mu_j'\Sigma^{-1}\mu_j + x'\Sigma^{-1}x - \mu_j'\Sigma^{-1}x - x'\Sigma^{-1}\mu_j$$

$$\Leftrightarrow \quad \mu_i'\Sigma^{-1}\mu_i - 2x'\Sigma^{-1}\mu_i > \mu_j'\Sigma^{-1}\mu_j - 2x'\Sigma^{-1}\mu_j$$

$$\Leftrightarrow \quad \tfrac{1}{2}\mu_i'\Sigma^{-1}\mu_i - x'\Sigma^{-1}\mu_i > \tfrac{1}{2}\mu_j'\Sigma^{-1}\mu_j - x'\Sigma^{-1}\mu_j$$

Hence a generalization of the discriminant function can be taken as

$$y = a_i'x + c_i, \quad i = 1,\ldots,k \qquad \text{(A5.9)}$$

$$\text{where} \quad a_i = \Sigma^{-1}\mu_i \qquad \text{(A5.10)}$$

$$\text{and} \quad c_i = -\tfrac{1}{2}\mu_i'\Sigma^{-1}\mu_i$$

The equivalent classification rule in terms of the discriminant function will then be:

Assign x to Π_i if

$$y_i > y_j \quad \forall\, j \neq i \quad \Leftrightarrow \quad y_i - y_j > 0 \quad \forall\, j \neq i \qquad \text{(A5.11)}$$

i.e.
$$y_i = \max_i y_i \qquad \text{(A5.12)}$$

We have assumed a common covariance matrices for all the groups. However if the group covariance matrices are not equal and suppose these matrices be denoted as Σ_i. The Mahalanobis distance can be modified as $\Delta^2(\mu_i, x) = (\mu_i - x)' \Sigma_i (\mu_i - x)$. Then the quadratic term $x' \Sigma^{-1} x$ cannot be canceled in the above calculation and as a result, a quadratic discriminant function will come out.

The foregoing discussion based on the discriminating criterion is really a special case of that treated in I. Under the framework developed in I, if we assume the costs of misclassification are all equal, the classification procedure of (A5.2) gives assigning x to Π_i if

$$p_i f_i(x) > p_j f_j(x) \qquad \forall j \neq i.$$

If we further assume multivariate normal distributions with common covariance matrix Σ and equal prior, that is,

$$f_i(x) = \left(\frac{1}{2\pi} \right)^{p/2} |\Sigma|^{-1/2} \exp\left(-\frac{1}{2}(x-\mu_i)' \Sigma^{-1} (x-\mu_i) \right) \quad \forall\, i$$

and $p_i = p_j \quad \forall\, i,j$ then (A5.2) can be further reduced to

$$f_i(x) > f_j(x) \qquad \forall\, j \neq i$$

$$\Leftrightarrow \left(\frac{1}{2\pi} \right)^{p/2} |\Sigma|^{-1/2} \exp\left[-\frac{1}{2}(x-\mu_i)'\Sigma^{-1}(x-\mu_i) \right]$$
$$> \left(\frac{1}{2\pi} \right)^{p/2} |\Sigma|^{-1/2} \exp\left[-\frac{1}{2}(x-\mu_j)'\Sigma^{-1}(x-\mu_j) \right] \quad \forall j \neq i$$

$$\Leftrightarrow (x - \mu_i)'\Sigma^{-1}(x - \mu_i) > (x - \mu_j)'\Sigma^{-1}(x - \mu_j) \quad \forall j \neq i$$

$$\Leftrightarrow (\mu_i - x)'\Sigma^{-1}(\mu_i - x) > (\mu_j - x)'\Sigma^{-1}(\mu_j - x) \quad \forall j \neq i$$

which is exactly the same as (A5.2). Thus our previous claim is justified.

When our assumption of equal prior probability is not true, the classification rule based on (A5.2) and the normality assumption will lead to

$$P_i \left(\frac{1}{2\pi} \right)^{p/2} |\Sigma|^{-1/2} \exp\left[-\frac{1}{2}(x-\mu_i)'\Sigma^{-1}(x-\mu_i) \right]$$
$$> P_j \left(\frac{1}{2\pi} \right)^{p/2} |\Sigma|^{-1/2} \exp\left[-\frac{1}{2}(x-\mu_j)'\Sigma^{-1}(x-\mu_j) \right]$$
$$\forall j \neq i$$

$$\Leftrightarrow P_i \exp\left[-\frac{1}{2}(x-\mu_i)'\Sigma^{-1}(x-\mu_i) \right]$$
$$> P_j \exp\left[-\frac{1}{2}(x-\mu_j)'\Sigma^{-1}(x-\mu_j) \right] \quad \forall j \neq i$$

$$\Leftrightarrow \ln p_i - \frac{1}{2}(x - \mu_i)'\Sigma^{-1}(x - \mu_i)$$
$$> \ln p_j - \frac{1}{2}(x - \mu_j)'\Sigma^{-1}(x - \mu_j) \quad \forall j \neq i \qquad \text{(A5.13)}$$

In such case, an extra term $\ln p_i$ is added and in the terminology of our generalized discriminant function, only the terms c_i is changed. The new c_i is increased by the term $\ln p_i$

$$\text{i.e.} \quad c_i = \ln p_i - \frac{1}{2}\mu_i'\Sigma^{-1}\mu_i \qquad \text{(A5.14)}$$

A.6 PASSAGE USED IN TESTING THE PERFORMANCE OF THE CLASSIFIER WITH CONTEXT CONSIDERATION

能冊之後
功手於新
入附來加
進出印改
又列號式
程一位置
時星態之
之慧狀列
刷用頁一
印文分第　數
稿本表時置　頁
完將報出位置之
冊可察印之位表　數
手者觀頁列之報數字
作用上該一列印頁文
操使幕在第一所之英
體明螢停時第機束中數
軟說由會出時表結之字
本中前標印出列及中文
號文之游頁印由始文英
一本表頁之頁際起出中
星在列一定一實定算之
慧處於上指下指指計中
用之可至至是可可窗　在在在均在在均
使增號跳跳跳頁表號視整所所列所所所列
您新一標標標的報一本調標標標有標標有
謝將星游游處印星示章游游游所游游所
謝特慧令令此列慧顯文將將若下將若下

以標游內間區則內格一間　央一格內則區間內游標以
中移區格移區格
窗平之一平之一
視水定移水定移
至左設平右設平
移向在水向在水
列列左列列右
之之向之之向
在在在均在在均

A.7 A PARTIAL LIST OF SEMANTICALLY RELATED CHINESE CHARACTERS

一丁乙七乃九了二人入八刀刁力匕十卜又三下丈上丫凡久也乞亡兀千叉口土士夕大女子孑寸小山川工已干弓才丑不中丰丹之予井互五^C

切段次行些生致律半打定雙旦對群份陣再同週向直場筆般回批串封面號頁列格
等種
等種存
折成樸月年百千萬點分情雄夕堵美發
是父兄
折成樸月年點龍族如
斷然解得結事了案債
段樸月點手專等次回種林信水
員才口品性緣心生際情格物類士文名倫群種證選質手力人間工們事家
門境股流口迷學選場時超黨圍賬會庫獄骨冬院閣籍殮
折成卦字樸月德面分堵里股
子傷口槍俎鋒痕刃
難民悍
氣量求行爭圖學
首
分成足全個樸月信
卦
說是有
段樸軍民專思總角生代重更信義
班雨車來面列等棋限部落方課屆場臺聘游跪風霜雪人身坡酒榻田文筆
夫市人量
市車進策漲班學課級任文等游品山場陣好方面糊算空街菜樸昇衣前書
頭
是事人
仰違留候別遠
許就有要是好對
討求丐食憐
命魂國
贏
秋古里載萬金歲年分億
子才路腰
才地吃音氣訣渴袋試哨號實述語舌授德紅腹琴供
地產著木銀質壞法匪城
氣兵官林
陽照
概家量約眾致地同意網局陸腦學腿器作權我成事宗人半海師都寒麥雪
人孩生兒子性士流婿權神皇傭郎色工中紅巫
孫女弟彈夜
然
步土金斷腸
心氣偷學康費路鳥溪鎮節組名事型別時生於島徑道調民腦腸麥米說
水上歌地谷坡路莊峰林川脈洞洪頂巔嶽腰罌河岳嵐崗野崖頭神胞茶蘿
菜資
作具人夫業商程會專資廠整兵匠藝讀錢地科寮頭
經知定故婚屬往有久是還
涉休戈預擾支撓貝政與犯
箭形弦矢手
氣華能智幹要算對會來好有是行貌子思情俊學藝器識
角旦
久論但僅符如顧妨幸朽凡甘平經尚測法肯料准得許堪可快仁愧絕敢曾
間心途止國華央外文庸立旬性年醫和原興學午樞山風毒校將飯秋正用英
采姿韻
青鳳田麥
間際類餘處外內中前後一二道於時久多上下左右列
以
水
相惠信選諒保動異
折助換金專穀臟彩代經權院行音嶽倫常更官帝嶺堵股指戒峰福華原百律

A.8 AN EXAMPLE OF MISCLASSIFICATION TABLE

SAS 17:50 Saturday, December 7, 1991 1

Discriminant Analysis

13093 Observations	13092 DF Total
8 Variables	12993 DF Within Classes
100 Classes	99 DF Between Classes

SAS 17:50 Saturday, December 7,

Discriminant Analysis

Class Level Information

GROUP	Frequency	Weight	Proportion	Probab
0	33	33.0000	0.002520	0.0
1	117	117.0000	0.008936	0.0
2	192	192.0000	0.014664	0.0
3	152	152.0000	0.011609	0.0
4	26	26.0000	0.001986	0.0
5	19	19.0000	0.001451	0.0
6	273	273.0000	0.020851	0.0
7	36	36.0000	0.002750	0.0
8	70	70.0000	0.005346	0.0
9	156	156.0000	0.011915	0.0
10	99	99.0000	0.007561	0.0
11	39	39.0000	0.002979	0.0

SAS 17:50 Saturday, December 7,

Discriminant Analysis

Class Level Information

GROUP	Frequency	Weight	Proportion	Probab
12	76	76.0000	0.005805	0.0
13	51	51.0000	0.003895	0.0
14	236	236.0000	0.018025	0.0
15	151	151.0000	0.011533	0.0
16	184	184.0000	0.014053	0.0
17	33	33.0000	0.002520	0.0
18	106	106.0000	0.008096	0.0
19	96	96.0000	0.007332	0.0
20	57	57.0000	0.004353	0.0
21	129	129.0000	0.009853	0.0
22	443	443.0000	0.033835	0.0
23	82	82.0000	0.006263	0.0

SAS 17:50 Saturday, December 7,

Discriminant Analysis

Class Level Information

GROUP	Frequency	Weight	Proportion	Probab
84	80	80.0000	0.006110	0.0
85	63	63.0000	0.004812	0.0
86	12	12.0000	0.000917	0.0
87	71	71.0000	0.005423	0.0
88	38	38.0000	0.002902	0.0
89	46	46.0000	0.003513	0.0
90	39	39.0000	0.002979	0.0
91	10	10.0000	0.000764	0.0
92	5	5.0000	0.000382	0.0
93	282	282.0000	0.021538	0.0
94	72	72.0000	0.005499	0.0
95	17	17.0000	0.001298	0.0

SAS 17:50 Saturday, December 7,

Discriminant Analysis

Class Level Information

GROUP	Frequency	Weight	Proportion	Probab
96	82	82.0000	0.006263	0.0
97	5	5.0000	0.000382	0.0
98	236	236.0000	0.018025	0.0
99	24	24.0000	0.001833	0.0

SAS 17:50 Saturday, December 7,

Discriminant Analysis Pooled Covariance Matrix Informatio

Covariance Matrix Rank	Natural Log of the Determinant of the Covariance Matrix
8	-26.321635

SAS 17:50 Saturday, December 7,

Discriminant Analysis

Pairwise Generalized Squared Distances Between Groups

$$D^2(i|j) = (\bar{X}_i - \bar{X}_j)' \, COV^{-1} \, (\bar{X}_i - \bar{X}_j)$$

Generalized Squared Distance to GROUP

From GROUP	0	1	2	3
0	0	15.86808	40.92412	39.86166
1	15.86808	0	19.62421	12.63673
2	40.92412	19.62421	0	11.27509
3	39.86166	12.63673	11.27509	0
4	17.58963	3.24730	10.53914	9.52195
5	27.90833	7.16681	11.71902	7.88185
6	16.87428	3.85443	16.66425	11.07530

Discriminant Analysis

Pairwise Generalized Squared Distances Between Groups

Generalized Squared Distance to GROUP

From GROUP	0	1	2	3
7	23.06894	6.65154	8.95407	8.95160
8	18.36506	4.23422	13.57648	11.39801
9	24.12825	11.11577	17.40203	14.25962
10	25.64130	1.66880	18.08759	9.12688
11	26.74619	3.74048	13.46667	8.91385
12	40.34600	12.65124	14.97727	4.20880
13	28.18604	4.10865	15.30706	3.81119
14	23.55590	4.35870	15.53312	3.65341
15	25.09215	11.13224	21.40302	12.08586
16	27.14399	4.71385	15.91768	3.29132
17	21.37768	10.90931	7.12406	16.21537

Discriminant Analysis

Pairwise Generalized Squared Distances Between Groups

Generalized Squared Distance to GROUP

From GROUP	0	1	2	3
18	34.26650	6.49275	19.55515	4.91344
19	23.55241	7.91028	15.30755	10.91255
20	28.47814	12.85073	13.30639	11.15935
21	14.89187	13.90955	9.83616	17.25501
22	27.18039	16.48131	5.89629	20.25172
23	24.42650	10.28278	11.29767	11.61187
24	20.08855	9.39867	7.81337	14.47392
25	23.40343	12.89247	9.76770	12.80159
26	35.77803	26.58990	7.81313	25.86069
27	34.09364	18.52761	2.64013	14.51358
28	19.27517	10.49412	8.42766	16.00337

A.9 LISTING OF THE PROGRAM

```
*
*   Program for the improved multistage tree classifier
*   with context consideration
*    . 3-tree protocol
*    . trained with 5400 characters
*    . with overlapping
*    . code file stored in testdata
*    . character bit pattern stored in c:\et\stdfont.24
*    . input passage stored in c:\sin\data\chfont.dat
*    . only the big 5 codes are stored in input passage
*    . the words which are semantically related are stored
*      in file c:\sin\data\word
*
*
*   To invoke the program, type
*      chdis c:\sin\data\chfont.dat
*
/
include <stdio.h>
include <dos.h>
include <alloc.h>
include <string.h>
include <stdlib.h>
include <graphics.h>
include <io.h>
include <fcntl.h>
include <math.h>
include <float.h>
include <time.h>

define  M  24
define  N  3
define  MAXRECS 55833

ime_t  *t1,*t2;
ouble  diff=0;
nt   recog,n;
har  *lcx;
har  *lcy;
nt   choice,y1,y2;
ong  pre_char;   /* previously recognized character */
ong  add;
ong  pos;
har  id[10];
ong  coun,lc_count;
nt   cat_1,cat_2,cat_3,cat_4;

 int       x,n,fp;
 int      Xmin;
 int      Xmax;
 int       Xpt;
 int       Xwidth;
 int       Xcenpt;
 int       Ymin;
 int       Ymax;
 int       Ypt;
 int       Ywidth;
 int       Ycenpt;
 float    Xcen;
```

```
float     Xoff;
float     Ycen;
float     Yoff;
int       Yfeapt[4];
int       Xfeapt[4];
float     profile[24];
unsigned char  buf[N*M];
unsigned long  bit[M];
unsigned long  k;

int     node;
int     co[3];
char    *code1;
long    code;
float   result;
FILE    *fp2,*fp3,*fin,*fp1,*fcode,*fopen();

/* discriminant functions */

float  disc_fcn[45][9]={
   {0.259918,0.621660,-0.770455,0.258583,0.994533,-0.370675,-0.853918,2.393407,-1.525431},
   {-0.381643,0.884263,1.696303,-0.510696,-0.522009,0.706840,-0.605000,0.562136,0.368424},
   {0.315879,1.177575,0.656201,-1.491476,1.217296,-0.039149,-0.175298,-0.574729,-1.040982},
   {-0.353333,-0.268644,0.338642,-0.406490,-1.517231,-0.124308,0.693505,1.508993,1.745559},
   {-0.671441,0.646147,1.347479,0.605567,0.421958,-0.222858,-0.001174,-0.505027,0.015337},
   {-1.008538,-0.664718,-0.932718,0.331255,-0.536410,-0.344057,-0.210009,2.403167,2.473747},
   {-1.374422,-0.773219,-1.275659,0.490463,-0.583687,-0.556719,-0.307605,1.434989,2.659484},
   {-0.268457,0.586162,0.913754,0.426680,1.027468,0.388865,0.628954,-0.488123,-1.526237},
   {0.496920,0.056958,0.612463,-1.384370,0.250985,-0.822307,2.185022,-1.193536,-1.129769},
   {1.473582,-1.503900,-0.078670,-1.137006,-0.969084,-0.871575,-0.094717,0.640670,0.139311},
   {0.199512,0.057929,-0.072065,-0.147075,0.449750,0.522399,-0.760808,-0.690857,-0.529508},
   {-0.272413,1.423927,0.783386,2.132299,-0.236750,0.567282,0.057778,0.467315,-1.309404},
   {0.311660,-0.912036,1.163019,-1.730857,1.351861,-0.441743,1.094023,0.900912,-2.145466},
   {1.192975,1.036124,0.721519,1.531767,0.449841,1.463854,-0.282628,-0.329865,-2.663348},
   {0.118644,-0.990489,-1.962845,-0.114584,0.036559,-0.481495,0.679746,1.805248,0.297864},
   {-0.351202,0.424194,0.492202,0.348260,-1.063629,-0.381929,1.349759,0.007510,0.710211},
   {-1.269880,0.652559,0.122311,0.289565,0.265075,1.038581,0.968956,1.412551,0.148753},
   {-0.594417,0.690181,0.987203,0.936801,-0.028701,0.467359,0.422616,-0.095399,-0.534935},
   {1.663410,-1.208385,-0.538888,-0.184842,-0.638519,0.123112,-0.074472,-0.851112,-0.249387},
   {-0.217317,-0.034913,-0.104038,-0.533322,2.984313,1.114763,1.312663,-5.038081,-0.628484},
   {-0.077140,-0.000843,-0.453704,-0.186843,0.615515,0.863985,5.162472,-2.568067,-0.983720},
   {-0.451112,0.650777,-0.650523,1.989002,-0.666488,0.267559,1.178607,-0.482039,1.153759},
   {-1.328029,2.494712,0.581879,0.284973,0.140345,-0.378698,0.321152,-0.634000,0.864982},
   {0.720861,-0.889762,-0.328747,0.214968,1.507224,0.794815,-0.345885,-2.011523,-0.509573},
   {-0.072493,0.694930,0.444604,-0.565087,-1.485905,3.948742,-0.934063,5.600830,0.051008},
   {-1.469259,1.137459,-0.147181,-0.249946,3.194611,3.577985,2.072181,2.827119,-0.207754},
   {0.754033,-0.596174,0.410781,-0.240091,-0.409267,-1.452092,4.983140,-5.328596,0.274952},
   {0.005162,-0.186474,-0.084109,-0.156268,-0.516985,3.953562,12.674566,-4.120590,0.107159},
   {0.217346,-0.050913,-1.181020,-0.397353,-0.699470,0.742753,0.956275,-2.399568,1.155229},
   {-1.195769,2.686259,1.819948,0.751346,0.323224,0.886600,-2.458093,-0.890145,2.797885},
   {0.799554,-0.457612,-0.191509,0.170058,-0.358146,-0.707604,-0.877927,-0.149762,-0.277449},
   {-0.856249,0.428454,0.150868,-0.193337,0.597869,1.193256,1.701772,-0.053726,0.304037},
   {-1.027854,0.358235,-0.108164,-0.219415,0.390514,0.911118,0.743459,0.129462,0.596922},
   {0.043969,0.276587,-0.413063,0.811957,0.245773,-0.795320,-0.619921,-0.098472,0.156250},
   {0.551382,-0.707475,0.319712,0.041579,-0.656943,-2.120445,-1.448657,0.260004,0.039185},
   {0.742407,-0.171074,0.623697,0.075668,0.006988,-1.096462,-0.312354,-0.155790,-0.446310},
   {0.200289,-0.222659,0.175490,-0.551110,0.392352,-0.832383,0.210715,-0.122989,-0.097941},
   {-0.674960,0.895881,0.016432,-0.056803,0.003973,0.851943,1.022395,0.407195,0.151322},
   {0.130038,-0.568141,0.176819,-0.842862,-0.765515,0.195519,1.058933,0.371929,-0.439006},
```

```
    {-0.055000,-0.295540,-0.784643,-0.080401,0.629070,2.934311,-0.526669,-0.222740,0.007183},
    {-0.594109,1.041155,-0.796324,-0.076152,0.964087,1.871607,1.052036,-0.429614,-0.231278},
    {-0.910578,0.600164,-0.237956,-0.271938,-0.035004,1.963163,0.702298,0.204623,-0.011660},
    {1.222852,-0.814026,-0.992097,0.345090,-0.662208,0.268078,-1.141079,0.030397,-0.434030},
    {-0.637045,0.774197,-0.050094,0.751595,-0.434984,2.389400,0.142221,0.424816,-0.017703},
    {-0.081660,0.076773,-0.361142,0.118932,-0.603035,0.996723,-0.875674,0.354863,0.323429}
  };

  long  begin,end;
  long  address,cc,c;
  int   find,c1,c2;
  int   front[10], rear[10];
  long  ex,first,p,q,kk;
  long  count,nn,m;
  long  low,high,mid,y;
  char  in_code[10];
  long  char_id;
  long  true_5;

  struct lcrec
   { long   lckey;
     int    begin;
     int    end;
    } huge *lc;

  long  huge *lcword;

long
num(char *x)

{
  int    flag,i;
  char   digit;
  int    y1,y2;
  long   id_1=0l;
  long   n,d;

  for (i=0;i<4;++i)
    {
       digit = *(x+i);
       switch(digit)
        {
          case 'a':  d=10; break;
          case 'b':  d=11; break;
          case 'c':  d=12; break;
          case 'd':  d=13; break;
          case 'e':  d=14; break;
          case 'f':  d=15; break;
          default:   d=atoi(&digit);
         }
       id_1 += (d<<(4*(3-i)));
    }

  true_5 = id_1;

    /*  calculate the position of the character in the file  */

  if (id_1 >= 0xC940)
    {
```

```
      id_1 -= 0x300;
      flag = 1;
    }
  n = id_1 - 0xA440 +1;
  y1 = n/256;
  y2 = n - 256 * y1;
  n = 157 * y1 + y2;
  if (y2 > 63)
    n -= 34;
  if (flag == 1)
    n +=63;

  return((3*24)*(long)(n-1));
 }

long
big_5(number)

long number;

{
  long  offset;
  long  c1;
  int   q,c2;

  offset = 0xA440;
  if (number > 5401)
    {
      offset = 0xC940;
      number -= 5401;
    }
  q = (number-1)/157;
  c2 = number - q*157;
  c1 = q*256 + offset - 1;
  if (c2 > 63)
    c2 += 34;
  return( c1 + c2 );
 }

/*  Binary seach Function  */

long
binary(long key)
{
  long  low,high,mid;
  long  char_id;
  char  in_code[10];
  long  c;

  low = 0;
  high = MAXRECS - 1;
  while (low <= high)
    {
      mid = (low+high)/2;
      fseek(fin,13*mid,0);
      fscanf(fin,"%lx %s",&char_id,&in_code);
      c = atol(in_code);
      if (key < c)
       high = mid-1 ;
```

```
      else if (key > c)
        low = mid+1;
      else
        return(mid);
     }
  return -1;
 }

/*  Binary seach Function for lclist */

long
bin(long max, long key, struct lcrec huge *tab)
{
  long  low,high,mid;

  low = 0;
  high = max - 1;
  while (low <= high)
    {
      mid = (low+high)/2;
      if (key < (tab+mid)->lckey)
        high = mid-1 ;
      else if (key > (tab+mid)->lckey)
        low = mid+1;
      else
        return(mid);
     }
  return -1;
 }

int
comp(int a, int b)
{
  if (a == b)
    return -1;
  else
    return 1;
 }

main(argc,argv)

int    argc;
char   *argv[];

{
  long   i,j,c,d;

  /*  read in the look-up table for semantically related characters */

    t1 = malloc(sizeof(time_t));
    t2 = malloc(sizeof(time_t));

    fp2 = fopen("c:\\sin\\data\\word.cwi","r");
    lcx = (char *)calloc(sizeof(char),3);
    lcy = (char *)calloc(sizeof(char),2);
    lc = (struct lcrec huge *)farcalloc(sizeof(*lc),3800);
    lcword = (long huge *)farcalloc(sizeof(long),20631);

    ccun = 0;
```

```
 lc_count = 0;
 while  ( (n=fread(lcx,2,1,fp2)) > 0 ) {
   y1 = *lcx + 256;
   y2 = *(lcx+1);
   if ( y2 < 0 )
    y2 += 256;
   (lc+coun)->lckey = (long)y1*16*16+y2;
   fread(lcy,1,1,fp2);   /*  space  */
   (lc+coun)->begin = lc_count;
   while  ( (n=fread(lcy,1,1,fp2)) > 0 ) {
    y1 = *lcy;
    if  ( y1 == 10 )    /* carriage return */
      break;
    else  {
      y1 += 256;
      fread(lcy,1,1,fp2);
      y2 = *lcy;
      if ( y2 < 0 )
        y2 += 256;
      lcword[lc_count] = (long)y1*16*16+y2;
      lc_count++;
     }
    }
   (lc+coun)->end = lc_count-1;
   coun++;
 }
 coun--;

/*  read in the look-up table for code conversion  */

fin = fopen("testdata","r");

fp=open("c:\\et\\stdfont.24", O_BINARY | O_RDONLY);

fp3 = fopen(argv[argc-1],"r");

cat_1 = cat_2 = cat_3 = cat_4 = 0;
count = 1;
while ( fscanf(fp3,"%s ",&id) != EOF )
{
   time(t1);

   pos = num(id);
   n = lseek(fp,pos,0);
   read(fp,buf,M*N);

   /*  Calculate the profile features of the inputted character  */

      for (y=0;y<M;y++)
      {
        bit[y]=0;
        for (c=0;c<N;c++)
              bit[y]=bit[y]+((long)buf[N*y+c]<<(8*(N-c-1)));
      }

      /*  Calculate the X, Y max. and min  */

      Xmin  =  23;
      Xmax  =  0;
```

```
Ymin  =  23;
Ymax  =  0;

for (y=0;y<M;y++)
{
      k=bit[y];
      for (x=8*N-1;x>=0;x--)
      {
        if (k & 1)
        {
             if (x>Xmax)
                Xmax = x;
             if (x<Xmin)
                Xmin = x;
             if (y>Ymax)
                Ymax = y;
             if (y<Ymin)
                Ymin = y;
        }
        k >>= 1;
      }
}

/*  First Corner Feature  */

/*  Calculate the feature extraction points on Y-axis  */

Ycen = (Ymax+Ymin)/2.0;
Yoff = (Ycen-Ymin)/3.0;
Xcen = (Xmax+Xmin)/2.0;
Xoff = (Xcen-Xmin)/3.0;
Xcenpt = (int)(Xcen+0.5);
Xwidth = Xcenpt-Xmin;
Ycenpt = (int)(Ycen+0.5);
Ywidth = Ycenpt-Ymin;

for (c=0;c<4;c++)
{
      Yfeapt[c]=0;
      Ypt=(int)(Ycen-c*Yoff+0.5);
      for (d=8*N-Xmin-1;d>8*N-Xcenpt-1;d--)
      {
         k=bit[Ypt]>>d;
         if ((k&1)!=1)
               ++Yfeapt[c];
         else d=-1;
      }
      profile[c]=(float)Yfeapt[c]/Xwidth;
}

/*  Calculate the feature extraction points on X-axis  */

for (c=0;c<4;c++)
{
      Xfeapt[c]=0;
      Xpt=(int)(Xcen-c*Xoff+0.5);
      for (d=Ymin;d<Ycenpt;d++)
      {
         k=bit[d]>>(8*N-Xpt-1);
```

```
        if ((k&1)!=1)
              ++Xfeapt[c];
        else d=Ycenpt+1;
      }
      profile[c+4]=(float)Xfeapt[c]/Ywidth;
}

/*  2nd Corner Features  */

/*  Calculate the feature extraction points on Y-axis  */

Ycen = (Ymax+Ymin)/2.0;
Yoff = (Ycen-Ymin)/3.0;
Xcen = (Xmax+Xmin)/2.0;
Xoff = (Xmax-Xcen)/3.0;
Xcenpt = (int)(Xcen+0.5);
Xwidth = Xmax-Xcenpt;
Ycenpt = (int)(Ycen+0.5);
Ywidth = Ycenpt-Ymin;

for (c=0;c<4;c++)
{
      Yfeapt[c]=0;
      Ypt=(int)(Ycen-c*Yoff+0.5);
      for (d=8*N-Xmax-1;d<8*N-Xcenpt-1;d++)
      {
         k=bit[Ypt]>>d;
         if ((k&1)!=1)
               ++Yfeapt[c];
         else d=8*N;
      }
      profile[8+c]=(float)Yfeapt[c]/Xwidth;
}

/*  Calculate the feature extraction points on X-axis  */

for (c=0;c<4;c++)
{
      Xfeapt[c]=0;
      Xpt=(int)(Xcen+c*Xoff+0.5);
      for (d=Ymin;d<Ycenpt;d++)
      {
         k=bit[d]>>(8*N-Xpt-1);
         if ((k&1)!=1)
               ++Xfeapt[c];
         else d=Ycenpt+1;
      }
      profile[c+12]=(float)Xfeapt[c]/Ywidth;

}

/*  3rd Corner Features  */

/*  Calculate the feature extraction points on Y-axis  */

Ycen = (Ymax+Ymin)/2.0;
Yoff = (Ymax-Ycen)/3.0;
Xcen = (Xmax+Xmin)/2.0;
Xoff = (Xcen-Xmin)/3.0;
```

```
    Xcenpt = (int)(Xcen+0.5);
    Xwidth = Xcenpt-Xmin;
    Ycenpt = (int)(Ycen+0.5);
    Ywidth = Ymax-Ycenpt;

    for (c=0;c<4;c++)
    {
          Yfeapt[c]=0;
          Ypt=(int)(Ycen+c*Yoff+0.5);
          for (d=8*N-Xmin-1;d>8*N-Xcenpt-1;d--)
          {
             k=bit[Ypt]>>d;
             if ((k&1)!=1)
                   ++Yfeapt[c];
             else d=-1;
          }
          profile[c+16]=(float)Yfeapt[c]/Xwidth;
    }

    /*  Calculate the feature extraction points on X-axis  */

    for (c=0;c<4;c++)
    {
          Xfeapt[c]=0;
          Xpt=(int)(Xcen-c*Xoff+0.5);
          for (d=Ymax;d>Ycenpt;d--)
          {
             k=bit[d]>>(8*N-Xpt-1);
             if ((k&1)!=1)
                   ++Xfeapt[c];
             else d=Ycenpt-1;
          }
          profile[c+20]=(float)Xfeapt[c]/Ywidth;
    }

/*  To calculate the code for the inputted character  */

code = 0;

for (c=0;c<3;++c)   /*  3 corners  */
 {
  node=1;   /*  the root node  */

  for (j=0;j<4;++j)      /* 4 levels in the tree classifier */
    {
     result = disc_fcn[c*15+node-1][0];    /* constant term */
     for (i=0;i<8;++i)
       result += disc_fcn[c*15+node-1][i+1] * profile[c*8+i];

     if ( result >= 0 )
       node = 2*node;         /*  left son  */
     else
       node = 2*node + 1;     /*  right son  */
    }

  co[c] = node - 16;
  code += co[c] * pow(100,2-c);

 }
```

```
cc=c=true_5;
c1=cc>>8;
c2=c-(c1<<8);
printf("%ld %c%c ",count,c1,c2);
printf("%ld\n",code);

address = binary(code);   /*  binary search  */
find = 1;
if (address == -1)
  {
    find = 0;

    /*  Back-Tracking  */

    printf("No find: back-tracking ... ");
    find = 0;
    i = 0;
    while  ( find == 0 && i<4 ) {
      c = 2;
      while ( find == 0 && c >= 0 ) {
        node = co[c] + 16 ;
        for (j=0;j<=i;++j)
          node = node/2;
        for (j=i;j>=0;--j)  {
          result = disc_fcn[c*15+node-1][0];   /* constant term */
          for (k=0;k<8;++k)
           result += disc_fcn[c*15+node-1][k+1] * profile[c*8+k];
          if ( comp(i,j)*result >= 0 )
           node = 2 * node;    /* left son */
          else
           node = 2*node + 1;   /* right son  */
         }
        code += (node-16-co[c])*pow(100,c);
        address = binary(code);
        if (address != -1)
          find = 1;
        --c;
       }
      ++i;
     }
 }

if (find == 0)
  printf("Still no find \n");
else
  {
    /*  output the big_5 code  */

    fseek(fin,13*address,0);
    fscanf(fin,"%lx %s",&char_id,&in_code);
    cc = atol(in_code);

    /* downward search */

    i=1;
    do {
     fseek(fin,13*(address+i),0);
     fscanf(fin,"%lx %s",&char_id,&in_code);
```

```
     c = atol(in_code);
     i++;
    } while ( c == cc && (address+i) <= MAXRECS );
    end = address+i;

    /* upward search */

    i=1;
    do {
     fseek(fin,13*(address-i),0);
     fscanf(fin,"%lx %s",&char_id,&in_code);
     c = atol(in_code);
     i++;
    } while ( c == cc && (address-i) >= 0 );
    begin = address-i;

    for (i=begin;i<=end;++i)  {
     fseek(fin,13*i,0);
     fscanf(fin,"%lx %s",&char_id,&in_code);
     cc=c=char_id;
     c1=cc>>8;
     c2=c-(c1<<8);
     printf("%d. ",i-begin+1);
     printf("%c%c ",c1,c2);
    }
 printf("\n");

recog = 0;
if ( begin == end )  { /*  only one single character  */
     pre_char = char_id;
     if ( char_id == true_5 )
       cat_1++;    /* unqiuely determined the character */
     else
       cat_3++;    /* incorrectly recognized the character */
    }
    else    /* get the semantically related words and make the decision */
     if ( count > 1 )   {
       add = bin(coun,pre_char,lc);
       if ( add != -1 )
         for ( j=(lc+add)->begin;j<=(lc+add)->end && recog==0;++j)
           for (i=begin;i<=end && recog==0;++i)  {
            fseek(fin,13*i,0);
            fscanf(fin,"%lx %s",&char_id,&in_code);
            if ( lcword[j] == char_id ) {
              recog = 1;
              pre_char = char_id;
             }
           }
        if ( recog == 1 )
          if ( pre_char == true_5 )
           cat_1++;
          else
           cat_3++;
       }
    if ( count == 1 || recog == 0 ) {
      printf("Please enter a number for your choice ==> ");
      scanf("%d",&choice);
      fseek(fin,13*(begin+choice-1),0);
      fscanf(fin,"%lx %s",&char_id,&in_code);
```

```
      pre_char = char_id;
      cat_2++;
    }

  }
 time(t2);
 diff += difftime(*t2,*t1);
 ++count;
}

count--;
printf("Percentage in category 1 = %f\n",(float)cat_1/count);
printf("Percentage in category 2 = %f\n",(float)cat_2/count);
printf("Percentage in category 3 = %f\n",(float)cat_3/count);
printf("Average processing time per character = %f\n",(float)diff/count);
farfree(lcword);
farfree(lc);
close(fp);
fclose(fin);
return 0;
```

REFERENCE

[1] Tappert, Suen & Walahara, "The State of the Art in On-Line Handwritting Recognition," IEEE Transaction on Pattern Analysis and Machine Intelligence, Vol. 12 No.8 August 1990.

[2] S.Rasoul Safavian and David Landgrede, "A Survey of Decision Tree Classifier Methodology," IEEE Transactions on Systems, man, and cybernetics Vol.21 No. 3 1991.

[3] R. Ahlsmede and I. Wegeru, Search Problems. New York: Wiley Interscience. 1987.

[4] P. Argentiero, R. Chin, and P. Beaudet, "An automated approach to the design of decision tree classifiers, "IEEE Trans. Pattern Analysis Machine Intelligent Vol. PAMI-4, pp. 51-57, 1982.

[5] L.A.Bartolucci, P.H.Swain, and C.Wu, "Selective radiant temperature mapping using a layered classifier," IEEE Trans. Geosci Electron., vol. GE-14, pp. 101-106, 1976.

[6] R.C.Casey and G.Nagy, "Decision tree design using a probablistic model," IEEE Trans. Information Theory, vol. IT-30, pp. 93-99, 1984.

[7] L Hyafil and R.L.Rivest, "Constructing Optimal Binary Decision Tree is NP-Complete," Inform. Processing Lett., Vol 5 No. 1 pp 15-17 1976.

[8] K.G. Beauchamp, "Walsh Functions and Their application," Academic Press, New York 1975

[9] Jun S. Hung & Ma-lung Chung, "Separating Similar Complex Chinese Characters by Walsh Transform," Pattern Recognition Vol 20 No. 4 1986.

[10] Shu Wenhao, Chi Guo-Wei, Zhao Ri-hua, "An accurate method for recognition of Printed Chinese characters," IEEE 1988.

[11] Michio Umeba, "Recognition of Multi-font Printed Chinese characters", IEEE 1982.

[12] Michael R. Anderberg, "Cluster Analysis for application, " 1973.

[13] Tou and Gonzalez, "Pattern Recognition Principles," Addison-Wesley Publish Co.

[14] Bryan F.J.Manly, "Multivariate Statistical Methods, A Primer," Chapman and Hall.

[15] Q.R.Wang & C.Y.Suen, "Analysis and Design of a Decision Tree Based on Entropy Reduction and its Application to large character set recognition," IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. PAMI-6, No. 4, July, 1984.

[16] Y.X.Gu, Q.R.Wang, and C.Y.Suen, "Application of a Multilayer Decision Tree in Computer Recognition of Chinese Characters," IEEE Trans. on Pattern Analysis and Machine Intelligence vol. PAMI-5, No. 1, January 1983.