DELFT UNIVERSITY OF TECHNOLOGY
Faculty of Electrical Engineering
Telecommunications and Traffic Control Systems Group

TITLE: Hidden Markov models applied to on-line handwritten character recognition and signature verification

Author: S.R. Veltman

Type: Thesis
Size: 150+XII
Date: May 4, 1992

Professor:	Prof. Dr. J.C. Arnbak Prof.
Mentor:	Dr. R. Prasad
Period:	November 1991 - May 1992

ABSTRACT:

Hidden Markov models are used to model the generation of handwritten, isolated characters. Models are trained on examples by optimization routines. Principally, optimization using the classical Lagrange multiplier method is possible, but in this work the so-called Baum-Welch optimization procedure is used. Then, given a model for each character in the vocabulary, unknown characters can be classified using maximum-likelihood classification. Experiments have been conducted, and an error rate of 6.6% over the English alphabet was achieved.

Some experiments on the application of HMMs to signature verification are presented. Results indicate that HMMs provide a solid basis for a HMM-based signature verification system.

Indexing terms:	Character Recognition, Signature Verification, Pattern Recognition

Summary

Recently, technology has become available that integrates a display and a writing tablet in a single unit, thus renewing the interest in machine interpretation of handwritten information. From the multimedia point of view, the user interface also becomes more important, because the integration of many services into one terminal makes the user more dependent on this terminal. In this work, two applications of machine interpretation of handwritten script are investigated: character recognition and signature verification. Hidden Markov models are examined, and applied to the matching problem in character recognition. An optimum set of models is determined, with which finally an error rate of 6.6% achieved over the English alphabet.

Some global concepts that play a role in signature verification are described. Next, HMMs are applied to signature verification. HMMs have properties that are well-suited for operation in a network, where many signature verification applications lie. Results of experiments are that HMMs seem to provide a solid ground on which to build a signature verification system.

List of symbols

$\mathbf{A}$	State transition matrix
$\mathbf{B}$	Symbol generation matrix
c_t	Scaling coefficient at time t
K	Number of observation sequences
M	Number of symbols
N	Number of states
$\mathbf{O}$	Observation sequence vector
P^{BW}	Baum-Welch probability
P^{V}	Viterbi probability
Q	Number of quantization levels
s	Distance along the curve
t	Normalized distance
v_k	k^{th} symbol in the alphabet
w_i	w^{th} word in the vocabulary
α	Forward partial probability
ß	Backward partial probability
$\epsilon_{quantization}$	Quantization distortion measure
$\epsilon_{intraclass}$	Within-class scatter measure
π	Initial probability vector
$\Theta(s)$	Absolute angular direction
$\phi(t)$	Cumulated angular difference
$\phi[k]$	discrete absolute angular direction
$\phi^*[k]$	Quantized discrete absolute angular direction
γ_{ij}	Expected number of transitions from state s_i to state s_j

γ_i	Expected number of transitions out of state s_i
$\psi_t(j)$	Most likely state at time t-1 given state s_j at time t

List of abbreviations

ASR	Automatic Speech Recognition
BW	Baum-Welch
DCC	Differential Chain Coding
DP	Dynamic Programming
FA	False Acceptance
FR	False Rejection
GFD	Generalized Fourier Descriptor
HMM	Hidden Markov model
LTR	Left-to-right
ML	Maximum Likelihood
OCR	Optical Character Recognition
SNR	Signal-to-Noise Ratio

Table of contents

Chapter 1.

Introduction

1.1. THE PEN AS AN INPUT DEVICE

With the introduction of the Integrated Services Digital Network (ISDN), multimedia communications applications are gaining importance. The integration of data from different sources into a single terminal opens a wide variety of new applications, such as voice-data systems and telewriting [16]. Due to this integration, the user interface, or the man-machine part of the system, becomes a very important aspect of the system.

An interesting step in the evolution of the user interface was the introduction of the pen combined with a digitizing graphics tablet as a pointing device [6]. The tablet is used to capture line drawings, which usually are passed to a personal computer as (x,y) coordinates for further processing. A typical multimedia application of line drawings is telewriting, where a writing tablet is used to digitize line drawings, and a personal computer to encode and transmit the drawing. Apart from the fact that a pen is more ergonomic than for instance a mouse, it also enables the input of handwritten script as a special class of line drawings.

Recently, technology has become available that integrates the tablet with a display unit, thus renewing the interest in this matter. Since a pen enables the input of handwritten script, machine interpretation of handwritten text becomes an important field of research. If a machine can understand handwritten script, applications lie in a wide

variety of systems. It is worth to notice here that in western script, a pen probably is not a complete substitute for a typing keyboard, since average typing in transcription, e.g. copying text into the machine, is faster than handwriting. Applications merely lie in areas that require high interactivity or the use of direct pointing and manipulation, such as filling in forms. Applications where text as well as line drawings have to be captured could also benefit, since time-consuming switching from tablet to keyboard is not necessary any more. Finally, (remote) signature verification also is an important application.

Especially for signature verification systems, it is very important to develop techniques that combine well with a telecommunications environment, since often applications lie in this area: consider for instance centralized signature databases, and remote verification terminals.

1.2. CLASSIFICATION OF HANDWRITTEN SCRIPT

Basically, the problems related to classifying script can be divided according to the a priori knowledge available on the classes under study and the type of information extracted from the objects [7]. This information can be divided into two groups (see Table 1-1): a 'semantic' part, which refers to the message content of the handwriting data, and a 'singular' part, which refers to individual characteristics of the writer.

Table 1-1: Four types of classification problems.

Object	Type of Knowledge	
	A Priori Unknown Class	A Priori Known Class
Semantic Part	Cognition/Learning	Recognition
Singular Part	Identification	Verification

'Cognition' and 'recognition' systems try to recover the content of a message irrespective of the writer, while 'identification' and 'verification' systems try to establish the identity of the writer, irrespective of the written content. Fig. 1-1 shows how current handwriting studies are organized.

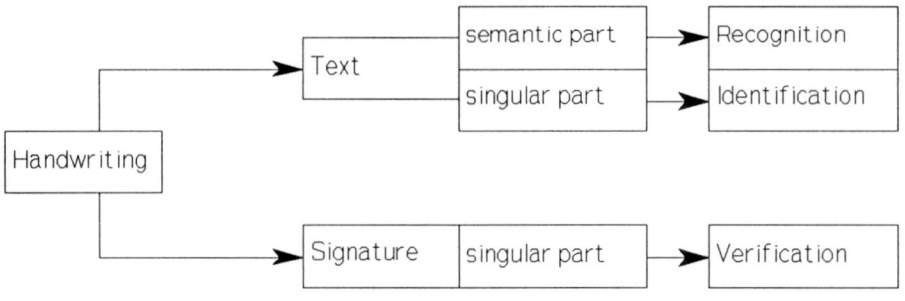


Fig. 1-1: Current handwriting studies.

In this report, we concentrate on the character recognition area from Fig. 1-1, combined with some work on signature verification. Although signature verification systems are more linked to a telecommunications environment than character recognition systems, a study of the latter leads to a good understanding of the phenomena that play a role, because of the simplicity of characters compared to signatures. Further, until now, no satisfactory method has been found that is able to on-line classify unconstrainedly handwritten script. Most commercial systems require the user to write in predefined boxes, and use only capital letters. Furthermore, these systems do not adapt easily to for instance personal writing styles.

Within the telecommunications section of TU Delft, several character recognition projects have been worked on [8]-[12]. Criteria for characterizations of line drawings were established by Prasad and Vieveen with the definition of Generalized Fourier Descriptors (GFDs)[8]-[10]. In this report, we shall elaborate some concepts introduced with the definition of GFDs. The performance of GFD-based systems was evaluated in [11] and [12]. Although GFDs are well-suited for characterization of line drawings, some

properties are less favourable when classifying handwritten script. Therefore a new area of research was opened, the application of Hidden Markov models to script classification.

1.3. CONTENTS OF THIS REPORT

In this report, hidden Markov Models (HMMs) are used to model the generation of handwritten script. HMMs have been successfully applied to speech recognition, and in a small degree also to character recognition. Principally, the application of HMMs requires a training stage, in which the model 'learns' the statistics of the underlying stochastic process, and a classification phase, in which an unknown character is 'scored' on the models. The training procedure is computationally very intensive, but can be done off-line, while the classification can be carried out in real-time. So, in a practical system, for instance signatures can be transmitted to a central host using for instance Differential Chain Coding (DCC). The host performs the training, and transmits the models back. Verification can then be done locally on the remote terminal. Fig. 1-2 shows this in a schematic way.

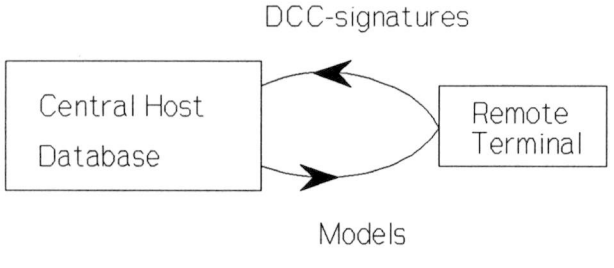


Fig. 1-2: Schematic example of the working environment of a signature verification system.

Chapter 2 starts with the theoretical background of HMMs. In a practical application of HMMs to the matching problem, several problems arise, like for instance computational feasibility of the training procedure. Solutions are offered in the form of powerful algorithms, like the Viterbi and the Baum-Welch algorithm.

Basically, in the application of a HMM, two families of model parameters are important; the state transition matrix **A** and the symbol generation matrix **B**. Chapter 3 discusses the symbols that are used, i.e. the character representation. This discussion is closely related to the family of the **B** parameters. This chapter extends some concepts introduced with the definition of GFDs.

Next, chapter 4 discusses the family of the **A** parameters. Constraints on this parameter (if any) can impose a certain model structure. This structure has to be chosen so that it best captures the real-world stochastic process of handwriting. In chapters 5 and 6, the results of extensive experimental work is presented. In chapter 5 optimum model parameters are empirically determined. With these parameters, writer-dependency tests are done in chapter 6. Next, chapter 7 discusses signature verification (see also Fig. 1-1). Some experiments on a HMM-based system are carried out. This topic, however, will not be treated as extensively as the character recognition. Finally, in chapter 8, conclusions are drawn and recommendations are given.

Chapter 2.

Hidden Markov Models applied to the matching problem

2.1. INTRODUCTION

The process of handwriting, like the process of speaking, can be regarded as a stochastic process, and thus be modelled using a stochastic model like the HMM. The advantage of this approach is that there is no need to explicitly codify subjective rules about the interpretation of patterns, whether they be speech features (for instance phonemes) or image features [3]. Due to the nature of the model, the rules are specified 'by example' during training or learning of the model. In this chapter, the mathematical background of the Hidden Markov Model is discussed. When applying HMMs to the signal matching problem, several problems arise. Solutions are introduced in the form of two algorithms, the Viterbi and the Baum-Welch algorithm. For each problem, one of the algorithms is chosen and implemented.

2.2. DESCRIPTION OF A HIDDEN MARKOV MODEL

A HMM is defined as a doubly stochastic process with an underlying stochastic process that is not observable, i.e., hidden, but can only be observed through another set of stochastic processes that produce the sequence of symbols [1]. Fig. 2-1 shows an example of a HMM.

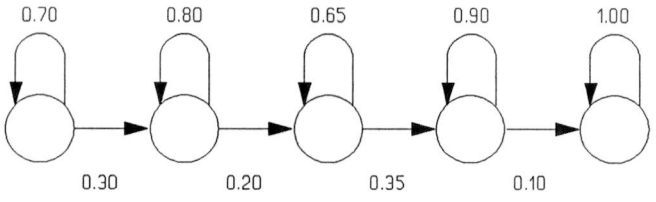


Fig. 2-1: Schematic example of a HMM.

The five circles represent the states of the model, and the numbers at the arrows the transition probabilities. At a discrete time instant t, the model occupies one of the states, and emits one observation. At time t+1, the model has either moved to a new state, or stayed in the same state. It also has emitted a new observation. This process repeats itself until a final terminating state is reached at time t=T. The model can be put into a matrix form as follows:

$$\boldsymbol{A}= \begin{bmatrix} 0.7 & 0.3 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.8 & 0.2 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.65 & 0.35 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.9 & 0.1 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} , \quad \boldsymbol{\pi}= [0.5\ 0.5\ 0.0\ 0.0\ 0.0] , \quad \boldsymbol{B}= \begin{bmatrix} 0.0 & 0.2 & 0.8 & 0.0 \\ 0.3 & 0.7 & 0.0 & 0.0 \\ 0.65 & 0.0 & 0.35 & 0.0 \\ 0.0 & 0.9 & 0.1 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

Fig. 2-2: Matrix representation of the previous example.

Here, a_{ij} denotes the probability of occupying state s_j at time t+1 given state s_i at time t. In the previous example only forward transitions are possible, so **A** is an upper triangular matrix. In a generalised HMM, however, transitions from any state to any other state are possible. We see that every transition only depends on the state occupied at time t; the

first-order Markovian property. Furthermore we notice the vector π, which stands for the initial state probability.

Consider now the generation of a symbol. When every symbol has a one to one correspondence with the states, we speak of a Markov Chain. In a HMM, however, a symbol is generated through an stochastic process. The **B** matrix above is an example of the representation of such a process for the previous example. Here, b_{jk} stands for the probability of emitting symbol v_k from state s_j. Since **B** has four columns, the set of symbols V contains four symbols $\{v_1,v_2,...,v_4\}$.

Almost all the ingredients necessary to formally define a HMM were discussed in the previous example. In the next section, a more formal definition of HMMs will be given.

2.2.1. Formal definition HMMs

The following model notation (see Fig. 2-3) uses symbols previously used in literature [1] to describe HMMs.

s_j	=	j^{th} state of the HMM	j = 1,2,...,N
v_k	=	k^{th} output symbol in the alphabet	k = 1,2,...,M
a_{ij}	=	Pr(state s_i @ t+1 |state s_i @ t)	i = 1,2,...,N
			j = 1,2,...,N
b_{jk}	=	Pr(output symbol v_k from state s_j)	j = 1,2,...,N
			k = 1,2,...,M
O_t	=	t^{th} observation (analysis vector)	t = 1,2,...,T
$b_j(O_t)$	=	Pr(output observation O_t from s_j)	
	=	b_{jk} when $O_t \rightarrow v_k$	
w_i	=	the i^{th} word in the recognition vocabulary	i = 1,2,...,W

Fig. 2-3: Formal definition HMMs

The HMM M is fully defined by the parameter set [π, **A**, **B**]. The following section will discuss the practical computational problems that arise when applying HMMs to signal classification.

2.3. TWO PROBLEMS FOR HMM'S

In a practical classification system, we are given W HMMs, one for each item in the vocabulary. Given the form of HMM discussed previously, two computational problems arise that must be solved before the HMM can be useful in an application. Solutions to these problems will be presented in the following sections. The problems are:

1. Given an unknown signal **O**, which consists of a sequence of T observations: **O** = $O_1,O_2,...,O_T$; each O_i is one of the symbols $\{v_1,v_2,...,v_M\}$. Given also the model M = **[A, B, π]**, how to compute Pr(**O**|M), the probability that the observation sequence is produced by the model.

2. How to adjust the model parameters M = **[A, B, π]** in order to maximize Pr(**O**|M), that is, to optimally train the model on a single or on multiple observation sequences of each item in the vocabulary.

2.4. PROBLEM 1: THE RECOGNITION PROBLEM

This problem is typical the classification problem. Given an observation sequence of an unknown signal, it involves computing the likelihood of each model emitting this observation sequence, and assigning the unknown signal to the class of the model that produced the greatest likelihood of having emitted the observation sequence. This kind of classification is called Maximum Likelihood (ML) classification.

Since we wish to calculate the probability of the observation sequence **O** given the model M, the most straightforward way of doing this would be by enumerating every possible state sequence of length T. Given the state sequence in time $S = s_1,s_2,...,s_T$, the probability of the observation sequence **O** is Pr(**O**|S,M), with

$$Pr(O|S,M)=b_{s_1}(O_1)b_{s_2}(O_2)...b_{s_T}(O_T) \qquad \textbf{(2-1)}$$

The probability of the state sequence S itself is:

$$Pr(S|M)=\pi_{s_1}a_{s_1}a_{s_2}a_{s_3}...a_{s_{T-1}}a_{s_T} \qquad \textbf{(2-2)}$$

The probability that **O** and S occur simultaneously is simply the product of the above two terms. The probability of **O** is then obtained by summing this joint probability over all possible state sequences:

$$Pr(O|M) = \sum_{all\,possible\,state\,seq.} Pr(O|S,M)$$

$$= \sum_{s_1,s_2,\ldots,s_T} \pi_{s_1} b_{s_1}(O_1) a_{s_1 s_2} b_{s_2}(O_2) \ldots a_{s_{T-1} s_T} b_{s_T}(O_T) \qquad \textbf{(2-3)}$$

The equation above can be interpreted as follows: Initially, at time t=1, the model is in state s_1 with probability π_{s1}, and generates the symbol O_1 with probability $b_{s1}(O_1)$. Next, a transition is made to state s_2 with probability a_{s1s2}, and symbol O_2 is generated with probability $b_{s2}(O_2)$. This process continues until time has reached t=T.

It may be clear, that due to the previously given definition, calculations of the order $2 \cdot T \cdot N^T$ are required, since at every time instant t = 1,2,...,T there are N possible states to go through, and for each summand about 2•T calculations are required. To be precise, $(2 \cdot T-1) \cdot N^T$ multiplications and N^T-1 additions are needed. Even for small values of N and T, this is unfeasible (consider N=5 and T=100, this leads to the order of $2.100.5^{100} \approx 10^{72}$ calculations!). Fortunately, more efficient procedures exist.

2.4.1. Baum-Welch recognition

An efficient procedure that tackles the problem from the previous section is the so called forward-backward procedure by Baum. This algorithm uses a 'forward'-variable, which stands for the probabilities of the joint events of emitting the partial observation sequence (until time t) $O_1,O_2,\ldots,O_t$ and of occupying state s_j at time t, and a complementary 'backward'-variable. Since these probabilities will be used later in the Baum-Welch training algorithm, the corresponding $Pr(\mathbf{O}|M)$ is denoted by P^{BW}. The forward probabilities are defined as follows:

$$\alpha_t(j) = Pr(O_1,O_2,\ldots,O_t,\ state\ s_j\ @\ time\ t|M) \quad , \quad j=1,2,\ldots,N \qquad \textbf{(2-4)}$$

α_t can be calculated recursively. Suppose $\alpha_t(i)$, i=1,2,...,N , has been computed at some time instant t, then

$$Pr(O_1,O_2,...,O_t,\ state\ s_j\ @\ time\ t+1 \mid state\ s_i\ @\ time\ t,M)=\alpha_t(i)a_{ij} \qquad \textbf{(2-5)}$$

This means that the probability of occupying state s_j at time t+1 can be expressed as:

$$Pr(O_1,O_2,...,O_t,\ state\ s_j\ @\ time\ t+1 \mid M)=\sum_{i=1}^{N}\alpha_t(i)a_{ij} \qquad \textbf{(2-6)}$$

The result of this summation over all N possible states at time t results in the probability of s_j at time t+1 with all the accompanying previous partial observations. When this step has been taken it is not hard to see that α_{t+1} is obtained by augmenting multiplicatively the sum with the probability $b_j(O_{t+1})$, thus accounting for the observation O_{t+1}:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N}\alpha_t(i)a_{ij}\right] b_j(O_{t+1}) \quad , \quad t=1,2,...,T-1 \qquad \textbf{(2-7)}$$

Since $\alpha_1(j)=Pr(O_1,$ state s_j @ time $t=1 \mid M)$, the recursion can be initialized by setting $\alpha_1(j)=\pi(j)b_j(O_1)$. Fig. 2-4 illustrates the previous steps by showing the calculation of $\alpha_{t+1}(4)$ for a five-state HMM.

The complementary 'backward'-probabilities are defined as follows:

$$\beta_t(i)=Pr(O_{t+1},O_{t+2},...,O_T\ state\ s_i\ @\ time\ t \mid M) \qquad \textbf{(2-8)}$$

So, $\beta_t(i)$ gives the probability of starting in state s_i at time t after which the observation sequence is completed, starting with observation O_{t+1} (not O_t). $ß_t(i)$ can be calculated using the same kind of recursion as described for $\alpha_{t+1}(j)$:

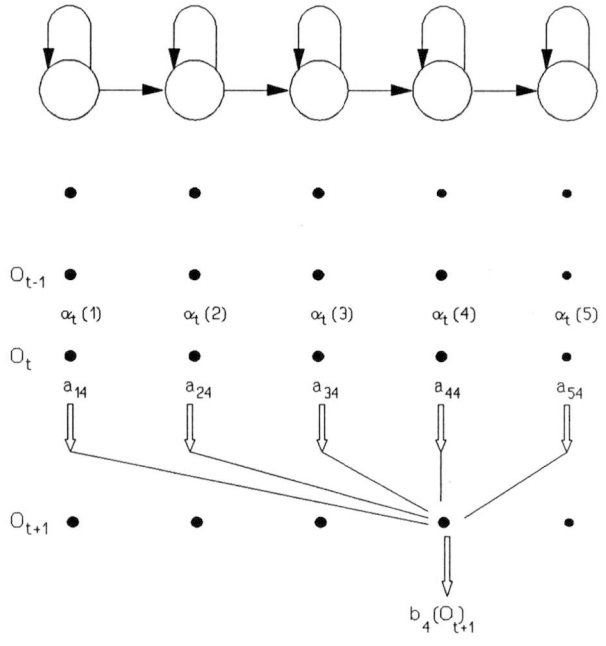


Fig. 2-4: Computation of $\alpha_{t+1}(4)$ for a five-state HMM.

$$\beta_t(i)=\sum_{j=1}^{N} a_{ij}b_j(O_{t+1})\beta_{t+1}(j) \quad , \quad t=T-1,T-2,...,1 \tag{2-9}$$

with $\beta_T(i)=1$ for i=1,2,...,N. The forward- and backward probabilities now can be combined to compute the desired probability P^{BW} by:

$$P^{BW}=\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j) \quad , \quad 1\le t\le T-1 \tag{2-10}$$

Setting t=T-1 in eq. (2-10) yields

$$P^{BW}=\sum_{j=1}^{N} \alpha_T(j) \tag{2-11}$$

so P^{BW} can be computed from the forward probabilities alone. Clearly, a similar formula for P^{BW} can be obtained from the backward probabilities by setting t=1.

A brief look at the computational effort involved tells us that about the order of N^2T calculations are required; to be precise, $N \cdot (N+1) \cdot (T-1) + N$ multiplications and $N \cdot (N-1) \cdot (T-1)$ additions. Using the figures in the previous example, N=5 and T=100, this means that about 3000 calculations are necessary; compared to the order of 10^{72} when using direct calculation, this means quite a reduction in computing time.

2.4.2. Viterbi recognition

In the previous section, P^{BW} represents the likelihood of emitting **O** summed over all possible state sequences, since each α_{t+1} is calculated by summing the contributions of α_t over all state sequences. An algorithm that calculates the likelihood of the most likely state sequence emitting the observation **O** also exists: the Viterbi algorithm, which is a form of the well-known Dynamic Programming method. The corresponding maximum likelihood is denoted by P^V. Using the backtracking procedure, the optimal state sequence can be recovered.

To compute P^V, the parameter $\phi_{t+1}(j)$ is introduced:

$$\phi_{t+1}(j) = \max_{i=1,2,\ldots,N} [\phi_t(i) a_{ij}] b_j(O_{t+1}) \quad , \quad t=1,2,\ldots,T-1 \tag{2-12}$$

P^V can then be found as:

$$P^V = \max_{j=1,2,\ldots,N} \phi_T(j) \tag{2-13}$$

Eq. (2-12) and (2-13) are identical to eq. (2-7) and (2-11), except that the summation is replaced by the max operator. Note that $P^V \leq P^{BW}$, with the equality satisfied only when the state sequence that produces the observations is unique.

As mentioned previously, it is possible to recover the most likely state sequence that could have emitted **O** using the backtracking procedure. Comparison of this sequence

with the physical signal may lead to further model improvements. The backtracking procedure involves the following steps:

A variable $\psi_t(j)$ is defined, representing the most likely state at time t-1 given state s_j at time t. In other words, $\psi_t(j)=i^*$, where i^* is the number of the state that maximizes the right-hand side of eq. (2-12). The backtracking algorithm proceeds as follows: at time T, the most likely state is s_k, where k maximises the right-hand side of eq. (2-13). ψ gives then the most likely state at time T-1, from which the most likely state at time T-2 can be found. This process continues until the most likely state at time t=1 has been found.

2.5. PROBLEM 2: THE TRAINING PROBLEM

This problem is the most difficult of the two problems associated with the application of HMMs. There is no way to solve this problem for a maximum likelihood analytically. Therefore, iterative procedures or gradient techniques (like the classical method using Lagrange multipliers) must be used for optimization. In this section, algorithms are discussed that optimize a HMM given an observation sequence. We start with a single observation sequence for each item in the recognition vocabulary; however, the extension to multiple observations is straightforward and will be discussed in section 2.5.2.

The steps that must be taken in a training algorithm are as follows:

1. make an initial estimate of the model parameters M.
2. use reestimation and **O** to generate a new model M', with the property that $\Pr(\mathbf{O}\,|\,M') \geq \Pr(\mathbf{O}\,|\,M)$.
3. M' now plays the role of M, and the process is repeated until the inequality in the expression above is arbitrary small.

Two training algorithms will be discussed in the following sections: the Baum-Welch and the Viterbi algorithm.

2.5.1. The Baum-Welch training algorithm

The general idea behind this algorithm is that given a model M and an observation **O**, the best estimates of the new model M' are given by:

$$a'_{ij}=\frac{Pr(transition\ from\ state\ s_i\ to\ state\ s_j|M)}{Pr(transition\ from\ state\ s_i\ to\ any\ state|M)} \qquad \textbf{(2-14a)}$$

$$b'_{jk}=\frac{Pr(emiting\ symbol\ v_k\ from\ state\ s_j|M)}{Pr(emitting\ any\ symbol\ from\ state\ s_j|M)} \qquad \textbf{(2-14b)}$$

$$\pi'_i=Pr(observation\ sequence\ begins\ in\ state\ s_i|M) \qquad \textbf{(2-14c)}$$

The Baum-Welch algorithm has the remarkable property that the reestimations of **A**, **B**, and $\boldsymbol{\pi}$ are guaranteed to increase $Pr(\mathbf{O}|M)$ until some critical point, from which the parameters do not change any more.

The quantities mentioned above can be computed by defining γ_{ij} as the expected number of transitions from state s_i to s_j, conditioned on the observation sequence:

$$\gamma_{ij}=\frac{1}{P^{BW}}\sum_{t=1}^{T-1}\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j) \qquad \textbf{(2-15)}$$

Recalling the definitions of $\alpha_t(i)$ and $\beta_t(i)$,

$$\alpha_t(i)=Pr(O_1,O_2,...,O_t,\ state\ s_i\ @\ time\ t|M)$$

$$\beta_t(i)=Pr(O_{t+1},O_{t+2},...,O_T,\ state\ s_i\ @\ time\ t|M)$$

it is not hard to see that $\alpha_t(i)ß_t(i)$ denotes the probability of occupying state s_i at time t, given the model M. Therefore the expected number of transitions γ_i out of s_i, given **O** is defined as:

$$\gamma_i=\sum_{j=1}^{N}\gamma_{ij} = \frac{1}{P^{BW}}\sum_{t=1}^{T-1}\alpha_t(i)\beta_t(i) \qquad \textbf{(2-16)}$$

Now, we can express the estimates of the entries of the state transition matrix of the new model M' in terms of the forward and backward probabilities, since the ratio γ_{ij}/γ_i represents an estimate of the probability of state s_j given the previous state s_i:

$$a'_{ij}=\frac{\gamma_{ij}}{\gamma_i} = \frac{\sum_{t=1}^{T-1}\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\sum_{t=1}^{T-1}\alpha_t(i)\beta_t(i)} \qquad \textbf{(2-17a)}$$

A similar reasoning leads to the estimates b'_{jk} as the frequency of occurrence of v_k in state s_j relative to the frequency of occurrence of any symbol in state s_j.

$$b'_{jk}=\frac{\sum_{t\ni O_t=v_k}\alpha_t(j)\beta_t(j)}{\sum_{t=1}^{T}\alpha_t(j)\beta_t(j)} \qquad \textbf{(2-17b)}$$

The summation in the numerator of eq. (2-17b) should be read as: 'sum over all t's at which the observation O_t is symbol v_k'.

Finally, there is one more reestimation to make, which is the one for the initial state probability π. It is given by:

$$\pi'(i)=\frac{1}{P^{BW}}\alpha_1(i)\beta_1(i) \qquad \textbf{(2-17c)}$$

In [1], the proof according to Levinson et al [2] is given that $\Pr(\mathbf{O}|\mathrm{M})$ indeed increases with every parameter update using the method described above. Finally it should be mentioned that the above procedure leads to an locally maximized $\Pr(\mathbf{O}|\mathrm{M})$.

2.5.2. Training on multiple observation sequences

The reestimation formulas (2-17a) to (2-17c) can easily be extended to handle multiple observation sequences on the same item in the recognition vocabulary. The most straightforward way is to consider eq. (2-17a) to (2-17c) to act over all the observation sequences. The modification to the training procedure is as follows: given $\mathbf{O} = [\mathbf{O}^{(1)}, \mathbf{O}^{(2)}, ...,\mathbf{O}^{(K)}]$ the set of observation sequences, where $\mathbf{O}^{(k)} = \mathbf{O}_1^{(k)}\mathbf{O}_2^{(k)}...\mathbf{O}_{Tk}^{(K)}$ the k^{th} sequence. The observation sequences are treated independent of each other, and we try to optimize the parameters of the model M in order to maximize

$$P=\prod_{k=1}^{K} Pr(\boldsymbol{O}^{(k)}|M) = \prod_{k=1}^{K} P_k^{BW} \qquad \textbf{(2-18)}$$

However, each observation sequence has a different P^{BW}. In the reestimation formulas eq. (2-17a) to (2-17c) this factor cancelled out since only one observation sequence was used, but clearly it has to be taken into account when using multiple observation sequences. Otherwise,the probabilities from sequences having a low P^{BW} will give a disproportional low contribution to the reestimations. The result will be that the model is only optimized for observation sequences having a high P^{BW}.

Denote N_k and D_k by the numerator and denominator of eq. (2-17a), formed when processing observation sequence $\mathbf{O}^k$, and let P_k^{BW} be $\Pr(\mathbf{O}^{(k)}|\mathrm{M})$. Let K be the total

number of observation sequences in the training set. The reestimation can then be given by:

$$a'_{ij} = \frac{\sum_{k=1}^{K} \frac{1}{P_k^{BW}} N_k}{\sum_{k=1}^{K} \frac{1}{P_k^{BW}} D_k} \qquad \textbf{(2-19)}$$

For eq. (2-17b) and (2-17c), similar approaches can be followed.

2.5.3. Viterbi training

The Viterbi algorithm can also be used for training of the model. Since the backward probabilities are not used, there is a computational saving. The Viterbi algorithm is used to segment each observation sequence into the most likely state sequence emitting this symbol sequence. The new values for [**A**, **B**, π] are then derived by examining the number of transitions to and from each state and the symbols outputted by each state. The procedure is shown below.

1. Make an initial estimate M^0 of the model M.
2. Execute the Viterbi algorithm on each of the observation sequences $\mathbf{O}^1,\mathbf{O}^2,...,\mathbf{O}^K$. Store the set of most likely state sequences produced, $S^1,S^2,...,S^K$, and set

$$L=\sum_{k=1}^{K} Pr(O^{(k)}|M)$$

3. Use the Viterbi reestimation formulas (2-20a), (2-20b) and (2-20c) to generate a new M.
4. Repeat steps 2 and 3 until the increase in L is arbitrary small in successive iterations.

The reestimations are given by applying all the sequences $S^1,S^2,...,S^K$ to the following equations:

$$\overline{a_{ij}} = \frac{(N^{o} \text{ of transitions state } s_j \text{ to state } s_j | M)}{(\text{Total } n^{o} \text{ of transitions out of state } s_i | M)} \qquad \textbf{(2-20a)}$$

$$\overline{b_{jk}} = \frac{(N^{o} \text{ of emissions of symbol } v_k \text{ state } s_j | M)}{(\text{Total } n^{o} \text{ of symbols emitted from state } s_j | M)} \qquad \textbf{(2-20b)}$$

$$\overline{\pi_i} = \frac{(N^{o} \text{ of observation sequences beginning in state } s_i | M)}{U} \qquad \textbf{(2-20c)}$$

Since numbers of transitions are used here rather than probabilities, weighting by 1/P is not necessary.

2.6. PRACTICAL ISSUES

In the previous sections, two problems for HMMs were discussed: the classification problem and the training problem. The Baum-Welch (forward-backward) as well as the Viterbi algorithm can be used to make the problems computationally feasible. However, some practical aspects must be considered for the procedures to be maximally useful.

2.6.1. Preventing underflow

An important practical aspect concerns the implementation of the forward-backward computation. A brief look at the recursions eq. (2-7) and (2-9) shows that the $\alpha_t(i)$ and $ß_t(i)$ tend to zero geometrically fast. A scaling technique of the α's and ß's must be used to avoid mathematical underflow. Here, a scaling technique according to [1] is discussed.

Principally, the scaling is based on multiplication of $\alpha_t(i)$ by some scale factor independent of i so that $\alpha_t(i)$ remains within the dynamic range of the computer for $1 \leq t \leq T$. On $ß_t(i)$ a similar operation can be performed. Then, at the end of the

computations, the total effect of the scaling can be removed. Let the scaling coefficient c_t be denoted by

$$c_t = \frac{1}{\sum_{i=1}^{N} \alpha_t(i)} \tag{2-21}$$

$\alpha_t(i)$ is computed using eq. (2-7), and scaled by c_t. Then, as $\beta_t(i)$ is computed by eq. (2-9), for $T \leq t \leq 1$ and $1 \leq i \leq N$ the product $c_t \beta_t(i)$ is formed. Eq. (2-17a) is changed in the following manner:

$$a_{ij}' = \frac{\sum_{t=1}^{T-1} C_t \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) D_{t+1}}{\sum_{t=1}^{T-1} \sum_{l=1}^{N} C_t \alpha_t(i) a_{il} b_l(O_{t+1}) \beta_{t+1}(l) D_{t+1}} \tag{2-22}$$

where

$$C_t = \prod_{\tau=1}^{t} c_\tau$$
$$D_t = \prod_{\tau=1}^{T} c_\tau \tag{2-23}$$

This is the result from the individual scale factors being multiplied together as the recursions for α and β are carried out. Note that each summand in both the numerator and the denominator of eq. (2-22) has the coefficient

$$C_t D_{t+1} = \prod_{\tau=1}^{T} c_\tau \tag{2-24}$$

These coefficients can be factored out and cancelled so that eq. (2-22) represents the correct value for a'_{ij}, as specified by eq. (2-17a). According to [1], similar reasoning leads to the reestimation formulas for the **B** and π, i.e. for eq. (2-17b) and (2-17c). However, some care must be taken in this case:

Recalling the reestimation formula for the **B** parameters,

$$b'_{jk} = \frac{\sum_{t \ni O_t = v_k} \alpha_t(j)\beta_t(j)}{\sum_{t=1}^{T} \alpha_t(j)\beta_t(j)}$$

We note that in terms of the scaled forward and backward probabilities eq. (2-17b) becomes

$$b'_{jk} = \frac{\sum_{t \ni O_t = v_k} C_t \alpha_t(j)\beta_t(j) D_t}{\sum_{t=1}^{T} C_t \alpha_t(j)\beta_t(j) D_t} \tag{2-25}$$

So, each summand in eq. (2-25) contains the factor

$$C_t D_t = \prod_{\tau=1}^{t} c_\tau \prod_{\tau=t}^{T} c_\tau = c_t \prod_{\tau=1}^{T} c_\tau \tag{2-26}$$

Because the extra factor c_t is time-dependent, the factors C_t and D_t cannot be factored out and cancelled as straightforward as with the **A** parameters. In each summand of both the numerator and the denominator of eq. (2-25), a factor c_t has to be cancelled before each addition in order for the procedure to function correctly.

Although the above described scaling technique leaves the reestimation formulas invariant, the classification formulas (2-10) and (2-11) are still useless in practical computations of P^{BW}. However, $\log(P^{BW})$ can be recovered from the scale factors as follows. Assume that c_t has been calculated according eq. (2-21) for t=1,2,...,T. Then

$$C_T \sum_{i=1}^{N} \alpha_T(i) = 1 \tag{2-27}$$

and it may be clear that according to eq. (2-27) $C_T = 1/P^{BW}$. Hence, from (2-23) we have

$$\prod_{t=1}^{T} c_t = \frac{1}{P^{BW}} \tag{2-28}$$

This product of individual scale factors cannot be evaluated directly but the logarithm can be computed as follows:

$$\log(P^{BW}) = -\sum_{t=1}^{T} \log(c_t) \tag{2-29}$$

In other words, $\log(P^{BW})$ can be found very easily by summing the logarithms of the individual scale factors.

When the Viterbi algorithm is used for classification, $\log(P^{V})$ can be computed directly from $\boldsymbol{\pi}$, **A**, and **B** without regard to the scale factors. Initially, we let $\phi_1(i) = \log[\pi_i b_i(O_1)]$, after which eq. (2-12) is modified in the sense that

$$\phi_t(j) = \max_{1 \le i \le N} [\phi_{t-1}(i) + \log(a_{ij})] + \log[b_j(O_t)] \tag{2-30}$$

In this case, we find that

$$\log(P^V) = \max_{1 \le i \le N} [\phi_T(i)]$$

2.6.2. Zero symbol probabilities

A second problem concerns the use of a finite set of training data for estimation of the HMM parameters. When we look at the reestimation formulas (2-17a) to (2-17c), we see that a parameter will be set to 0 if there are no occurrences in the training set. Usually this happens with one or more symbol probabilities. If this effect is due to the small size of the training set, care must be taken that no HMM parameter becomes too small. If it is a real effect, a zero probability is perfectly reasonable. Note that setting a state transition probability to zero can be used to constrain a model, which will be widely discussed in chapter 4.

The problem on for instance the b_{jk}-coefficients can be effectively tackled by using post-estimation constraints on the b_{jk}'s of the form

$$b_{jk} \ge \epsilon$$

where ϵ is a suitably chosen threshold. All b_{jk}'s that are below ϵ are replace by ϵ_j for each j. Let R be the number of changed b_{jk}'s, then after this operation, each b_{jk} that was not changed is rescaled in the following manner:

$$\tilde{b}_{jk} = (1 - R\epsilon) \frac{b_{jk}}{\sum_{i=1}^{N-R} b_{ji}} \quad \forall\ k \notin \{ k_i \mid 1 \le i \le R \} \tag{2-31}$$

Finally, we mention here that the constraints may be imposed after each iteration of the reestimation formulas, or once as a post-processing stage after the Baum-Welch algorithm has converged. A proof can be found in [1]. Experiments by Rabiner [2] have

shown that in practice, the estimation error is not affected when using the post-estimation constraints in one final stage.

2.6.3. Extension to continuous probability density functions

Until now, we have only considered the case of discrete symbol HMMs, i.e. the observation sequence is a set of M discrete symbols. The rows of the **B** matrix represent the discrete probability density given the state. It is possible to extend the model to continuous symbols: the rows of **B** are replaced by parameters of the continuous probability density.

This technique is used in for instance Automatic Speech Recognition (ASR), where the observation sequence is an analogue signal. Using discrete symbols here involves a process of quantization, with inherent quantization distortion. This distortion can be minimized by using a large number of quantization symbols, however, this means that a large amount of training data must be available to avoid problems as mentioned in section 2.6.2. So, if a known continuous probability density accurately describes the underlying process, this method can improve performance. In ASR, weighted sums of multivariate Gaussian probability density functions (PDFs), by which any multivariate PDF can be approximated, have been used. The reestimation formulas can be extended to handle any log-concave distribution.

However, this requires an extensive study of vector probabilities under different styles of handwriting, which is far beyond the scope of this report. Therefore, only discrete symbols will be used.

Chapter 3.

The Symbol Generation Matrix: Feature Extraction

3.1. INTRODUCTION

The previous chapter discussed the theory of HMMs and the practical problems that arise in the implementation of HMMs. We saw that a HMM is fully determined by its parameters [**A**, **B**, π]. The next step is to investigate which model captures best the underlying process of handwriting. When we look at the parameter set defining a HMM, we see that basically, the performance of a HMM-based recognition system is dependent of two parameters: the state transition probability matrix **A**, and the symbol probability matrix **B**. The state transition matrix, i.e. the model structure, will be discussed in the next chapter. In this chapter, the symbol probability matrix **B** will be treated. Central in this discussion will be the relation between the symbols and features or descriptors that we can extract from a character.

3.2. PATTERN CLASSIFICATION SYSTEMS

The investigation of the symbols to use starts with a look at a typical pattern classification system (see Fig. 3-1).

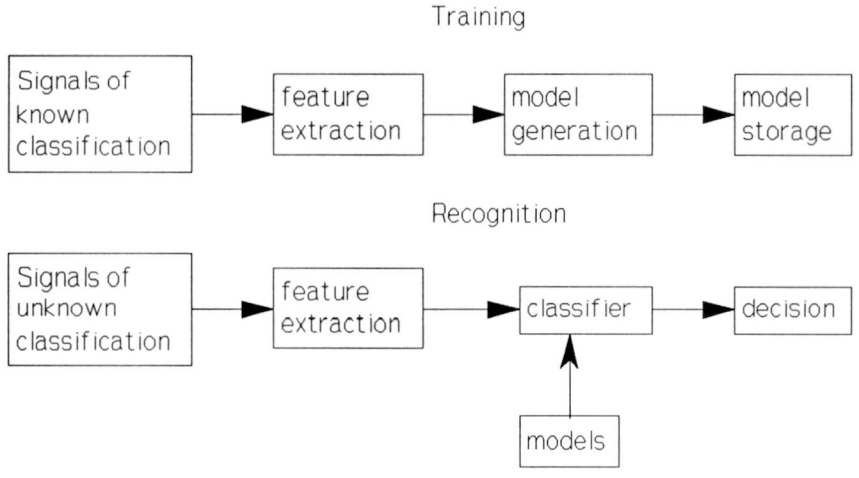


Fig. 3-1: Basic pattern classification system.

As can be seen, the system has a training mode, in which it is trained on a specific writer, i.e. where the model is optimized, and a recognition mode, in which a pattern of unknown class is classified by comparing it with each model in a vocabulary and assigning it to the model to which it is 'closest' according to some criterion.

The feature extraction (i.e. 'symbol sequence generation') part is needed for two reasons:

1. it enables focusing on information within the signal which is important for discriminating between patterns of different classes. A good set of features 'enhances within-class similarity and between-class dissimilarity' [3]. This will be further elaborated in section 3.4.1
2. it reduces the amount of data, which is necessary for computational reasons.

It may be clear that the choice of good features is an essential part in any pattern classification system. When the features replicate each other, or show an unstable behaviour, the total system performance will be determined by the features, no matter how good the classifier is.

Some kind of preprocessing is often carried out to condition the signal, and may include filtering, noise reduction, and normalization. Since the type of preprocessing that is

required depends on the curve description and the feature extraction, it will be treated last in this chapter.

Finally, there is a difference between on-line and off-line processing.

3.2.1. On-line versus Off-line recognition

In off-line (static) processing, the image is captured after the writing is completed, for instance using a TV camera or an optical scanner. The image is usually stored as a two-level bit-image pattern. Off-line handwriting recognition is a subset of optical character recognition (OCR). Although most OCR work has been done on machine-printed characters, it has been applied on handwriting as well.

On-line (dynamic) processing, on the contrary, captures the writing as it is written. The temporal order of the character is thus preserved. The input device usually is a digitizing tablet.

The differences between on-line and off-line recognition seem clear, but there are some more subtle differences that should be mentioned here, since they affect the requirements on a recognition system.

An application of off-line recognition is for instance automatic reading of postal cheques or ZIP-codes. So, off-line systems are required to process several hundreds of characters a second, each with a very high accuracy. Off-line systems usually run on mainframe computers to meet the speed requirements.

On-line systems, on the contrary, mostly operate in an interactive environment. Characters are recognized as they are written, so recognition needs only be as fast as the writing itself. However, since on-line systems should be able to run on microcomputers, off-line techniques are computationally too intensive to use in on-line systems. Further, due to the feedback from the user, higher error rates are tolerable. In this report, an on-

line technique will be discussed. The symbols that can be used to represent a character can depend on the coding method, so we shall first investigate the writing tablet.

3.3. THE TABLET

The writing tablet, type DSD 703, was manufactured especially for the GraphiMail project at TU Delft [16]. It is a high-speed digitizer, and has a writing surface of 21x15 cm. plus 40 additional fields for control purposes (see Fig. 3-2). It is connected to a PC via standard communications lines.

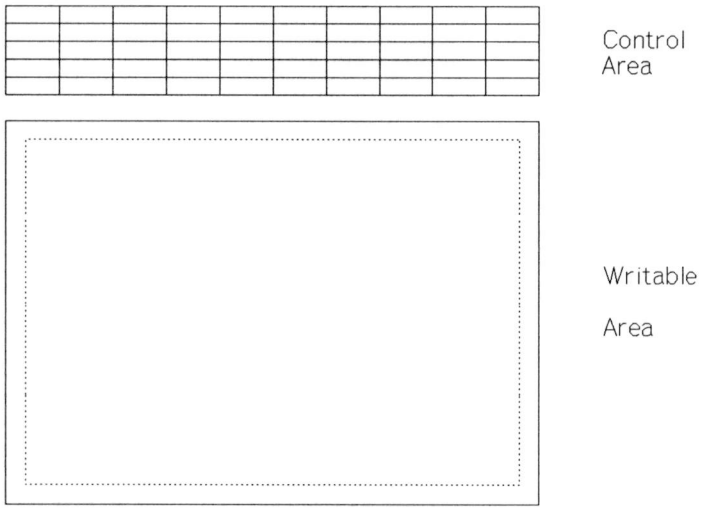


Fig. 3-2: the writing tablet.

The writing area contains a grid of 1536 horizontal and 2100 vertical lines, which means that the resolution in both the x and y direction is about 100 points/cm. According to [6], a digitizing tablet should have a spatial resolution of at least 200 points/inch, so these requirements are sufficiently met.

3.3.1. Tablet quantization method

Pen-down, or 'inking' is registrated, as well as 'pen-up'. The quantization process involves the following steps (see Fig. 3-3):

1. registration of pen-down.
2. place a rectangular frame (the coding 'ring') around the current pen position, and wait for the pen to intersect this frame or to be lifted (pen-up).
3. make this intersection the new current position, and repeat step (2) until a pen-up takes place.

Table 3-1: Modes and resolutions.

Mode:	EGA			Grafivision		
Frame:	1	2	3	1	2	3
Resolution in gridlines:	3x4	6x8	9x12	4x3	8x6	8x9
No. of possible codevectors:	28	56	84	28	56	68

So, the tablet typically uses space sampling rather than time sampling. The size of the coding ring can be chosen as one of three differently-sized frames around the current point. Two operating modes are available, with corresponding frame sizes expressed in tablet gridlines. Table 3-1 lists the operating modes and resolutions, and Fig. 3-3 shows and example of the quantization resolution and the frame size.

The original curve is thus quantized into a number of straight-line sections, the codevectors. Clearly, the size of the coding ring determines the amount of possible codevectors, and thus the quantization distortion. However, we have to make the following notes:

1. A sampling process is involved, so inherently we have to meet the Nyquist theorem: the bandwidth of the signal generated by the stylus may not be larger than twice the

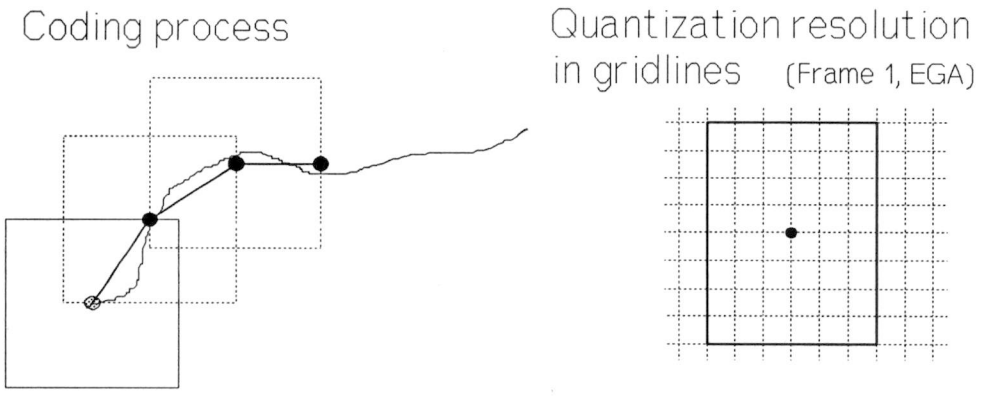


Fig. 3-3: Coding process and quantization resolution.

sampling frequency in order to avoid aliasing. Note that in this respect, we speak of spatial bandwidth rather than temporal bandwidth. Since the tablet resolution in gridlines per unit length is fixed, and thus the sampling frequency is fixed, we have to control our handwriting so that the Nyquist theorem is met, i.e. the curvature of the original curve has to be restricted [10].

2. Choosing for instance frame 3 instead of frame 1 reduces the quantization error (there are more possible codevectors, i.e. quantization levels), but also reduces the allowed bandwidth by a factor three, since the spatial resolution is also reduced by a factor three. If we speak of the signal-to-noise ratio (SNR) as a measure for the quantization distortion, it should be mentioned that oversampling (i.e. choosing a smaller frame than required) also increases SNR.
3. The codevectors are not of equal length. See Fig. 3-4 : clearly the codevectors pointing towards the corners of the frame are $\sqrt{2}$ times longer than the codevectors along the x- or y axis. As we shall see, this property somewhat complicates normalization procedures.

Frame 2 appears to give the best trade-off between sampling rate and quantization distortion for handwriting.

Finally, we note that this type of coding is also known as Freeman chain coding of line drawings [13].

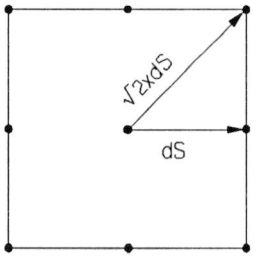


Fig. 3-4: Different length of codevectors.

3.4. REQUIREMENTS ON FEATURES

Since the digitizing method has been discussed, we can move on to formulate requirements that the features extracted from the sampled character should meet. In the formulation of these requirements, some concepts play an important role. The next section introduces these concepts using an experiment-of-thought.

3.4.1. Some important concepts

Consider an alphabet of, say, 5 characters. Consider also an human being that produces specimens of each character. We assume, that he or she writes in such a manner that any other human could correctly recognize any written character. The produced specimen show so-called intraclass variation (denoted by the rectangles in Fig. 3-5), however, the features extracted by the recognizing human from the specimen do not show class overlap, so the human is able to score a 100% recognition rate. The interclass variation is large enough to distinguish the characters from each other. Or, in other words, a threshold can be found that completely separates the characters from each other.

Consider now a less ideal situation, for instance a very sloppy writer. Again the specimen of each character will show intraclass variation, however, this time there is class overlap. Due to the decreased interclass variation or the increased intraclass scatter, even our

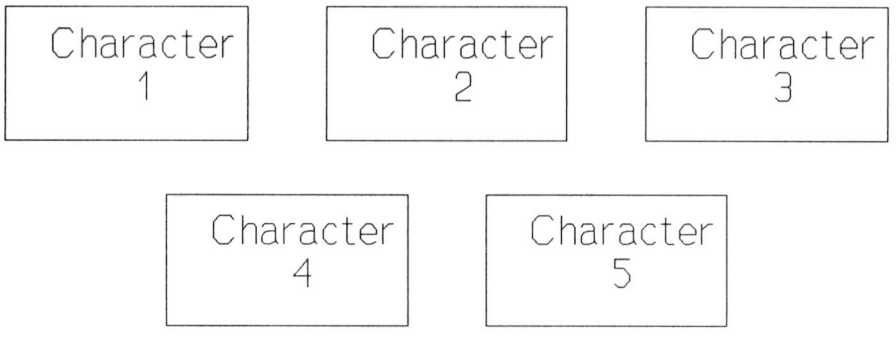


Fig. 3-5: Ideal class separation.

recognizing human cannot score 100% recognition rate, and will make errors when recognizing specimen that fall in the shaded areas of Fig 3-6. For instance, a specimen of character 4 may be misinterpreted as character 2. Note that human readers have significant error rates when recognizing characters out of their context: for isolated block characters, handprinted in English, error rates of 4% and 1.25 % have been reported [6]. For lowercase writing, an error rate of 2.4%, and for cursive writing an error rate of 4.4% has been reported [6].

A global feature requirement can be formulated now: features should show as little intraclass variation as possible (small rectangles in the previous example), and should have as much interclass discriminant power as possible (distance between the rectangles).

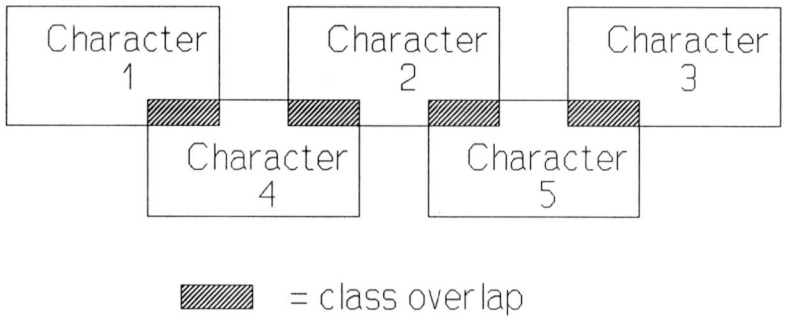


Fig. 3-6: Practical class overlap.

Finally, it is worth to mention here that HMMs were chosen as a model for handwritten script, because they are able to capture a great variability in the handwriting. It may seem contradictionary, that we try to reduce intraclass variability of the symbols in this chapter. However, recalling section 2.6.2., the advantage of reduced intraclass scatter is obvious; it implies that less training data are required.

3.4.2. Symbol requirements

The global feature requirement formulated previously can be contradictionary: an increased discriminant behaviour between classes can also increase the variation within a class. We can, however, intuitively formulate more specific requirements on the symbol set: a good set of features should

1. be independent of translation, rotation and scaling of the line drawing.
2. be chosen so that they do not replicate each other
3. be easily computable
4. preferably (not necessarily) employ the dynamic information provided by the tablet.

The first requirement is obvious; when this requirement is not met, an intraclass variability is introduced even for equally shaped characters. It may also be clear that different characters should not be represented by identical sets of features, as stated in the second requirement. Finally, we note that features that incorporate temporal

information are particulary suited for use with a special case of HMMs, which will be discussed in the next chapter.

3.5. DESIGN OF THE SYMBOL SET

Summarizing the previous sections, we see that the encoding method of the handwriting was discussed. Some requirements on the feature set were also formulated. In this section, both will be combined in the design of the feature set.

The encoding method described previously results in a time sequence of directional codevectors. We start with an investigation of angular curve description as defined for use with Generalized Fourier Descriptors (GFDs) [8] for three reasons:

1. Angular descriptions (angular information versus time or versus position along the curve) reduce the two-dimensional (x,y) format to a one-dimensional format.
2. Angular information is by nature independent of translation.
3. The description according to GFDs use a normalization procedure.

3.5.1. Cumulated angular differences

First, $\Theta(s)$ is defined as the cumulated angular difference of the angular direction of the curve as a function of the distance s along the curve. $\theta(0)$ is defined as the absolute starting direction. Next, a function $\phi(s)$ is defined:

$$\phi(s) \triangleq \theta(s) - \theta(0) \tag{3-1}$$

Let L denote the total length of the curve, then s is normalized by $t \triangleq s/L$, $0 \leq t \leq 1$, and ϕ can be expressed as:

The previous definitions hold for arbitrary continuous open or closed curves. We note that due to the subtraction of the starting angle, $\phi(s)$ is independent of rotation, and

$$\phi(t)=\phi(\frac{s}{L})=\theta(\frac{s}{L})-\theta(0) \quad , \quad 0\leq t\leq 1 \tag{3-2}$$

because of the normalization on $0\leq t\leq 1$, the description is independent of scaling along the trajectory of the curve. Thus, equally shaped drawings that differ only in size will result in identical $\phi(t)$ functions. For instance extracting a feature vector from $\phi(t)$ by Vector Quantization therefore seems a good way of generating symbols. Unfortunately, this description has some drawbacks, which will be discussed next.

3.5.2. Drawbacks of cumulated angular differences.

The main disadvantage of using cumulated angular differences comes from the cumulative nature of the description. The description is independent of rotation, but only for similar shapes. A difference in the starting angle will affect the complete curve description. Fig. 3-7 shows this effect: shapes that look very similar, but have a different starting angle, result in very different symbol vectors.

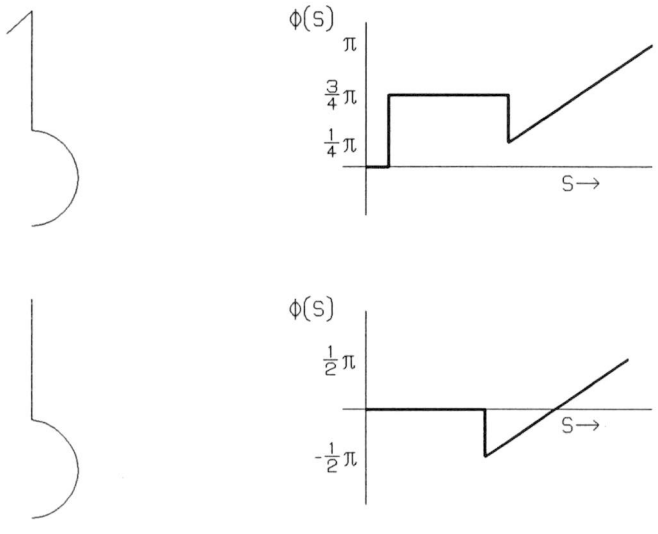


Fig. 3-7: Sensitivity for hooks of cumulated description.

Clearly, this kind of sensitivity is undesired when describing handwriting, since handwriting often shows this variability in starting angle (also known as 'hooks'). Hooks

are due to inaccuracies in pen-down detection and to rapid or erratic motion in placing the stylus on, or lifting it off the tablet. Since the effect of a hook on rotation-independency is fatal, dehooking as a preprocess is an absolute necessity. Fortunately, several 'dehooking'-algorithms exist to solve this problem.

However, at 'retrace points', i.e. points where the pen retraces (usually done to avoid lifting the pen), again large differences in the descriptions of roughly similar shaped characters can occur. The retrace can be approached in a clockwise or a counterclockwise direction, resulting in a positive or negative sign of the angular difference to be added. Again, due to cumulative nature, these differences have effect on the rest of the curve description.

Finally, using the method described above leads to a large dynamic range for the symbols. For instance the character 'm' can have an range of $(0,+6\pi)$ (assuming the clockwise direction to be positive), while the 'd' can have a range of $(0,-4\pi)$. The dynamic range for quantizing an alphabet would be approx. (-20,20) (assuming simple uniform quantization). In order not to loose discriminant power, this would lead to a large number of quantization levels, which makes this description computational unattractive.

3.5.3. Absolute angular symbols

The main disadvantage of the method described in the previous section was the cumulative nature of the description. If we maintain the normalization along the curve, the independency of scaling is preserved. Further, using absolute angles instead of cumulative angular differences reduces the dynamic range to $[0,2\pi)$. The only problem left is the rotation-dependency.

This rotation-dependency can be tackled in various ways. Using differential angles rather than absolute angles would remove the rotation-dependency. However, this method has shown to perform badly in combination with the proposed normalization procedure. Further, a dehooking algorithm would still be necessary.

Another method would be to align every character according to for instance a long straight-line piece in the character [6]. When the angle does not bend over a predefined length, a straight-line piece is assumed and the whole character is aligned. This alignment procedure, however, probably causes problems with characters that do not contain a straight-line section, like 's' or 'c'.

Yet another approach is subtraction of the starting angle combined with a 'dehooking' algorithm. This method is used in this work

All the restrictions of the curve description are removed now, and the vector quantizer that generates the symbol sequence can be designed.

3.6. VECTOR QUANTIZER DESIGN

The vector quantization process mentioned in the previous section has the following two design parameters that need to be chosen:

1. the size of the feature vector, i.e. the length of the observation sequence.
2. the number of quantization levels per entry.

Note that in our case, the VQ actually is a scalar quantizer. Because the quantized symbols are contained in a feature vector, we shall still refer to the quantizer as a vector quantizer.

3.6.1. Feature vector size and normalization procedure

The first problem is to choose the length of the feature vector. It is important to notice in this respect that $\phi(t)$ is not continuous, but a piecewise linear function since the character is already sampled by the tablet. In order to generate an observation sequence of equal length for each sampled character, we cannot simply leave out every i^{th} character, since the samples are not equidistant. Therefore, the interval on the s-axis $0 \leq s \leq L$ is divided into K equidistant steps. For each k, $k=1,2,...,K$, $\phi[k]$ is obtained from $\phi(s)$ using linear interpolation. Note that linear interpolation corresponds to a first-order

reconstruction filter: in spatial domain, this means that a curve is reconstructed by circle segments.

Experiments have shown that with handwriting of normal size and using frame 2 in Grafivision mode, the most complex character consists of at most 75 (x,y) coordinates. Since the interpolation mentioned previously was only meant to normalize 'smaller' characters, i.e. characters that consist of less (x,y) datapoints, this value can be seen as an upper bound for the feature vector length. In all experiments, an observation sequence length of 64 is used.

3.6.2. Number of symbols

The second choice, however, is a lot harder to make, since it involves the trade-off between quantization distortion and computational effort. In this respect should be considered how large the quantization distortion may be before it starts affecting the recognition performance. Therefore, we formulate the following requirement on the VQ accuracy:

The difference between the VQ-representation and its original may not be larger than the average intraclass variability according to some distance criterion.

The distance measure will be defined as a relative error measure. Consider a sampled character, sampled with N discrete (x,y) coordinates. Then we have N-1 discrete $\phi(s_n)$ values, n=1,2,...,N-1. These values are interconnected using linear interpolation, resulting in a piecewise-linear function $\phi(s)$. Next, the s-axis is divided into K equidistant sections. Let L denote the total length of the curve, then for each k (k=1,2,...,K) $\phi[k]$ is determined (size normalisation):

$$\phi[k]=\phi(\frac{k \bullet s}{L}) \tag{3-3}$$

Each value of ϕ[k] is quantized and thus mapped into one of M quantization levels, resulting in ϕ^*[k], the observable symbols:

$$\phi^*=Q\ [\phi(\frac{k \bullet s}{L})]=O_k \tag{3-4}$$

We now define the relative quantization error $\epsilon_{quantization}$ as

$$\epsilon_{quantization} \triangleq \frac{\sum_{k=1}^{K} \|\phi[k]-\phi^*[k]\|}{\sum_{k=1}^{K} \phi[k]} \tag{3-5}$$

In a similar fashion we can formulate an measure for the intraclass variability. Consider two characters, both belonging to the same class. Let ϕ_1[k] and ϕ_2[k] be the normalized description of character 1, and character 2, respectively. We can define a measure for the intraclass variability $\epsilon_{intraclass}$ as

$$\epsilon_{intraclass} \triangleq \frac{\sum_{k=1}^{K} \|\phi_1[k]-\phi_2[k]\|}{\sum_{k=1}^{K} \phi_1[k]} \tag{3-6}$$

Using the previous definitions, $\epsilon_{quantization}$ was computed for different numbers of quantization levels. The result of the computation is shown in Fig. 3-8.

Also shown in Fig. 3-8 is $\epsilon_{intraclass}$ (the dotted line) for two very similar characters.

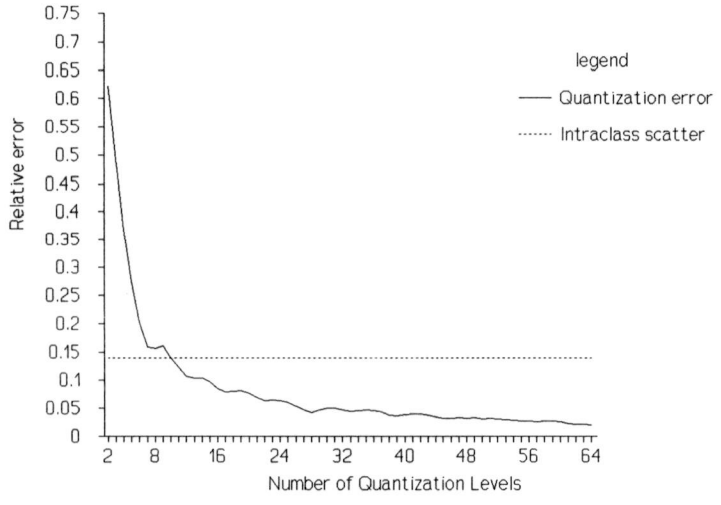


Fig. 3-8: Quantization distortion and intraclass scatter.

Intraclass scatter is highly dependent of the author, the mental state of the author, the time between the writing of the specimen, and so on. The two characters used for determination of the intraclass scatter were written by the same person, shortly after each other. Therefore the external conditions are the same for both characters, and we may assume that the internal conditions (i.e. the writer himself) also remained constant. The indicated value for $\epsilon_{\text{intraclass}}$ may therefore be seen as a lower bound for the intraclass scatter.

Finally, to determine the number of quantization levels, we assert that 32 quantization levels is an upper bound for the number of symbols.

3.7. PREPROCESSING

One aspect of the classification system has not been discussed yet, since it is somewhat dependent of the type of curve description used: the preprocessing of the curve. This step is carried out directly after the sampling by the tablet. Here, two preprocessing

techniques are discussed: a dehooking algorithm, and a method to treat within-character pen-lifts.

3.7.1. Dehooking algorithm

The dehooking algorithm is very simple, but effective. When within a predefined distance s_{hook} from the starting point of the curve the difference between two consecutive angular directions exceeds a threshold ξ, all samples before this point are eliminated. Expressed in ϕ this criterion can be written as

$$\|\phi(s_n)-\phi(s_{n-1})\| \geq \xi \quad , \quad 0 \leq s \leq s_{hook} \tag{3-7}$$

Clearly, s_{hook} is chosen small relative to the total length of the curve, and ξ should be chosen large enough. Setting values for s_{hook} and ξ to $s_{hook}=0.05 \cdot L$ and $\xi=½\pi$ yielded good results.

3.7.2. Within-character pen-up

Some characters, like the 'j' or the 't' are not written continuously but have an extra pen-up. A method has to be found to deal with these pen-ups.

One way could be by simply ignoring the pen-up, and connecting the pen-up point with the pen-down point by a straight line. This method is very easy to implement, since characters are encoded as (x,y) coordinate files.

However, examining the handwritten script closer leads to a more uniform way of coding. Consider for instance a 'j'. Clearly, the pen is lifted in order to place the dot on top of the character. Linear interpolation would be a way of encoding the whole character, but nothing more than that. Consider now an capital 'R'. some people lift the pen to write the character, while others will retrace the pen without lifting it from the paper (see Fig. 3-9).

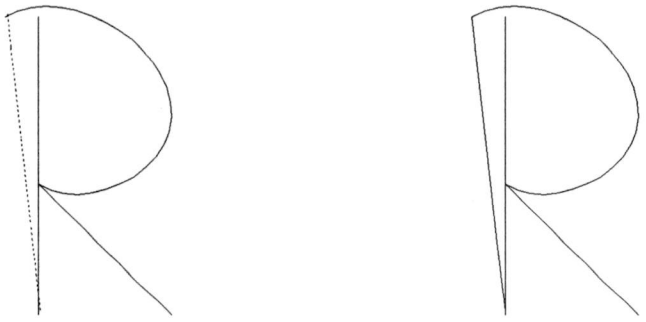


Fig. 3-9: Capital 'R' with (left) and without (right) penlift.

Clearly, linear interpolation here would be a considerable more effective way of encoding, since a greater unity is achieved, and thus the intraclass variability is reduced.

Therefore, we the following algorithm is used to treat pen-ups: A extra symbol ('dot') is added to the feature vector. When the shape succeeding a pen-up is a dot (which is easy detectable), this symbol is generated. When the succeeding shape is not a dot, the previously mentioned straight-line interpolation is used.
Finally, Fig. 3-10 shows a flowchart of the complete symbol generation procedure.

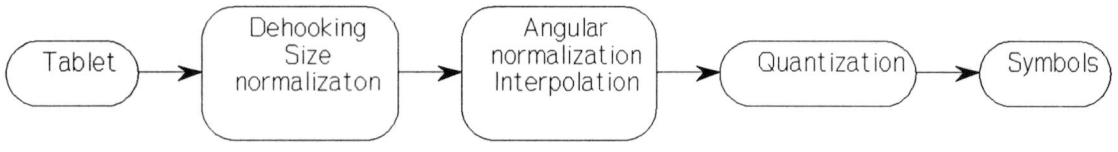


Fig. 3-10: Flowchart preprocessing procedure.

Some examples of used characters can be found in Appendix B, while in Appendix C a typical observation sequence is shown.

Chapter 4.

The state transition matrix; HMM structure

4.1. INTRODUCTION

The previous chapter discussed the symbols of the HMM system. The second important parameter that needs to be estimated in the application of HMMs is the state transition matrix **A**. Our goal here is to investigate what type of model (i.e. number of states and type of constraints on state transitions) best describes the underlying stochastic process, which is in our case handwriting. No answer can be given to this question in advance, because the parameter estimation algorithms discussed in chapter 2 find locally optimal models. As we shall see, even under ideal conditions, i.e. the training data are generated from a known HMM using for instance a Monte Carlo simulation, the estimated models may contain artifacts and may not faithfully represent the inherent structure of the data. Therefore, great caution and empirical validation is required in the use of this technique.

4.2. INFLUENCE OF INITIAL ESTIMATES

The Baum-Welch reestimation procedures require initial estimates for the parameters **A** and **B**. Since the Baum-Welch algorithm finds a local maximum, this may vary under the initial estimates. An important question is how large the influence of this effect is on overall recognition performance. By selecting R random starting estimates for **A** and **B**, and optimizing a model for each initial estimate, this effect can be investigated.

Basically, there are two choices to reduce the influence of initial estimates: use multiple HMMs per character, followed by selecting the best one, or average the R models. The latter method, as we shall see, requires some care in the implementation.

4.2.1. Model estimation on a Markov Source

We now turn our attention to the testing of the reestimation procedures. Experiments have shown that it is difficult to reliably estimate parameters of a state whose average occupancy is very much smaller than that of the states to which it is connected [1]. These states tended to merge with the preceding or the following state. To test the reestimation procedures on models that should be 'easy' to estimate, i.e. models with high expected average occupancies, a 5-state left-to-right HMM was defined according to [1] with the following structure and state transition probabilities (see Fig. 4-1):

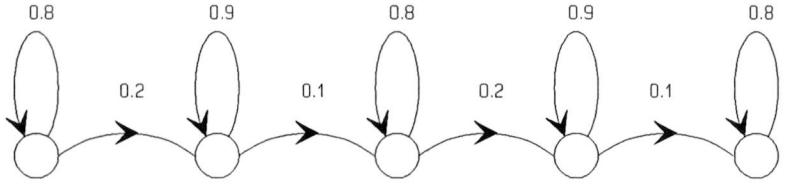


Fig. 4-1: Test left-to-right HMM.

A symbol set size of 9 was chosen with the following symbol generation probabilities:

$$B = \begin{bmatrix} 0.7 & 0.3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.8 & 0.2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.2 & 0.8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.3 & 0.7 \end{bmatrix}$$

Now, 10 observation sequences were generated by a random number generator using the following procedure:

1. Let R be a random number generator whose output is uniform on [0,1]
2. Partition the unit interval proportionally to the π vector, and generate a random number. Select the starting state s_i according to this number, and set $t=1$.
3. Partition the unit interval proportionally to the i^{th} row of **B**. Generate a random number, select a symbol v_k and set $O_t=v_k$
4. Partition the unit interval according to the i^{th} row of **A**. Generate a random number, and select the new state s_j.
5. Increment t. If $t \leq T$ then set $s_i=s_j$, and repeat steps 2-5. Otherwise stop.

Using these observation sequences, a HMM was trained with the Baum-Welch training algorithm several times under different (essentially random) initial estimates.

For some initial estimates, the correct parameters were obtained to within small estimation errors. This is the same result as obtained in [1]. However, for about 50% of the initial estimates, no entirely correct parameter set was obtained. Thus, an important conclusion can be drawn: the influence of initial estimates probably affects the recognition performance, and must be quantified first before the other recognition experiments can take place.

4.2.2. Averaging models

Another way to reduce the influence of initial estimates is to average several models that were obtained under different initial estimates. This will lead to improved model stability. The problem that arises when averaging models is that the local maximum points of P^{BW} are only unique to within a renaming of the states. Two different observation sequences s_i and s_j may be topologically equivalent, but $i \neq j$. One way of circumventing this problem is to use the converged estimated values for one sequence as the initial estimates for the next observation sequence, hoping that the search now is restricted to the neighbourhood of one single local maximum. Unfortunately, this method is not reliable.

A better solution to the problem is to find a renaming of the states that minimizes the distance between the models according to some distance measure. Let $\mathbf{b}_j$ and $\mathbf{b}_j\tilde{}$, $1 \leq j \leq N$, be the rows of two estimates of **B**. Let p(j) be a permutation of the state index j, and let d be some distance measure. Then we seek the permutation p of $(s_1, s_2, ..., s_N)$, such that D is minimized, where D is given by

$$D = \sum_{j=1}^{N} d[b_j \ , \ \tilde{b}_{p(j)}] \qquad \textbf{(4-1)}$$

Denoting the number of states by N, the number of symbols by M, and selecting a quadratic error measure for D yields

$$D = \sum_{j=1}^{N} \sum_{k=1}^{M} \| \ b_{jk} - \tilde{b}_{p(j)k} \ \| \qquad \textbf{(4-2)}$$

Clearly, trying all N! possible permutations is computationally very unattractive. Fortunately, algorithms exist that make the problem manageable. By transforming the problem to a minimum-weight bipartite graph-matching problem on 2N vertices, the problem can be solved using combinatorial optimization algorithms. These tend to grow with the number of states as N^3. In [1], such an algorithm is discussed.

4.3. MODEL STRUCTURE

In a generalized HMM, a transition from any state to any other state is possible. From section 2.6.2. it follows, that a model can easily be constrained by setting reestimation parameters to zero. In this way, a left-to-right model can be constructed. Such a left-to-right model inherently imposes a temporal order to the HMM since lower numbered states account for observations occurring prior to those for higher numbered states, which is the main motivation for the use of this structure. Since we use a curve encoding technique that captures this temporal information, a left-to-right HMM might effectively model this process, however, also generalized model structures will be empirically investigated. Since LTR HMMs have slightly different properties than generalized HMMs, the following sections will discuss some issues concerning left-to-right HMMs.

4.3.1. Left-to-Right HMM characteristics

A left-to-right HMM is characterized by the following properties:

1. The first observation is produced while the HMM is in a distinguished state called the starting state, denoted by s_1.
2. The last observation is generated from the final state, s_N
3. Once the HMM leaves a state, it cannot revisit this state any more.

The properties mentioned above lead to an important observation: using a single, long observation sequence for training of the model makes no sense in this case, since once the final absorbing state s_N is reached, the rest of the sequence provides no further information about earlier states. The appropriate training data for such a model are a set of observation sequences, and the theory explained in section 2.5.2. should be used to estimate the parameters.

4.3.2. Constraining left-to-right HMMs

The properties of an left-to right HMM can be imposed on a model as follows:

1. condition (1) from the previous section will be met by simply setting $\pi=[1,0,...,0]$. Needless to say that π does not require reestimation.
2. The second property can be imposed by setting

$$\beta_T(j)=\begin{cases}1 & \text{for } j=N\\ 0 & \text{otherwise}\end{cases} \qquad \textbf{(4-3)}$$

3. By setting the initial estimate for a_{ij} in the Baum-Welch algorithm to $a_{ij}=0$ for any disallowed state transition the second property is implemented, since once a parameter is set to zero, it will remain zero.

A model structure can be imposed on a model using the previous described methods. This leads to the central question what type of model (model structure and model size) best captures the process of handwriting.

Unfortunately, there appears to be no good theoretical way to choose the number of states needed for a character model, since the states need not be physically related to any single observable phenomenon. Given the symbol generation method from chapter 3, the closest relation between a state and the original character is that a state represents part of the character with a constant direction. This interpretation will be further elaborated.

4.3.3. State interpretation

The introduced relation between HMMs and physical entities lead to the following reasoning:

1. straight-line pieces are modelled by a state with a high self-transition probability, and a corresponding symbol row that is especially sparse.
2. Curved sections of a character are modelled by a state with very low occupancy. The occupance will depend on the curvature of the original.again, the **B** matrix will be especially sparse.

Some characters can be regarded as part of other characters. These characters could simply reach their final state sooner than other characters. Fig. 4-2 shows this effect: for instance a 'c' is part of an 'a', and an 'a' is part of an 'g'. This phenomenon could effectively be modelled by not imposing the constraint that the last symbol must be generated from the final state

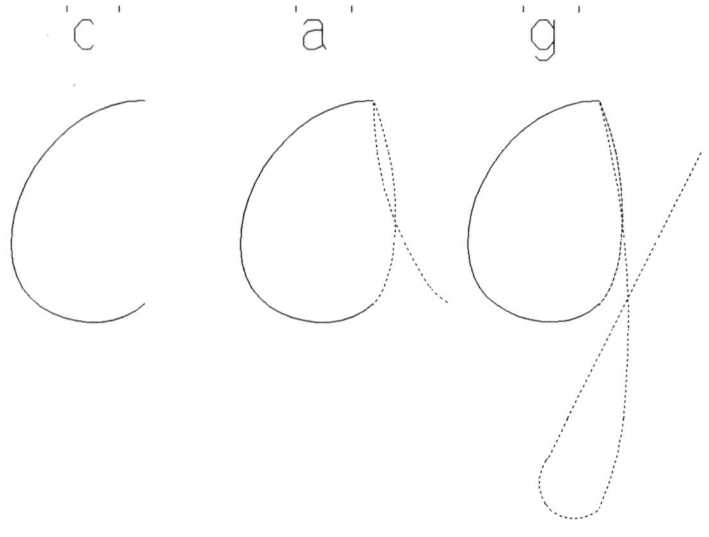


Fig. 4-2: Characters as part of other characters.

According to the previous reasoning, probably large models sizes are required to capture the curved characters correctly.

In any case, the model structure will therefore be subject to an empirical analysis, which is discussed in the next chapter. In this chapter, we proceed with some issues in the implementation of HMMs that affect system performance, and were not mentioned in chapter 2.

4.4. CONVERGENCE BEHAVIOUR

The Baum-Welch reestimation formulas are guaranteed to increase P^{BW} at each iteration, until a critical point is reached from where P^{BW} does not change any more. Experiments by [1] have shown that the estimated models make large improvements to P^{BW} during early iterations, and only slight improvements later. The convergence criterion was set at a relative increment of 1e-7. For this criterion, the second half of the iterations showed no significant improvements to the model any more. Thus, the convergence criterion may be substantially relaxed. However, the following plot of $\log(P^{BW})$ versus number of iterations (see Fig. 4-3) shows that the rate of convergence may have several points of inflexion.

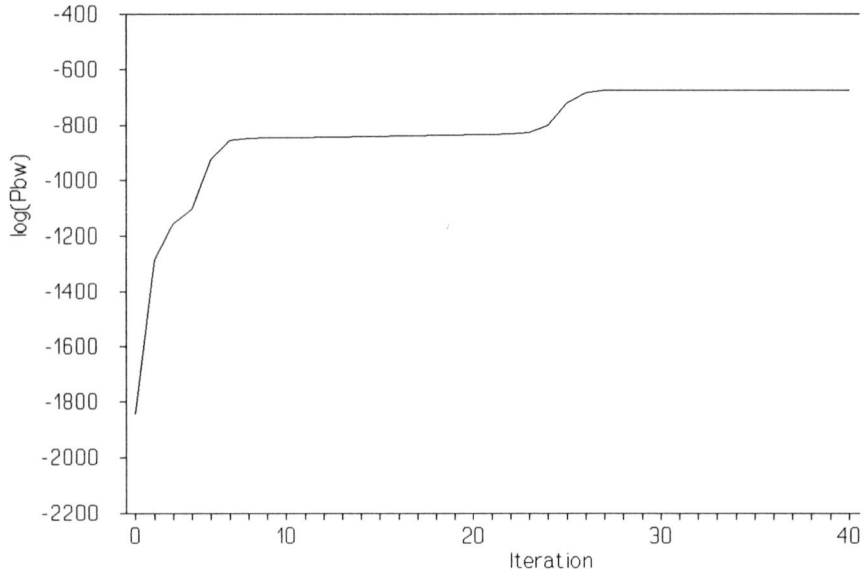


Fig. 4-3: Convergence behaviour.

Therefore, some care is required in the determination of a convergence criterion.

4.4.1. Computation times

It is worth to notice the computational expense when setting for instance a convergence criterion. Basically, the following parameters affect the computational expense during the training of the model:

N,	the number of states
T,	the observation sequence length.
I,	the number of iterations
R,	the number of optimizations per model.

Roughly, the forward-backward procedure requires about $2 \cdot T \cdot N^2$ computations. The Baum-welch reestimation formulas require $T \cdot N^2$ computations for the **A** matrix, and $2T \cdot N$ calculations for the **B** matrix. Denoting the number of iterations by I, then the total number of required computations C is given by eq. 4-4:

$$C = I \cdot T \cdot (3 \cdot N^2 + N) \tag{4-4}$$

Note that the computational effort during training is not dependent of the number of symbols M. So, when it is necessary to choose between either a larger model size or a larger symbol set, the latter deserves priority.

4.5. VITERBI AND BAUM-WELCH ALGORITHMS

Finally, a choice has to be made about the algorithms to use in the experiments. Since the Viterbi algorithm generates a subset of the Baum-Welch procedure (this is the case when the state sequence is unique), only the Baum-Welch procedure will be implemented.

Chapter 5.

Empirical investigation; optimum parameters

5.1. INTRODUCTION

All the issues of practical implementation of HMMs have been discussed in the previous chapters. Now, the performance of HMMs as a recognizer can be tested. First, however, we have to determine how each parameter individually affects performance of the whole system. The basic scheme for these experiments is simple: we vary one parameter (for instance the number of states), and see how the recognition performance is affected. So, it is assumed that the model parameters are independent of each other, although this assumption probably is not entirely justified. However, an overall-approach would lead to too many experiments to carry out in a limited time. For computational reasons, only a 10-word vocabulary will be used at this stage (typically characters 'a'-'j').

Maximum-Likelihood classification principally is able to give the most likely character for each observation sequence, all the way down to the least likely character. This property makes ML classification well-suited for a hybrid recognition system, since when the distance between the most likely character and 'one-less' likely character falls below a threshold, a secondary method can be chosen to decide between both characters (in fact, this method is not limited to two characters). However, throughout this section, we shall consider every misinterpreted character as an error, thus determining the influence of each individual parameter.

After these experiments, for each parameter a compromise is found between computation times and recognition accuracy (see Fig. 5-1).

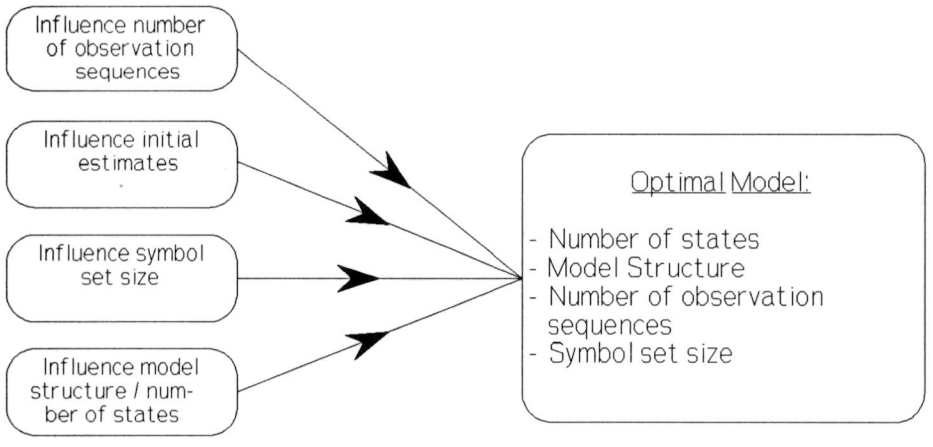


Fig. 5-1: Experiments on individual parameters.

With this set of parameters, a second stage of experiments is entered in chapter 6: now, using the optimum model parameters, writer-dependency is subject to investigations. (see Fig. 5-2). An error analysis is given here, too.

5.2. INITIAL PROBABILITIES

As we saw in chapter 4, the estimation accuracy for the trained model will depend on the initial values for the **A** and **B** parameters. The question is how this effect affects the recognition performance, and to what extent. First, the other parameters that are fixed are chosen empirically (see Fig. 5-2):

Next, 15 HMMs per character were trained, each under different (essentially random) initial estimates. Thus, we have 15 different sets of models, denoted by R1-R15. Using

States (N)	8
Quantization Levels (Q)	16
Symbols (M)	17
Constraints on model structure	Only left-to-right transitions allowed
Observation sequence length	64
Number of training sequences (K)	25
Number of test sequences	25

Fig. 5-2: Model parameter values.

the 25 test sequences per character, for each set of models, the individual error rate was determined, see Fig. 5-3.

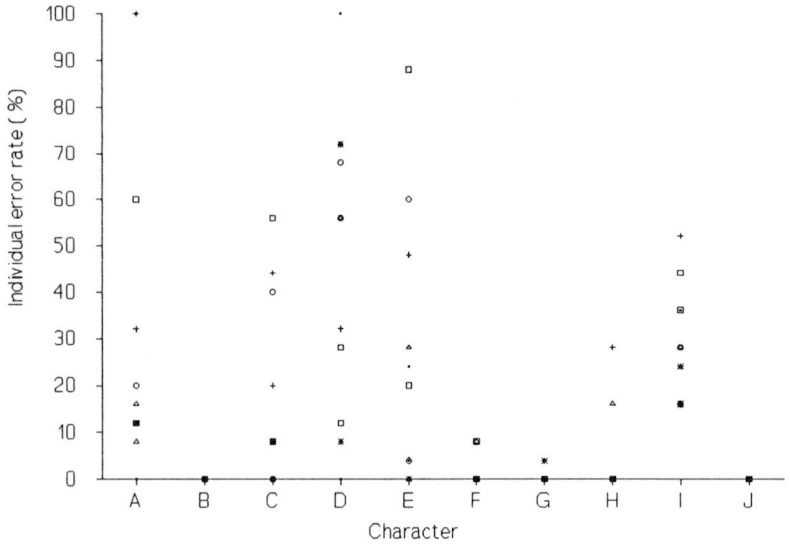


Fig. 5-3: Individual error rate under different initial estimates.

Next, the overall error rate, i.e. the average error rate over all characters, was determined. It is shown in Fig. 5-5.

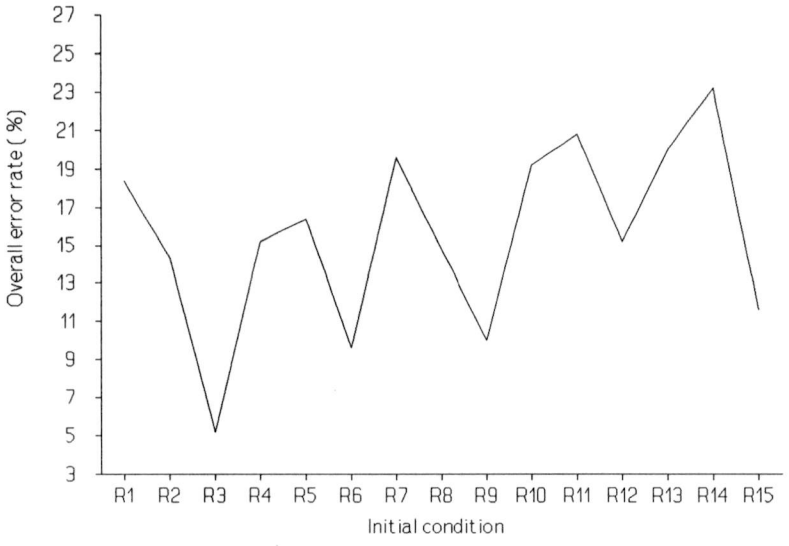


Fig. 5-4: Overall error rate under different initial estimates (the periodicity is fortuitous).

We see that some individual characters show a very stable behaviour. notice for instance the 'j' and the 'b', which were not misinterpreted under any condition. However, other characters show a severe fluctuation under different initial values. This fluctuation can also be observed in the overall error rate, although less strong because of an averaging effect: individual fluctuations may average out in the overall performance.

These results justify the assumption from chapter 4 that the influence of initial estimates can also be observed in the recognition performance. Two methods were introduced for dealing with this problem: multiple training under different initial estimates of each individual model, followed by selecting the model that yields the highest P^{BW}, or averaging models.

One conclusion can already be drawn at this time: the influence of initial estimates is too high to be ignored. This implies that the method that will be used to reduce this influence must also be used in further experiments that will be conducted, since otherwise the results will not be reliable.

5.2.1. Selecting optimum models

For each of the 15 HMMs per character, P^{BW} was calculated over the 25 training sequences. Hereafter, the models with the highest P^{BW} were selected. Fig. 5-5 shows the individual percentual error rates that were obtained with the sets that contained these models.

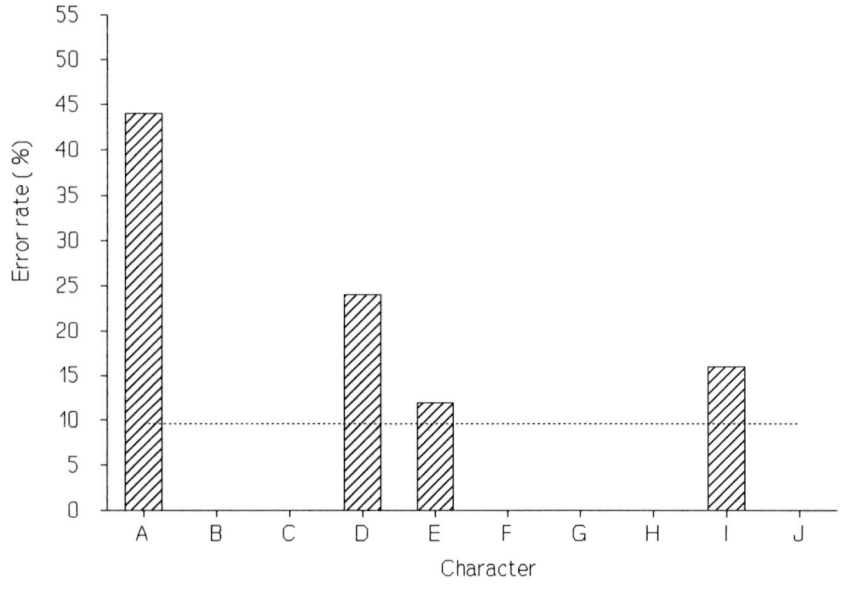


Fig. 5-5: Individual error rates for models with P^{BW} as optimality criterion.

Also shown in Fig. 5-5 is the overall error rate (shown by the dotted line). At this time, comparing these results with the results obtained in the previous section, we can state that using P^{BW} as a optimality criterion for selecting the best models leads to a considerably lower overall error rate, although the lowest error rate was obtained for initial estimate R3 (see Fig. 5-4), and not for the set containing the optimum models. This is because of the previously mentioned relativity of models to each other.

5.2.2. Averaging models

The second method introduced in chapter 4 to reduce the influence of initial estimates and to improve model stability is to average models obtained under different initial

estimates. The 15 obtained models per character from the previous section were averaged. With the resulting set of models, the recognition test was done. Fig. 5-6 shows the individual error rates (in percents), as well as the overall error rate, which again is indicated by the dotted line

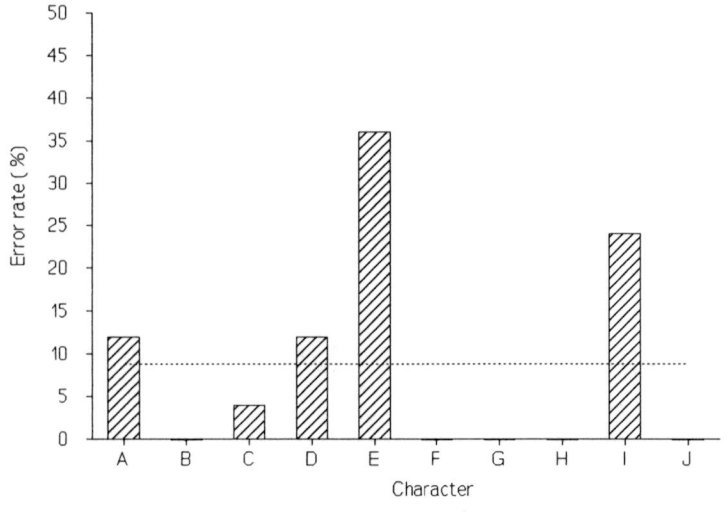


Fig. 5-6: Error rates for averaged models.

As can be seen, the overall error rate does not differ much from the error rate obtained with optimum models (compare 9.6% to 8.8%), although the individual error rates show some differences. This is again due to the averaging effect of the overall error rate. Concluding we can say that both methods work equally well, so the optimum model method is chosen for further experiments for computational reasons.

5.2.3. Generating initial estimates

In the previous section, the influence of initial estimates was investigated by generating initial estimates randomly. An alternative and more logical method is to divide the unit interval of each variable into μ sections, and to optimize over all possible combinations.

However, if a LTR model has N states and M symbols, then the number of possible combinations is given by

$$C=\mu^{\left(\frac{N^2+N}{2}+N\bullet M\right)} \qquad \textbf{(5-1)}$$

Setting μ=3, N=4 and M=16 gives $2\cdot10^{35}$ combinations. Clearly, not an very attractive prospect, especially since the number of optima probably is a lot smaller.

Therefore, we shall use the method of random initial estimates. We have to bear in mind that all the results that are obtained with this method may show a statistical fluctuation.

5.3. NUMBER OF OBSERVATION SEQUENCES

The next factor of interest is the number of observation sequences that are used to train the models. In previous experiments, a value of K=25 observation sequences was used. Here, we vary the number of observation sequences as follows:

$$K=\{5,10,25,50,100\}$$

Fig. 5-7 shows the overall error rate as a function of the number of training sequences.

As expected, we notice a decreasing error rate as the number of training sequences increases. However, for values of K≥25, the error rate does not decrease anymore, and for K≥50 the error rate even drastically increases. This is highly against our expectations, since one would think that increasing the number of observation sequences leads to more accurate models. To understand this phenomenon, we recall the definition of P^{BW} when training under multiple observation sequences:

The observation sequences are treated as being independent of each other, and we optimize the product of the individual P^{BW}'s. So, the more observation sequences, the

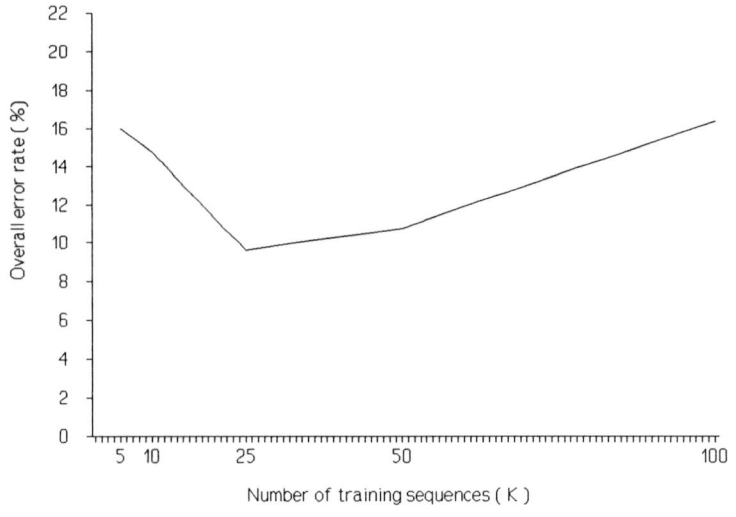


Fig. 5-7: Overall error rate versus number of training sequences.

$$P=\prod_{k=1}^{K} Pr(O^{(k)} \mid M) = \prod_{k=1}^{K} P_k^{BW}$$

more complex the likelihood function becomes, and the more difficult it is to find a strong local maximum (i.e. a maximum near the global maximum) using the previously described procedure. In other words: the more observation sequences, the more optimizations per character are necessary to find a satisfactory strong maximum. Apparently, the intraclass scatter for the given number of symbols is small enough to be captured by 25 training sequences.

5.4. MODEL SIZE

Keeping the results and assumptions from the previous section in mind, we now investigate the size of the model, i.e. the number of states. The number of states was varied as follows:

$$N=\{4,5,6,7,8\}$$

The other parameters were again kept the same as in the previous sections. 30 optimizations per model were used to obtain optimum models. The results of the recognition experiment is shown in Fig. 5-8.

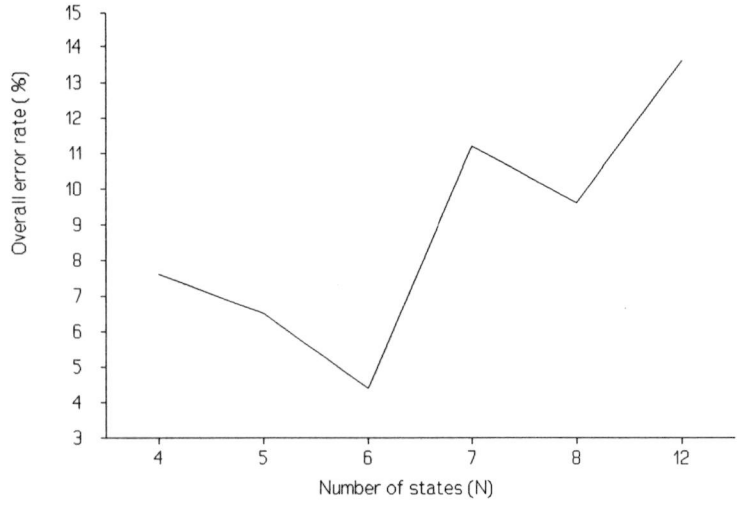


Fig. 5-8: Recognition error as a function of the number of states N.

Again, this graph is somewhat against our expectations: the recognition error first decreases until at N=6 an error rate of 4.4% is achieved. Then the error increases again with the number of states.

The previous section showed that good performance is highly dependent of the distance from a local maximum from the global maximum. We see that increasing the number of states by one to a total of N states, increases the total number of variables by N·M+2·N-1. So, the number of possible combinations for the initial values increases dramatically with the number of states. 30 optimizations per model simply is highly inadequate to find a strong maximum model.

5.5. SYMBOL SET SIZE

The next parameter of influence on the system performance is the number of quantization levels, i.e. the size of the symbol set. We can say that increasing the symbol set also increases the number of required training sequences, simply because more symbols need to be observed. In the following experiment, the number of quantization levels was varied from 8 to 24, while the rest of the parameters were kept the same as in the previous experiment. The model size, however, was reduced from N=8 to N=4, in order not to have to increase the number of optimizations per model. Fig. 5-9 shows the results of the recognition experiment.

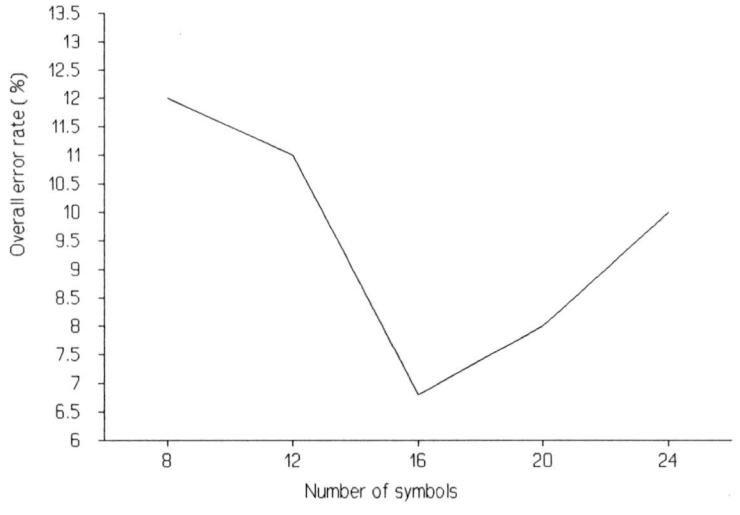


Fig. 5-9: Recognition error versus number of quantization levels.

The graph shows a descending error rate as the number of quantization levels increases. For M≥16 the error increases again, probably due to the inadequate training data problem.

5.6. MODEL STRUCTURE

The model structure is also part of the investigation. Basically, 4 model structures will be examined (see Fig. 5-10):

1) Generalized: transitions from any state to any other state possible.
2) LTR-1: only left-to-right transitions allowed.
3) LTR-2: only left-to-right transitions allowed, no skips of states.
4) LTR-3: only left-to-right transitions allowed, only single skips of states allowed.

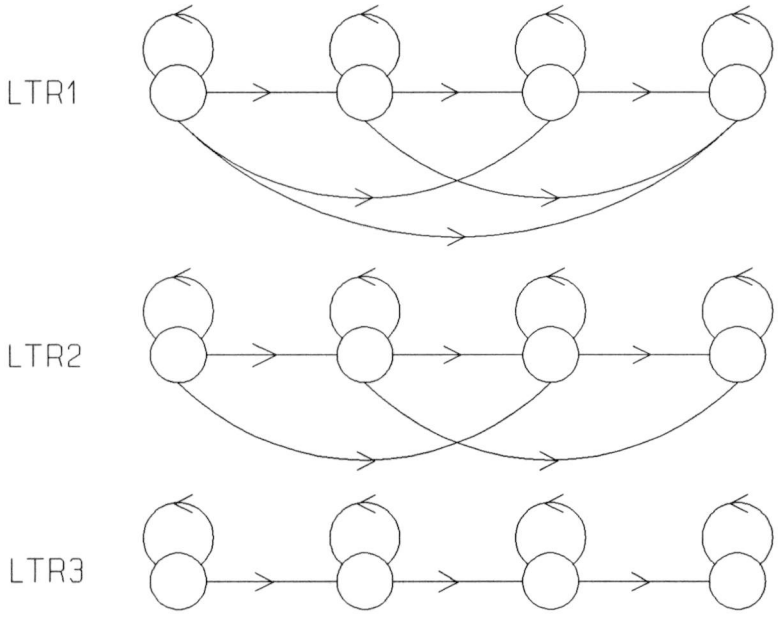


Fig. 5-10: 3 Left-to-right HMM structures.

For each model structure, the recognition error was determined. Again, 4 states and 16 symbols were used. The results are shown in Fig. 5-11.

Clearly, the left-to-right models score considerably better than the generalized model. The temporal order that is imposed by a LTR model combines well with the coding method. Within the LTR models, the model that has no further constraints on the

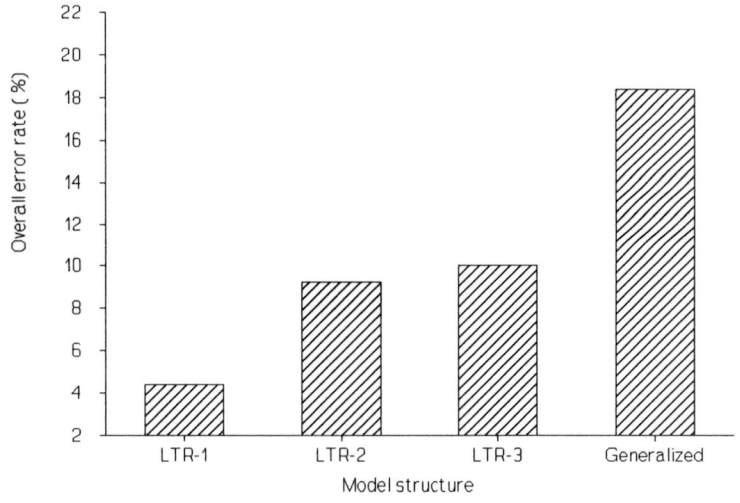


Fig. 5-11: Overall error rate versus model structure.

structure, i.e. model LTR-1, shows the best performance, probably because it has the largest capacity to capture different structures in one initial model. Since models LTR-2 and LTR-3 are subsets of model LTR-1, the results obtained are quite reasonable.

5.7. OPTIMUM PARAMETERS

Several important conclusions concerning the application of HMMs can be drawn, based on the previous experiments. The most important conclusion is that multiple training of each character, followed by selecting the one that yields the highest P^{BW} is an absolute necessity for good performance. Training on more observation sequences principally captures a greater variability, however, the likelihood function also becomes far more complex. This increases the number of required optimizations per character dramatically. So, if the intraclass variability is not too large, it pays to optimize on fewer observation sequences. Given the variability in the training data used in the experiments (see also Appendix C), about 25 observation sequences yielded the best performance.

The same reasoning applies to the model size. Larger models can capture more characteristics of the underlying process, however, at the cost of greatly increasing computational effort. Especially from the multimedia point of view, where the terminals are not powerful workstations, this is very unattractive. A model size of N=6 combined with 16 quantization levels is the best compromise between training effort and recognition accuracy, especially if we quantify the computation times mentioned in section 4.4.1.

5.7.1. Quantification computation times

For the given parameters, the time required to recognize a character is less than one second (on an AT-type 486 computer). The time required to train an complete alphabet is about 24 hours on the same machine. Increasing the model size implies that at the current state of technology, training cannot be done on PC's. The recognition time will also increase to a unpractical value.

Chapter 6.

Writer-dependency experiments

6.1. INTRODUCTION

In the previous chapter, all the parameters that affect recognition performance were investigated. An optimum set of parameters was determined. In this chapter, we use these optimum parameters to investigate the writer-dependency of a HMM-based system (see Fig. 6-1). The kind of errors the system makes is also subject to investigation.

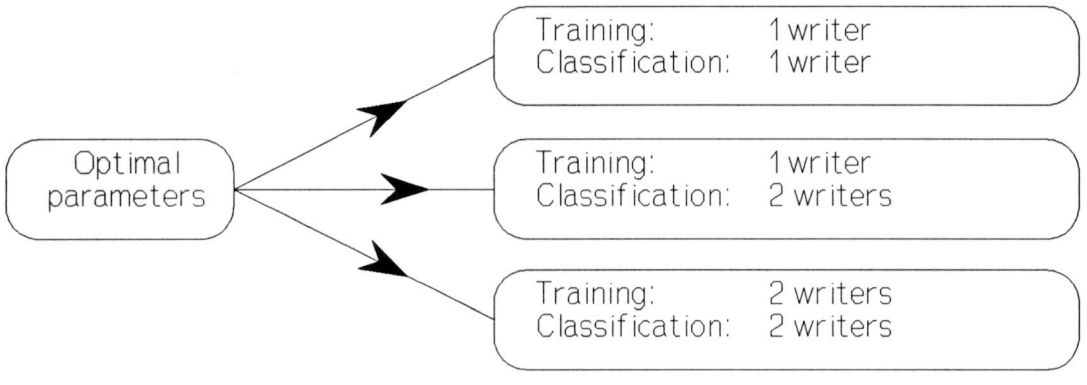


Fig. 6-1: Writer dependency experiments.

The vocabulary will not be restricted to the characters 'a'-'j', but will consist of the complete English alphabet.

6.2. SINGLE-USER SYSTEM

The first experiment evaluates the performance of a system trained on one writer, and used by one writer. A vocabulary consisting of characters 'a' to 'z' was trained. With the models obtained by this procedure, both the individual and the overall recognition errors was determined. Fig. 6-2 shows the errors for individual characters. The overall performance is also shown, indicated by the dotted line.

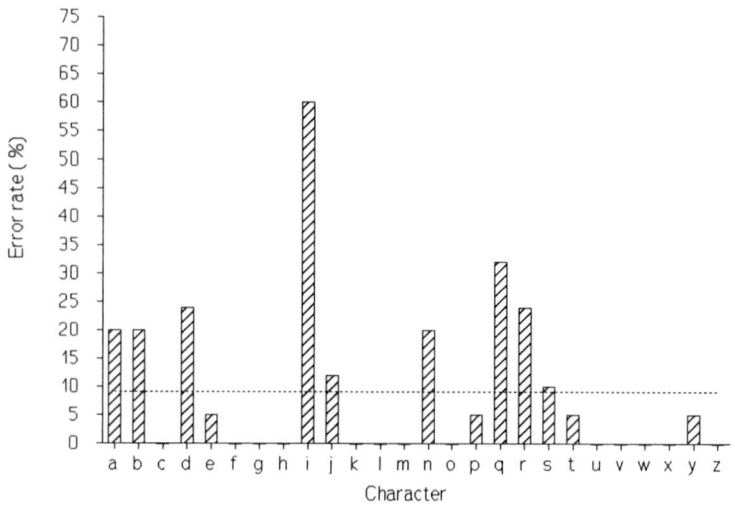


Fig. 6-2: Individual and overall errors characters 'a'-'z'.

As can be seen, an overall recognition rate of 90.9% is achieved. Next, we shall turn our attention to the kind of errors the system makes in order to improve the performance further.

6.2.1. Error analysis

Examining fig. 6-2 closer learns that the 'i' provides a considerable contribution to the recognition error. So, despite of the extra symbol 'dot', it is often misinterpreted. Probably, the generation of one symbol on an observation sequence of length T=64 is

not enough to direct the optimization procedures in a specific direction. Or, other symbols for the final state are dominant over the dot-symbol.

To investigate if the recognition rate can be improved, the number of dot-symbols per observation sequence was varied from 1 to 8. Fig. 6-3 shows the recognition error for character 'i'.

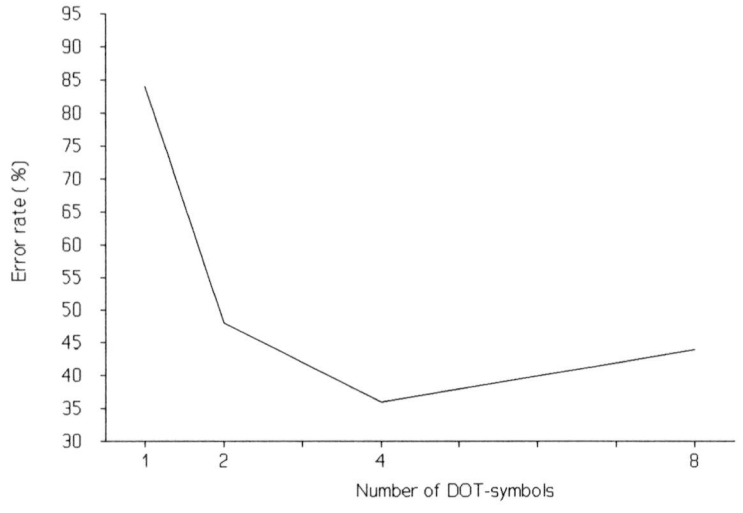


Fig. 6-3: Recognition error versus number of 'dot'-symbols (character 'i').

As can be seen, the recognition error can be reduced by generating more than one dot symbol. The error rises again for 8 dot symbols. This is because of the fixed length of the observation sequences. Increasing the number of dot-symbols inherently decreases the number of symbols left for curve description.

Repeating the recognition experiment for the whole curve leads to a recognition error for 'i' and 'j' of 12% and 0% respectively; quite an improvement. The overall error rate now obtained is 6.6%. Extending the vocabulary with the numerals '1'-'9' slightly increases the recognition error to 7.5%

In order to understand the limitations of a HMM-based recognizer, we now investigate how characters are misinterpreted. Table 6-1 shows for every character with more than two errors per 25 samples the error that was made.

Table 6-1: Types of errors.

Character:	a	b	d	i	n	q	r	s
Interpreted as:	d	p	a	w	m	g	n	g

As can be seen, the 'a' was often misrecognized as an 'd', and the 'b' as an 'p'. Keeping in mind that a 'd' is written exactly like an 'a', except for the higher extension of the vertical stroke, this error is quite logical. The same reasoning applies to the 'b'-'p' and 'q'-'g' couples.

Apparently, the model is unable to capture the subtle differences, probably because the number of observations of these differences are too small compared to the total observation sequence length. Therefore, larger models, with inherently a smaller average state occupancy, may improve the recognition performance, however, again at the cost of increased computational effort.

Applying ad-hoc decision rules for ambiguous characters is then a better way of increasing the performance, although this method also suffers from disadvantages [12]. Changing the writing style for badly recognizable characters, thus increasing the distance between the squares form section 3.4.1., is also a method to increase performance.

6.3. MULTIPLE USERS SYSTEM

The performance is now evaluated for a system that has multiple users. The experiments as shown in Fig. 6-1 are carried out: First, the performance is calculated for a system trained on one writer, and used by 2 writers. Next, models were obtained by training on

sequences from the two writers. The recognition errors for this system, used by the two writers on which it was trained, were also obtained. The error rates for these systems are shown in Table 6-2.

Table 6-2: Recognition errors for various systems.

TRAINING:	USERS:	RECOGNITION ERROR:
1 writer	1 writer	6.6%
1 writer	2 writers	11.41%
2 writers	2 writers	10.96%

The results are clear: Due to the differences in writing style is it necessary that the system is trained on its user. The best performance is achieved for a single-user system, trained on its user. The error rate for the two-user, two-training system is lower than the error rate for the two-user, single-training system, however, the error rate degrades highly as the number of users increases.

By recalling the concept introduced in section 3.4.1., this phenomenon can be explained:

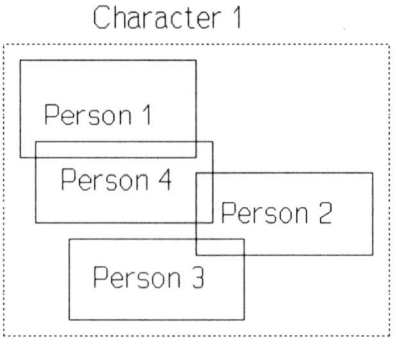


Fig. 6-4: Increased intraclass scatter with multiple users.

Due to the multiple users, the intraclass scatter increases. Thus, the rectangles from Fig. 3-5 and 3-6 become larger, and the class overlap will also increase (see also Fig. 6-4). So, the recognition performance decreases.

Chapter 7.

HMMs and signature verification

7.1. INTRODUCTION

For several reasons, signature verification is not a trivial pattern recognition problem. Compared to character recognition, several non-technical issues play an important role, such as [7]:

1. How to integrate a system in a actual working environment. (e.g. to control access, to verify cheques, to authenticate a document, and so on).
2. The impact of these control and security systems on private life and personal data. This can be viewed in relation with the global discussion about telecommunications and privacy.
3. The legal problems related to the use of these systems.

Although point 1 mentioned above also applies to character recognition, it is clear that both the legal and the social impact of signature verification systems will be reflected in heavy constraints on system design and system performance of a verification system. The following section will discuss the global signature verification system concept.

7.2. CONCEPTS IN SIGNATURE VERIFICATION

To discuss the general system description for a signature verification system, it is useful to slightly modify the concepts introduced in chapter 3. Ideally, signature verification can be presented as a two-class partitioning problem. Denote Ω^i by the set of all signatures of writer i. Ideally, Ω^i can be binary separated into two classes [7]:

1. ω_1^i, the class of with genuine signatures, and
2. ω_2^i, the class with forgeries, which can be split up into [7]:
    1. 'simple forgery': The forger makes no attempt to simulate or trace a genuine signature.
    2. 'substitution forgery': The forger uses his/her own signature instead of the real signature.
    3. 'skilled forgery': The forger tries to imitate the genuine signature as much as possible.

Signing is a relatively stable process; however, successive trials will not lead to identical characteristics. The process is dependent of for instance practical conditions or mental state of the writer.

7.2.1. Two types of variability

The two types of variability from chapter 4 can be clearly distinguished, however, this time in a slightly different context:

1. Intraclass or intrapersonal variability, i.e. the variation observed within a class of genuine signature specimens of one person.
2. Interclass or interpersonal variability, i.e. the variation between genuine signature classes from different persons.

Again, intraclass scatter should be as low as possible, and interclass scatter as high as possible for class separation. Ideally, the classes are well-separated, and a binary decision threshold T_0 can separate the two classes (see Fig. 7-1).

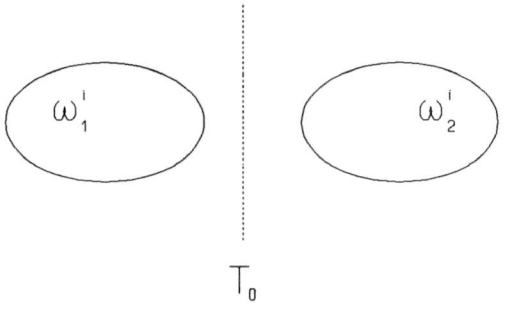


Fig. 7-1: Ideal class separation.

In practice, however, the classes show overlap (see Fig. 7-2), and two types of errors are introduced: type I, False Rejection (FR) and type II, False Acceptance (FA).

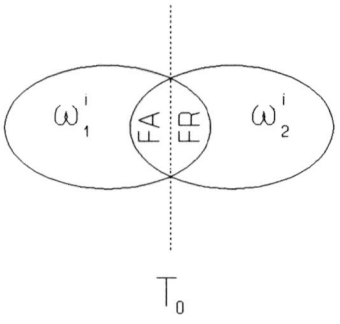


Fig. 7-2: Practical class overlap.

7.3. SIGNATURE GENERATION

We now turn our attention to the generation of a signature by a human. Fig. 7-3 shows the general model of this process.

According to this model [7], some central nervous mechanism within the brain generates impulses that activate through the nerve network the proper muscles in a predetermined order. This results in a trace of the pen tip on the paper. However, it is not yet determined where the boundary stands between fast handwriting and slower handwriting, i.e. where position and visual feedback apply.

From this model, a signature can be viewed as the output

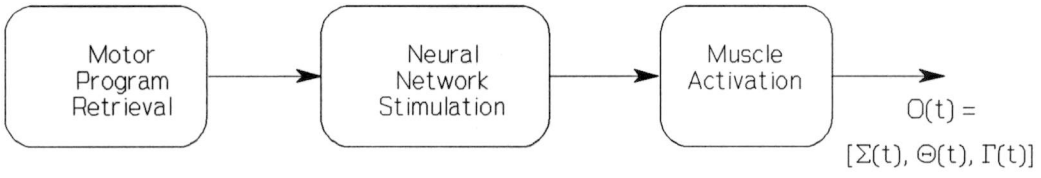


Fig. 7-3: Signature generation.

$$O(t)=[\Sigma(t),\ \Theta(t),\ \Gamma(t)] \qquad \textbf{(7-1)}$$

of a space-time variant system, described by the curvelinear displacement $\Sigma(t)$, the angular displacement $\Theta(t)$, and the torsion $\Gamma(t)$. How this output is constructed, used, and influenced by for instance by psychological mechanisms is still an open question.

Which parts of the process are writer dependent, and which parts are writer independent (singular and semantic information, see also chapter 1), is also still an open question. Which way is the best to extract specific information, and how it is reflected in local or global characteristics is also an question that is unanswered yet [7].

7.4. SIGNATURE VERIFICATION SYSTEM DESCRIPTION

The general signature verification system description is very similar to the character recognition system, however, there are some fundamental differences. Fig. 7-4 shows a data flow diagram of a signature verification system. On the basis of a one-to-one comparison process between the signature under test and the reference signature, a decision (acceptance / rejection) is made.

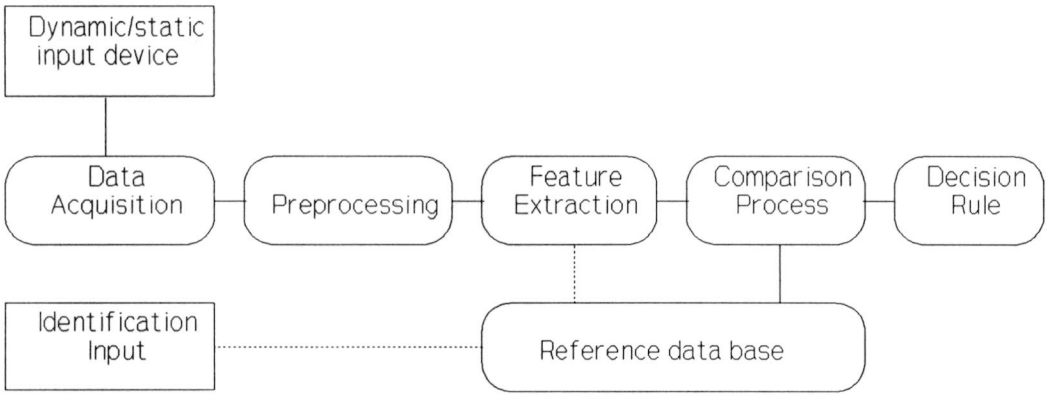


Fig. 7-4: Signature verification system description.

7.4.1. Off-line and on-line signature verification

Signature verification can, just like character recognition, be split-up into on-line and off-line verification. Generally, the problems in off-line verification are considered to be more difficult than in on-line verification. A signature can easily be copied optically or mechanically. Further, dynamic information and for instance pressure signals are lost in a static environment.

Dynamic techniques, on the contrary, do not suffer from this problems

7.4.2. Performance evaluation

Comparison of results from verification experiments is very difficult. Due to the lack of ergonomic equipment, the results from signature verification experiments in laboratory environments differ much from actual field tests [7]. In the laboratory, control over the process is possible, while in the field, almost no control over the signers is possible.

Moreover, tests may differ as the application differs: signature verification systems cover a wide range of applications, varying from local systems with a limited number of users to

very large systems, controlling the identity of a large number of users using remote terminals and a network.

Finally, due to the lack of public signature databases, and the great variation in experimental protocols, comparison of experimental results is very difficult [7].

7.5. HMMs APPLIED TO SIGNATURE VERIFICATION

We shall now try to apply HMMs to signature verification. Principally, all the design parameters that were determined in the previous sections will have to be chosen again. However, since a complete empirical determination of optimum design parameters is beyond the scope of this report, they will be chosen and not empirically determined.

7.5.1. Design parameters

First, the observation sequence for a signature must be considerably larger than for an isolated character. Again based on the maximum number of occurring (x,y) coordinates, the observation sequence length was chosen at T=330.

Due to the faster handwriting in a signature, subtraction of the starting angle is not a good method of gaining rotation-independency. Other methods of angular alignment will have to be implemented. For simplicity, we here constrain the handwriting.

Moreover, although the tablet provides positional information and its derivatives, i.e. speed and acceleration, we shall employ only the (x,y) coordinates.

7.5.2. Signature verification experiment

Using the design parameters from the previous section, HMMs were obtained for four persons: 2 male, 2 female. Next, each test person produced 15 test signatures. In this experiment, no decision threshold was set (see section 7.2.1.), but the following way was

followed (see also Fig. 7-5): for each person, the most-likely reference signature was determined, very similar to the recognition experiments from chapters 5 and 6.

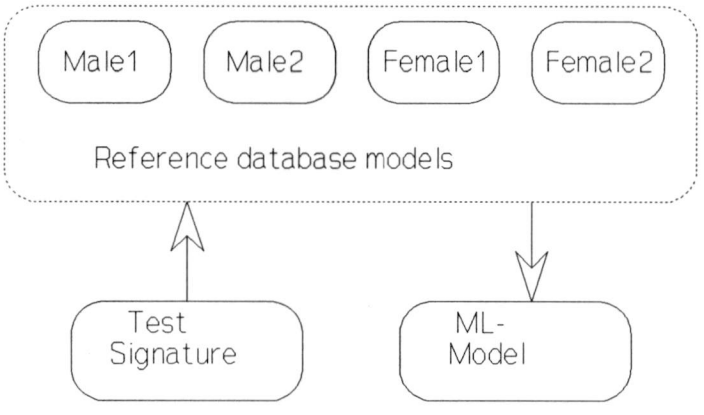


Fig. 7-5: Signature verification experiment.

Result of this experiment is a 100% 'recognition rate', so 100% protection against substitution forgery is achieved. This result indicates that for the limited 4-user system, the model size is sufficient.

7.5.3. Decision threshold

In a verification system, writer identity is verified on the basis of a one-to-one comparison process with a database-signature. So, a decision threshold T_0 has to be chosen. To illustrate the procedure to find a threshold, $\log(P^{BW})$ was determined for all 4 persons (over the 15 test signatures per person), over the 4 database models. So, the system was tested against substitution forgery, using the 3 remaining persons as forgers. Fig. 7-6 illustrates this experiment: it shows the occurrences of $\log(P^{BW})$ of the $4 \cdot 15 = 60$ test signatures over database model Male-2.

Clearly, the signatures belonging to the correct person yield the lowest $\log(P^{BW})$, and thus the highest P^{BW}. A threshold can be found that completely separates the classes

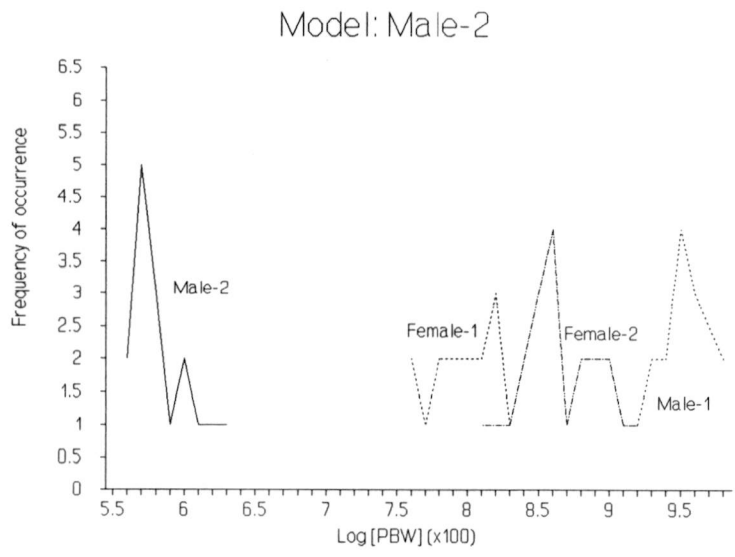


Fig. 7-6: Occurrences of P^{BW} over model Male-2 for each person.

from each other. In this particular case, the threshold could be set at a value for $\log(p^{BW})$ of for instance 7e2. For the 3 other persons, similar graphs can be made. Since the graphs in Fig. 7-6 actually represent discrete distributions, standard detection theory can be applied to determine the threshold. Fig. 7-6 also shows that the smaller the threshold (i.e the more towards the left), the lower the FA ratio, but the higher the FR-errors.

The exact value for the threshold is dependent of the following factors:

1. The average of $\log(P^{BW})$ of all the test signatures over their 'own' HMM.
2. The variation of the $\log(P^{BW})$'s around this average.
3. The trade-off between FA and FR errors, i.e between security demand and user-friendliness

To determine a threshold, the averages of $\log(P^{BW})$ are calculated over all models (see Table 7-1):

Table 7-1: Average $\log(P^{BW})$ over all models.

	Male-1	Male-2	Female-1	Female-2
Male-1	459	1028	656	751
Male-2	947	583	796	874
Female-1	816	1050	703	948
Female-2	804	891	822	708

On basis of these averages, we note that the threshold cannot be fixed, but must be person-dependent. FA and FR errors were computed for two different thresholds: T_0=5% of ave[$\log(P^{BW})$], and T_0=10% of ave[$\log(P^{BW})$].

Table 7-2: FA and FR errors for two values of T_0.

T_0=5%	T_0=5%	T_0=10%	T_0=10%
FA	FR	FA	FR
0%	6.6%	2.8%	1.6%

As can be seen from table 7-2, very reasonable figures can be obtained, given the limitations of for instance the size of the models.

Chapter 8.

Conclusions and recommendations

8.1. CONCLUSIONS

8.1.1. Character recognition

Hidden Markov models provide a solid basis for the recognition of handwritten characters. An error rate of 6.6% was obtained for a system, trained on its user. Multiple training, followed by selecting the model that yields the highest P^{BW} is a condition for good performance.

The intraclass scatter of the characters written by the same person is effectively captured in the models. The error rate may, but not necessarily will, be reduced further by using larger models. However, the increased computational effort makes this unattractive for multimedia applications. An error analysis showed that either changing the writing style, or using ad-hoc decision rules for ambiguous characters [12] is a better way of improving the recognition performance, although the latter method also suffers from disadvantages, such as loss of generality.

When multiple writers are using a system, the error rate rapidly increases. So, a single-user system is the main application of a HMM-based system.

8.1.2. Signature verification

The signature verification experiments show that the application of HMMs are very promising in this field. The 'recognition-system', with only protection against substitution forgery, yielded 0% FA and FR errors. This indicated that for four users the model parameters are sufficient.

The threshold-system, with inherently varying errors depending on the threshold, showed a 0% FA error at 6.6% FR ratio. For a different threshold, 2.8% FA versus 1.6% FR was obtained.

8.2. RECOMMENDATIONS

8.2.1. Character recognition

For non-multimedia applications, it is interesting to determine the performance of systems that use essentially larger models. Another point of interest is to make a different approach to writer-independency: instead of using a single model, optimized on all the users per character, multiple models, all corresponding to the same character, but belonging to different users can be used. The drawback of this method is the increased computational and storage effort.

Another recommendation is the use of continuous probability density functions as a replacement for the rows of the **B** parameter. This investigation can be combined with research that is done at Nijmegen University about handwriting and its relative vector probabilities [14]-[15].

8.2.2. Signature verification

The signature verification based on HMMs should be tested more extensively. The limit of the number of users as a function of the model size is a very interesting point. Since

computational power is less a problem in signature verification applications, experiments with larger populations of users and larger models should be conducted.

Especially for signature verification, but also for character recognition systems, the performance may be increased by enhancing the preprocessing stage. For instance the use of polygonal approximations of the line drawings may result in decreased intraclass scatter.

In order to investigate a system that also protects against skilled forgery, speed, acceleration and pressure information must be used as well. A similar approach as in chapter 7 can be followed, replacing the angular function by one of the signals mentioned previously.

References

[1] LEVINSON, S.E., RABINER, L.R., and SONDHI, M.M., *An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition*, Bell Systems Technical Journal, Vol. 62, No. 4, 1983, pp. 1035-1074.

[2] LEVINSON, S.E., RABINER, L.R., and SONDHI, M.M., *On the application of vector quantization and hidden Markov models to speaker-independent, isolated word recognition*, Bell Systems Technical Journal, Vol. 62, No. 4, 1983, pp. 1075-1105.

[3] COX, S.J., *Hidden Markov models for automatic speech recognition: theory and application*, Br. Telecom Technol. Journal, Vol. 6, No. 2, April 1988, pp.105-115

[4] KUNDU, A., and BAHL, P., *Recognition of handwritten script: A hidden Markov model based approach*, in Proc. Int. Conf. Acoust., Speech, Signal Processing, 1988, pp. 928-931.

[5] NAG, R., WONG, K.H., and FALLSIDE, F., *Script recognition using hidden Markov models*, in Proc. Int. Conf. Acoust., Speech, Signal Processing, 1986, pp. 2071-2074.

[6] TAPPERT, C.C., SUEN, C.Y., and WAKAHARA, T., *The state of the art in on-line handwriting recognition*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 12, No. 8, August 1990, pp. 787-808.

[7] PLAMONDON, R., and LORETTE, G., *Automatic signature verification and writer identification - the state of the art*, Pattern Recognition, Vol. 22, No. 2, 1989, pp. 107-131.

[8] VIEVEEN, J.W., and PRASAD, R., *Generalised Fourier Descriptors for use with line-drawings and other open curves*, Proc. 6th Scandinavian Conference on Image Analysis, Oulu, Finland, June 1989, pp. 820-827.

[9] WEYLAND, N.B.J., and PRASAD, R., *Characterization of line-drawings using Generalised Fourier Descriptors*, Electronics Letters, Vol. 26, No. 21, October 1990, pp. 1794-1795.

[10] WEYLAND, N.B.J., and PRASAD, R., *Criterion for characterization of line-drawings using Generalised Fourier Descriptors*, Proc. 7th Scandinavian Conference on Image Analysis, Aalborg, Denmark, August 1991, pp. 48-55.

[11] BORGGREVEN, M.W., and VELTMAN, S.R., *A study of Generalized Fourier Descriptors applied to handwritten character recognition*, internal report TU Delft, 1991.

[12] YANG, L., and PRASAD, R., *Recognition of line-drawings based on Generalised Fourier Descriptors*, Proc. 4th IEE Int. Conf. on Image Processing and its Applications, Maastricht, Netherlands, April 1992, pp. 286-189.

[13] FREEMAN, H., *On the encoding of arbitrary geometric configurations, IRE Trans. Electronic Computers*, Vol. EC-10, June 1961, pp. 260-268.

[14] NEISSER, U. and WEENE, P., *A note on human recognition of handprinted characters*, Inform. Contr.,Vol. 3, 1960, pp. 191-196.

[15] ----------, *Handwriting generation, perception and recognition*, Acta Psychol., Vol. 54, 1983, pp. 295-312.

[16] ARNBAK, J.C., BONS, J.H., and VIEVEEN, J.W., *Graphical correspondence in electronic mail using personal computers*, IEEE Selected Areas in Comm., Vol. SAC-7, February 1989, pp. 257-267.

Appendix A: Listings of Pascal Programs

```
program Tablet;


{*******************************************
 *                                         *
 *  This program reads a character or a    *
 *  signature from the writing tablet and  *
 *  displays it on the screen. The xy data *
 *  are written to a specific user-definable *
 *  file. Automatic sequence numbering is  *
 *  provided.                              *
 *                                         *
 *******************************************}


uses crt,graph,dos,rs232;


const  MaxSize = 3000;
       start   = 00; {startcode from tablet}
       stop    = 02; {stopcode from tablet}
       retry   = 40; {bottom left, retry with same sequence number}
       next    = 49; {bottom right, save xy data file & goto next sequence number}
       quit    = 09; {top right, new identifier}
       ClrScrn = 00; {top left, clear graphics screen}
       observ  = 32;


var x,y                          : array[0..MaxSize] of integer;
    s_c,ident,temp_input,dummy,
    path,filename                : string[11];
    SeqNum,code                  : integer;
    data,data0,data1,data2,
    data3,data4,dx,dy            : byte;
    sn,Xvid,Yvid                 : integer;
    exit                         : boolean;


Procedure Initialize;

var gm,gd                        : integer;
    font, direction, charsize    : word;

begin
   path :='c:\hmm\';
```

```
   DetectGraph(gd,gm);
   Initgraph(gd,gm,'');
   Rectangle(0,0,Getmaxx,Getmaxy);
   line(0,GetMaxY-20,GetMaxX,GetMaxY-20);
   font :=0; direction :=0; charsize :=0;
   SetTextStyle(font, direction, charsize);
end;


Procedure Message(Textstring : string);

begin
   SetViewPort(1,GetMaxY-19,GetMaxX-1,GetMaxY-1,clipon);
   ClearViewPort;
   MoveTo(10,6);
   OutText(Textstring);
end;


Function Readstring : string;

var key       : string[1];
    sum_key   : string;

begin
   key :=''; sum_key :='';
   while (key <> chr(13)) do
   begin
      key :=ReadKey;
      if (key <> chr(13)) then
      begin
         sum_key :=sum_key+key;
         OutText(key);
      end;
   end;
   Readstring :=sum_key;
end;


Procedure StartUpDialog;

begin
   message('Signature / Character  (S/C) >');
   repeat
      s_c :=ReadKey;
   until (s_c='s') or (s_c='c');
   filename :=s_c;
   repeat
      Message('Identifier (max 4 characters) > ');
      ident :=Readstring;
   until (length(ident)<5);
   filename :=filename+ident;
   repeat
      Message('Initial sequence number (000 - 999) > ');
      temp_input :=Readstring;
      Val(temp_input,SeqNum,code);
   until (code=0) and (SeqNum>=0);
   filename :=filename+temp_input;
   Message('filename : '+path+filename+'.XY');
end;


Procedure ProcessExitCode(fieldcode : byte);

var d_out                    : text;
    j,Xfile,Yfile            : integer;

begin
   fieldcode :=fieldcode and 63;
```

```
   case fieldcode of

   retry   : begin
               {ClearViewPort;}
               sn :=1;
               Message('Filename : '+path+filename+'.XY');
               SetViewPort(1,1,GetMaxX-1,GetMaxY-21,clipon);
             end;

   next    : begin
               assign(d_out,path+filename+'.xy');
               rewrite(d_out);
               for j :=1 to (sn-1) do
               begin
                  Xfile :=x[j]-x[1];
                  Yfile :=y[j]-y[1];
                  writeln(d_out,Xfile:6,'     ',Yfile:6);
               end;
               close(d_out);
               SeqNum :=SeqNum+1;
               Str(SeqNum, temp_input);
               filename :=s_c+ident+temp_input;
               {ClearViewPort;}
               Message('Filename : '+path+filename+'.XY');
               SetViewPort(1,1,GetMaxX-1,GetMaxY-21,clipon);
               sn :=1;
             end;

   ClrScrn : ClearViewPort;

   quit    : exit :=true;

   else      begin
               {ClearViewPort;}
               sn :=1;
               Message('Filename : '+path+filename+'.XY');
               SetViewPort(1,1,GetMaxX-1,GetMaxY-21,clipon);
             end;

   end;
end;



Procedure ReadTablet;

var scale,scaleX,scaleY : real;

Procedure CalcVideo;

begin
   Xvid :=round(x[sn]/scale);
   Yvid :=round((1536-y[sn])/scale);
end;

begin
  SetViewPort(1,1,GetMaxX-1,GetMaxY-21,clipon);
  scaleX :=2100/(GetMaxX-2);
  scaleY :=1536/(GetMaxY-22);
  if (scaleX>scaleY) then scale :=scaleY else scale :=scaleX;
  BaudRate;
  exit :=false;
  sn :=1;                                          {SampleNumber :=1}
  repeat
     GetByte(data0);
     if data0=0 then                               {Startcode}
     begin
        GetByte(data1);
```

```
        GetByte(data2);
        if (data2 <> stop) then                    {Pen down, no Fieldcode}
        begin
           GetByte(data3);
           GetByte(data4);
           x[sn] :=(data1 shl 6) + data2 - 4160;
           y[sn] :=(data3 shl 6) + data4 - 4160;  {Absolute XY location calculated}
           CalcVideo;
           putpixel(Xvid,Yvid,15);
           MoveTo(Xvid,Yvid);
           sn :=sn+1;
        end
        else ProcessExitCode(data1);               {Pen down, Fieldcode}
     end else
     begin
        if (data0 <> stop) then
        begin
           dx:=data0 and $BF;
           GetByte(data1);
           dy:=data1 and $BF;
           if dx > 31 then x[sn]:=x[sn-1]-64+dx
           else x[sn] :=x[sn-1]+dx;
           if dy > 31 then y[sn] :=y[sn-1]-64+dy
           else y[sn]:=y[sn-1]+dy;                 {XY calculated from differentials}
           CalcVideo;
           LineTo(Xvid,Yvid);
           sn :=sn+1;
        end;
     end;
     if (sn>MaxSize) then
     begin
        Message('Out of video memory error ; Save data / Retry (S/R)');
        SetViewPort(1,1,GetMaxX-1,GetMaxY-21,clipon);
        repeat
           dummy :=ReadKey;
        until (dummy='s') or (dummy='r');
        if (dummy='s') then data :=next else data :=retry;
        ProcessExitCode(data);
        x[0] :=x[MaxSize]; y[0] :=y[MaxSize];
        MoveTo(round(x[MaxSize]/scale),round((1536-y[MaxSize])/scale));
     end;
  until (Exit)
end;


BEGIN
   Initialize;
   repeat
      StartUpDialog;
      ReadTablet;
      Message('Quit / Continue (Q/C)');
      Repeat
         dummy :=ReadKey;
      until (dummy='q') or (dummy='c');
   until (dummy='q');
   closegraph;
END.




unit rs232;

INTERFACE

uses dos;
```

```
var regs                        : Registers;

Procedure BaudRate;

Procedure GetByte(var a: byte);


IMPLEMENTATION


procedure BaudRate;

{bit 7/6/5  :     111 : 9600 baud
                  110 : 4800
                  101 : 2400
                  100 : 1200
                  011 : 600
                  010 : 300
                  001 : 150
                  000 : 110

 bit 4/3    :      x0 : no parity
                   11 : even parity
                   01 : odd parity

 bit 2      :       1 : 0 stop bit
                    0 : 1 stop bits

 bit 1/0    :     11 : 8 bit data
                  10 : 7 bit data
                  01 : undefined
                  00 : undefined}

begin
  regs.ax:=$00E3;
  regs.dx:=0;
  intr($14,regs);
end;

Procedure GetByte{(var a:byte)};

begin
   inline
    ($50/
     $52/
     $BA/>$03FC/
     $B0/$02/
     $EE/
     $BA/$03FD/
     $EC/
     $24/$01/
     $74/$F8/
     $BA/>$03F8/
     $EC/
     $C4/$BE/a/
     $26/$88/$05/
     $BA/>$03FC/
     $32/$C0/
     $EE/
     $5A/
     $58);
end;


END.
```

```
PROGRAM CHHARACTER PREPROCESSING;        CONVERTS THE XY-DATAFILE TO AN OBSERVATION SEQUENCE


Program Char_Preprocessing;

uses Crt,Dos,Printer;


const ksi    = 1.5;
      dSmax  = 15;
      DotSyms= 4;

var       x,y                          : array[0..500] of integer;
          s                            : array[0..500] of real;
          Theta                        : array[1..500] of real;
          Phi,Phi_s                    : array[1..64] of real;
          dX,dY,n,Nmax,k,levels,PenUp,
          SeqNum,Kmax                  : integer;
          path,filename,Num            : string[12];
          d_out,d_out2                 : text;
          dS                           : real;
          hook,dot                     : boolean;
          letter                       : char;


Procedure ReadXY;

var d_in  : text;

begin
   assign(d_in,'b:\'+path+filename+'.XY');
   reset(d_in);
   n :=0; s[0] :=0;
   while not eof(d_in) do
   begin
      readln(d_in,X[n],Y[n]);
      n :=n+1;
   end;
   Nmax :=n-1;
   close(d_in);
end;



Procedure CalcTheta;

var quadrant        : integer;
    dS              : real;

begin
   PenUp :=0;
   for n :=1 to Nmax do
   begin
      quadrant :=0;
      dX :=X[n]-X[n-1]; dY :=Y[n]-Y[n-1];
      dS :=sqrt(sqr(dX)+sqr(dY));
      if (dS>dSmax) then
      begin
         PenUp :=n;
         writeln(d_out2,'PenUp detected, character ',letter+Num);
      end;
      s[n] :=s[n-1]+dS;
      if (dX<>0) then
      begin
         if (dX<0) then quadrant :=1
           else if (dY<0) then quadrant :=2;
         Theta[n] :=arctan(dY/dX)+quadrant*pi;
      end else
```

```
      if (dY>0) then Theta[n] :=pi/2 else Theta[n] :=3*pi/2;
      if (Theta[n]<0) then Theta[n] :=Theta[n]+2*pi;
   end;
end;


Procedure HookDetection;

var Delta,dT : real;

begin
   n :=1; Delta :=0; hook :=false;
   Repeat
      dT :=(Theta[n+1]-Theta[n]);
      if dT>pi then dT :=dT-2*pi;
      if dT<-pi then dT :=dT+2*pi;
      Delta :=Delta+dT;
      n :=n+1;
      if (abs(Delta)>ksi) then hook :=true;
   until (hook) or (s[n]>0.05*s[Nmax]);
end;



Procedure HookCorrection;

var j          : integer;
    offset     : real;

begin
   offset :=s[n-1];
   for j :=1 to Nmax-n+1 do
   begin
      s[j] :=s[j+n-1]-offset;
      Theta[j] :=Theta[j+n-1];
   end;
   writeln(d_out2,Nmax-j,' samples removed, character ',letter+Num);
   Nmax :=Nmax-n+1;
end;


Procedure DetectDot;

begin
   dot :=false;
   if (PenUp<>0) and (Nmax-PenUp<3) then
   begin
      Nmax :=PenUp-1;
      dot :=true;
      writeln(d_out2,'Dot generated, character ',letter+Num);
      Kmax :=Kmax-DotSyms;
   end;
end;



Procedure Rotate;

begin
   for n :=2 to Nmax do
   begin
      Theta[n] :=Theta[n]-Theta[1];
      if (Theta[n]<0) then Theta[n] :=Theta[n]+2*pi;

   end;
   Theta[1] :=0;
end;
```

```
Procedure Normalize;

var position         : real;

Function Interpolate(Y0,Y1 : real): real;

begin
   Interpolate := Y0+(position-s[n-1])*(Y1-Y0)/(s[n]-s[n-1]);
end;

begin
   phi[1] :=Theta[1]; n :=2;
   for k :=1 to Kmax-2 do
   begin
      position :=s[1]+(s[Nmax]-s[1])*k/Kmax;
      while (s[n]<position) do n :=n+1;
      if (Theta[n]-Theta[n-1]<=-3.1416) then
      begin
         Phi[k+1] :=Interpolate(Theta[n-1],2*pi+Theta[n]);
      end else
      if (Theta[n]-Theta[n-1]>=3.1416) then
      begin
         Phi[k+1] :=Interpolate(Theta[n-1],Theta[n]-2*pi);
      end
        else Phi[k+1] :=Interpolate(Theta[n-1],Theta[n]);
      if (Phi[k+1]>2*pi) then Phi[k+1] :=Phi[k+1]-2*pi;
      if (Phi[k+1]<0) then Phi[k+1] :=Phi[k+1]+2*pi;
   end;
   phi[Kmax] :=Theta[Nmax];
end;



Procedure Quantize;

var      StepSize,Symbol   : real;


begin
   StepSize :=2*pi/(levels);
   for k :=1 to Kmax do
   begin
      symbol :=int(Phi[k]/StepSize);
      Phi_s[k] :=(0.5+Symbol)*StepSize;
      writeln(d_out,Symbol+1:3:0);
      if (symbol>levels) then
      begin
         sound (400);
         delay(1000);
         Nosound;
      end;
   end;
   if (dot) then for k :=1 to DotSyms do writeln(d_out,levels+1:3);
end;




BEGIN
   levels :=16; Kmax :=64;
   assign(d_out2,'c:\hmm\report.2');
   rewrite(d_out2);
```

```
   for letter := 'a' to 'z' do
   begin
      MkDir('C:\HMM\SYM_'+letter);
      for SeqNum :=0 to 9 do
      begin
         Kmax :=64;
         Str(SeqNum,Num);
         filename :='c'+letter+Num;
         path :='char_'+letter+'\';
         assign(d_out,'C:\hmm\SYM_'+letter+'\'+filename+'.SYM');
         rewrite(d_out);
         ClrScr;
         writeln('Processing character : ',letter,SeqNum);
         ReadXY;
         CalcTheta;
         HookDetection;
         if (hook) then HookCorrection;
         DetectDot;
         if (dot) then writeln('dot found');
         Rotate;
         Normalize;
         Quantize;
         close(d_out);
      end;
   end;
   close(d_out2);
END.



PROGRAM TRAINING;        TRAINS AN HMM ON MULTIPLE OBSERVATION SEQUENCES.

Program BaumWelch_Training;

uses crt,dos;


const States   = 8;
      Symbols  = 17;
      T        = 64;
      StopCrit = 1e-6;
      Epsilon  = 1e-6;


var   i,j,k,MinSeq,MaxSeq,time,
      trial                        : integer;
      ident                        : char;
      A,NumA                       : array[1..States,1..States] of Extended;
      B,NumB                       : array[1..States,1..Symbols] of Extended;
      Ini,DenA,DenB                : array[1..States] of Extended;
      observation                  : array[1..T] of integer;
      alpha,beta                   : array[1..T,1..States] of Extended;
      c                            : array[1..T] of Extended;
      Pbw,PbwTotal,PbwOld,PbwMax   : Extended;
      path                         : string;
      converged                    : boolean;
      d_out                        : text;
      filename                     : string[11];

{
   Procedure Display_A
   Displays the state transition matrix A
}
Procedure Display_A;

var ii,jj     : integer;
```

```
begin
   writeln('  A matrix : ( ',ident,' )');
   for ii:= 1 to States do
   begin
      for jj :=1 to States do
      begin
         write(A[ii,jj]:6:2);
      end;
      writeln;
   end;
   writeln; writeln('  B Matrix : ( ',ident,' )');
   for ii:=1 to States do
   begin
      for jj := 1 to Symbols do
      begin
         write(B[ii,jj]:6:2);
      end;
      writeln;
   end;
   writeln;
end;


{
  Procedure Initialize;
  Initializes the system. Model structure can be
  read from disk, as well as initial estimates.
}
Procedure Initialize;


begin
   ClrScr;
   path :='c:\hmm\';
   writeln('Program Model Training');
   writeln;
   write('Enter starting observation sequence number > '); readln(MinSeq);
   writeln;
   write('Enter final observation sequence number > '); readln(MaxSeq);
   writeln;
   write('Enter model structure filename > '); readln(filename);
end;



Procedure InitValue;

var    d_in       : text;
       sum        : Extended;
       l,dummy    : integer;

begin
   path :='C:\HMM\';
   assign(d_in,path+filename);
   reset(d_in);
   ClrScr;
   writeln('Model Structure : ',filename);
   for i := 1 to States do
   begin
      sum :=0;
      for j := 1 to States do
      begin
         read(d_in,A[i,j]);
         write(A[i,j]:7:3);
         if (A[i,j]=1) then A[i,j] :=1+random(100);
         sum :=sum+A[i,j];
      end;
      readln(d_in); writeln;
```

```
      for j:=1 to States do A[i,j] :=A[i,j]/sum;
   end;
   readln(d_in); writeln;
   for i :=1 to States do
   begin
      sum :=0;
      for l :=1 to Symbols do
      begin
         read(d_in,B[i,l]);
         write(B[i,l]:7:3);
         if (B[i,l]=1) then B[i,l] :=1+random(100);
         sum :=sum+B[i,l];
      end;
      readln(d_in); writeln;
      for l :=1 to Symbols do B[i,l] :=B[i,l]/sum;
   end;
   sum :=0; readln(d_in);
   for j :=1 to States do
   begin
      Read(d_in,Ini[j]);
      if (Ini[j]=1) then Ini[j] :=1+random(100);
      sum :=sum+Ini[j];
   end;
   for j :=1 to States do Ini[j] :=Ini[j]/sum;
   close(d_in);
end;


{
  Procedure Forward_Backward.
  This procedure calculates the forward and back-
  ward probabilities, plus Pbw.
}
Procedure Forward_Backward;


begin
   Pbw :=0; c[1] :=0;
   for j := 1 to States do alpha[1,j] :=Ini[j]*B[j,observation[1]];
   for j := 1 to States do c[1] :=c[1]+alpha[1,j];
   for j := 1 to States do alpha[1,j] :=alpha[1,j]/c[1];
   for time :=1 to T-1 do
   begin
      c[time+1] :=0;
      for j :=1 to states do
      begin
         alpha[time+1,j]:=0;
         for i :=1 to states do
         begin
            alpha[time+1,j] :=alpha[time+1,j]+alpha[time,i]*A[i,j];
         end;
         alpha[time+1,j] :=alpha[time+1,j]*B[j,observation[time+1]];
      end;
      for j :=1 to States do c[time+1] :=c[time+1]+alpha[time+1,j];
      for j :=1 to States do alpha[time+1,j] :=alpha[time+1,j]/c[time+1];
   end;

  {start of backward procedure}

   for i := 1 to States do beta[T,i] :=1/c[T];
   for time := T-1 downto 1 do
   begin
      for i :=1 to States do
      begin
         beta[time,i] :=0;
         for j := 1 to States do
         begin
            beta[time,i] :=beta[time,i]+A[i,j]*B[j,observation[time+1]]*beta[time+1,j];
```

```
        end;
      end;
      for i :=1 to States do beta[time,i] :=beta[time,i]/c[time];
   end;
   for time := 1 to T do PbwTotal :=PbwTotal-ln(1/c[time]);
end;


{
  Procedure Reestimate
  Reestimates the A and B parameters
}
Procedure Reestimate;

var       gamma     : Extended;

begin
   for i :=1 to states do
   begin
      for j :=1 to states do
      begin
         gamma :=0;
         for time :=1 to T-1 do
         begin
            gamma :=gamma+alpha[time,i]*A[i,j]*B[j,observation[time+1]]*beta[time+1,j];
         end;
         NumA[i,j] :=NumA[i,j]+gamma;
         DenA[i] :=DenA[i]+gamma;
      end;
   end;

  {Reestimate B parameters}

   for j :=1 to States do
   begin
      for time :=1 to T do
      begin
         NumB[j,observation[time]] :=NumB[j,observation[time]]+alpha[time,j]*beta[time,j]*c[time];
         DenB[j] :=DenB[j]+alpha[time,j]*beta[time,j]*c[time];
      end;
   end;
end;


{
  Procedure ClearSums
  Clears the numerators and Denominators
}
Procedure ClearSums;

var ii,jj : integer;

begin
   PbwOld :=PbwTotal;
   PbwTotal :=0;
   for ii := 1 to States do
   begin
      DenA[ii] :=0;
      DenB[ii] :=0;
      for jj := 1 to States do NumA[ii,jj] :=0;
      for jj := 1 to Symbols do NumB[ii,jj] :=0;
   end;
end;
```

```
{
  Procedure ReadSequence
  Reads an observation sequence from disk
}
Procedure ReadSequence;

var       filename,num    : string;
          d_in            : text;
          dummy           : integer;

begin
   Str(k,num);
   filename :='C'+ident+num+'.SYM';
   path :='B:\SYM_'+ident+'\';
   assign(d_in,path+filename);
   reset(d_in);
   for time := 1 to T do
   begin
      readln(d_in,observation[time]);
   end;
   close(d_in);
end;


{
  Procedure Replace Coefficients
  Replaces the coefficients of A and B with new
  estimates.
}
Procedure ReplaceCoefficients;

var ii,jj       : integer;

begin
   for ii :=1 to States do
   begin
      for jj :=1 to States do A[ii,jj] :=NumA[ii,jj]/DenA[ii];
      for jj :=1 to Symbols do B[ii,jj] :=NumB[ii,jj]/DenB[ii];
   end;
end;


{
  Procedure CalcCriterion
  Checks if the Baum-welch algorithm has
  converged
}
Procedure CalcCriterion;

var       deltaPbw : Extended;

begin
   deltaPbw :=PbwTotal-PbwOld;
   if deltaPbw<0 then
   begin
      sound(400);
      delay(250);
      nosound;
   end;
   writeln('ln[Pbw] : ',PbwTotal,'     Delta : ',deltaPbw:1,' char : ',ident,trial);
   if (abs(deltaPbw/PbwOld)<StopCrit) then converged :=true
      else converged :=false;
end;


{
```

```
{
  Procedure ConstrainB
  Places Epsilon type constraints on B.
}
Procedure ConstrainB;

var R     : integer;
    sum   : Extended;

begin
   for i :=1 to States do
   begin
      R :=0; sum :=0;
      for j :=1 to Symbols do
      begin
         if (B[i,j]<=Epsilon) then
         begin
            R :=R+1;
            B[i,j] :=epsilon;
         end else sum :=sum+B[i,j];
      end;
      for j :=1 to Symbols do
      begin
         if (B[i,j]>Epsilon) then B[i,j] :=(1-R*Epsilon)*B[i,j]/sum;
      end;
   end;
end;


{
  Procedure StoreModel;
  Stores optimized model on disk
}
Procedure StoreModel;

var ii,jj         : integer;
    filename      : string;
    d_out         : text;
    try           : string;

begin
   filename :=ident+'_opt.hmm';
   assign(d_out,'c:\hmm\'+filename);
   rewrite(d_out);
   writeln(d_out,'A matrix :');
   for ii :=1 to States do
   begin
      for jj := 1 to States do writeln(d_out,A[ii,jj]);
   end;
   writeln(d_out,'B matrix :');
   for ii :=1 to States do
   begin
      for jj :=1 to Symbols do writeln(d_out,B[ii,jj]);
   end;
   close(d_out);
end;


BEGIN
   Randomize;
   Initialize;
   for ident :='a' to 'z' do
   begin
      PbwMax :=1e100;
      for trial := 1 to 10 do
      begin
         PbwTotal :=-3e-3;
```

```
        InitValue;
        repeat
           ClearSums;
           for k :=MinSeq to MaxSeq do
           begin
                ReadSequence;
              Forward_Backward;
              Reestimate;
           end;
           display_a;
           ReplaceCoefficients;
           CalcCriterion;
        until (converged);
        if abs(PbwTotal)<PbwMax then
        begin
           ConstrainB;
           StoreModel;
           PbwMax :=abs(PbwTotal);
        end;
     end;
  end;
END.
```

```
PROGRAM CLASSIFY;        CLASSIFIES AN UNKNOWN OBSERVATION SEQUENCE TO THE MODEL WITH THE MAXIMUM
                         LIKELIHOOD OF HAVING EMMITTED THAT SEQUENCE.

Program ML_Classification;

uses Crt,Dos;

const States   = 8;
      Symbols  = 17;
      T        = 64;
      Classes  = 10;


var   i,j,k,MinSeq,MaxSeq,time,
      class                          : integer;
      ident,ML                       : char;
      A                              : array[1..Classes,1..States,1..States] of Extended;
      B                              : array[1..Classes,1..States,1..Symbols] of Extended;
      Ini                            : array[1..Classes,1..States] of Extended;
      observation                    : array[1..T] of integer;
      alpha,beta                     : array[1..T,1..States] of extended;
      Pbw                            : array[1..Classes] of extended;
      path                           : string;
```

```
{
  Procedure ReadSequence
  Reads an unknown observation sequence from disk
}
Procedure ReadSequence;

var       filename,num    : string;
          d_in            : text;


begin
   Str(k,num);
```

```
   filename :='C'+ident+num+'.SYM';
   assign(d_in,'B:\SYM_'+ident+'\'+filename);
   reset(d_in);
   for time := 1 to T do
   begin
      readln(d_in,observation[time]);
   end;
   close(d_in);
end;
```

```
{
  Procedure ReadModels
  Reads for each class an HMM from disk.
}
Procedure ReadModels;

var d_in           : text;
    ModelName,path  : string[15];

begin
   path :='C:\HMM\';
   for Class := 1 to Classes do
   begin
      ModelName :=char(96+Class)+'_opt.HMM';
      assign(d_in,path+ModelName);
      reset(d_in); readln(d_in);
      for i :=1 to States do
      begin
         for j :=1 to States do
         begin
            readln(d_in,A[Class,i,j]);
         end;
      end;
      readln(d_in);
      for i :=1 to States do
      begin
         for j :=1 to Symbols do
         begin
            readln(d_in,B[Class,i,j]);
         end;
      end;
      readln(d_in);
      for i :=1 to States do
      begin
         Ini[Class,i] :=0;
      end;
      Ini[Class,1] :=1;
      close(d_in);
   end;
end;
```

```
{
  Procedure Forward_Backward.
  This procedure calculates the forward and back-
  ward probabilities, plus Pbw.
}
Procedure Forward_Backward;

begin
   Pbw[class] :=0;
   for j := 1 to States do alpha[1,j] :=Ini[Class,j]*B[Class,j,observation[1]];
   for time :=1 to T-1 do
   begin
      for j :=1 to states do
      begin
         alpha[time+1,j]:=0;
```

```
         for i :=1 to states do
         begin
            alpha[time+1,j] :=alpha[time+1,j]+alpha[time,i]*A[Class,i,j];
         end;
         alpha[time+1,j] :=alpha[time+1,j]*B[Class,j,observation[time+1]];
      end;
   end;

  {start of backward procedure}

   for i := 1 to States do beta[T,i] :=1;
   for time := T-1 downto 1 do
   begin
      for i :=1 to States do
      begin
         beta[time,i] :=0;
         for j := 1 to States do
         begin
            beta[time,i]
:=beta[time,i]+A[Class,i,j]*B[Class,j,observation[time+1]]*beta[time+1,j];
         end;
      end;
   end;
   for i :=1 to States do Pbw[Class] :=Pbw[Class]+alpha[T,i];
   Pbw[class] :=abs(ln(Pbw[Class]));
end;


{
   Procedure Classify
   Maximum-Likelihood classfier
}
Procedure Classify;

var PbwMax         : extended;

begin
   PbwMax :=1e6;
   for Class :=1 to Classes do
   begin
      if Pbw[Class]<PbwMax then
      begin
         PbwMax :=Pbw[Class];
         ML :=char(Class+96);
      end;
   end;
end;


{
            START OF MAIN PROGRAM LOOP
}
BEGIN
   ClrScr;
   ReadModels;
   for ident :='a' to 'z' do
   begin
      MinSeq :=0; MaxSeq :=24;
      for k :=MinSeq to MaxSeq do
      begin
         ReadSequence;
         for class :=1 to Classes do
         begin
            Forward_Backward;
         end;
         Classify;
```

```
            write(ML);
        end;
        writeln;
    end;
END.
```

Appendix B: Examples of used characters and signatures

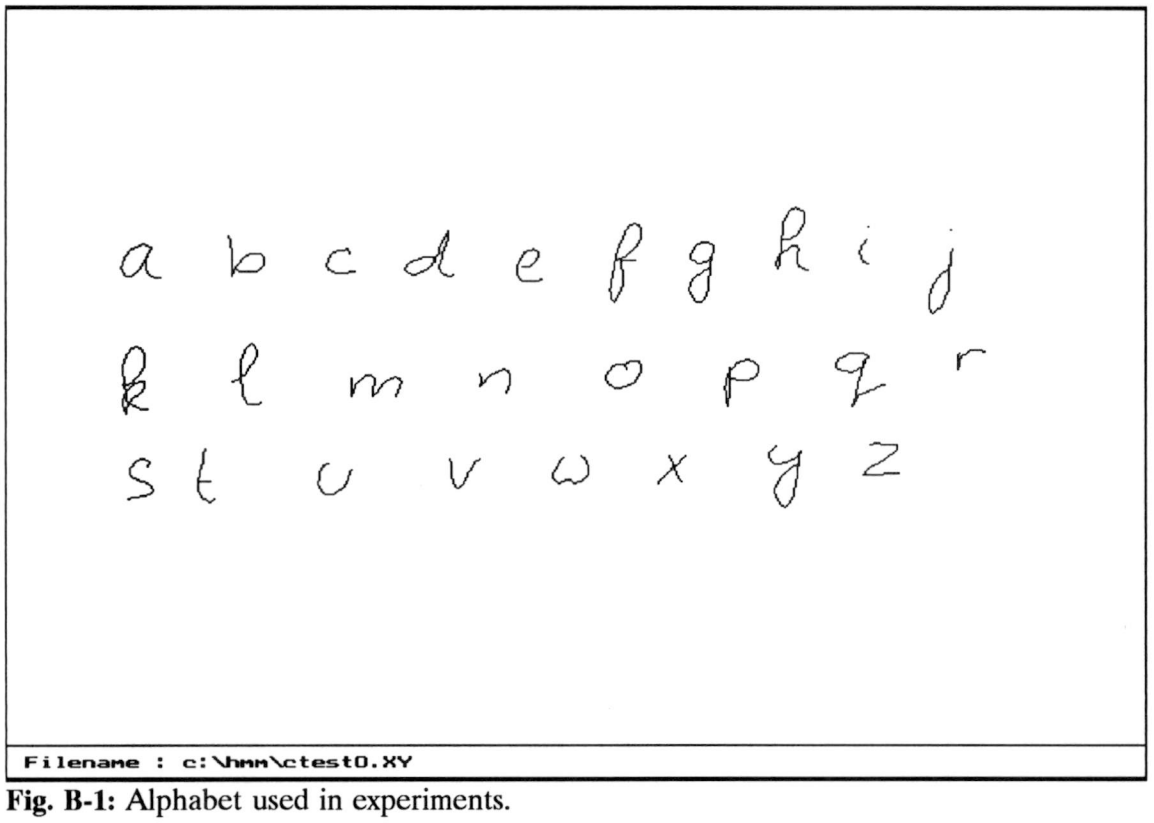


Fig. B-1: Alphabet used in experiments.

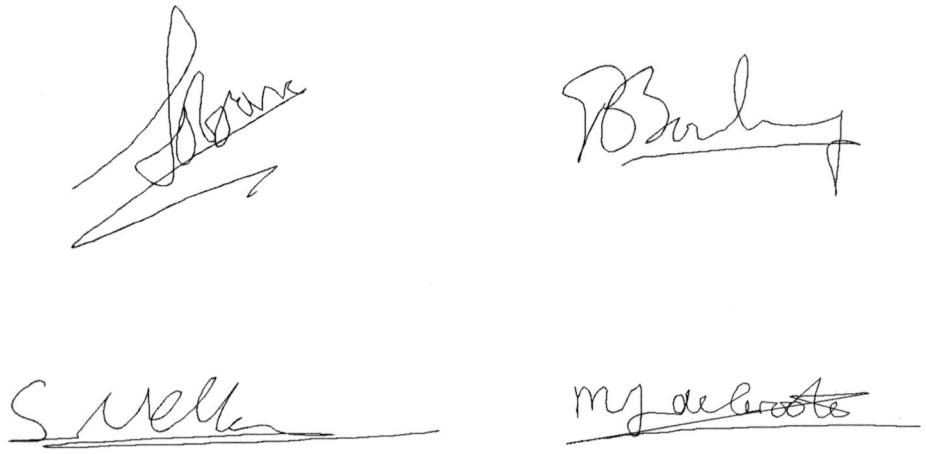


Fig. B-2: Signatures of four test persons.

Appendix C: Examples of symbols

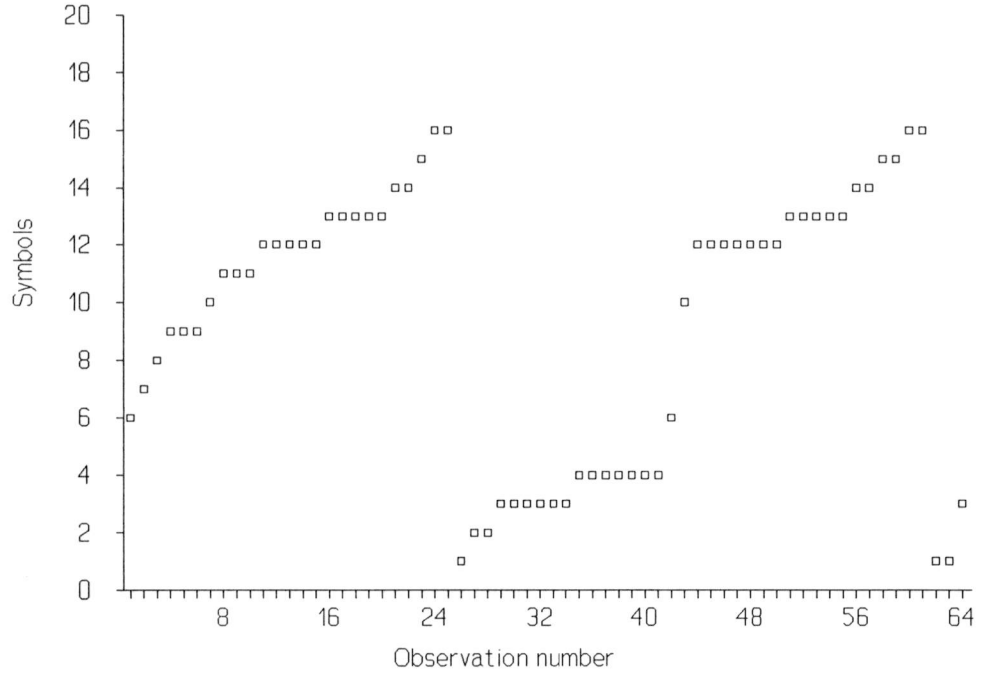


Fig. C-1: Symbol sequence for character 'a'.

Appendix D: Paper submitted for publication in Electronics Letters

HIDDEN MARKOV MODELS APPLIED TO ON-LINE SIGNATURE VERIFICATION

Ramjee Prasad and Stephan R. Veltman

Delft University of Technology
Telecommunications and Traffic Control Systems Group
P.O. Box 5031, 2600 GA Delft, The Netherlands
Tel.: +31 15 782417, Fax:+31 15 781774

ABSTRACT:

The theory of probabilistic functions of Hidden Markov Models is applied as a novel technique to the on-line verification of signatures. This method has some properties that make it well-suited to operate remotely using a network. Experiments have shown that good results can be obtained with this method.

INTRODUCTION Due to the integration of digitizing tablets and display units into a single terminal, the interest in on-line classification of handwritten script is renewed. This letter focuses on on-line signature verification. On the basis of 'singular', i.e. writer-dependent information, we try to verify the writer identity by a one-to-one comparison process with a reference database. Signatures are captured on-line by a digitizing tablet. Techniques which are suited for remote signature verification through a network are especially interesting, since many applications lie in this field. The theory of Hidden Markov Models (HMMs), which has shown to be successful in Automatic Speech Recognition (ASR), is applied to the verification problem. A special class of HMMs has shown to combine well with the temporal order of the strokes, which is captured by the tablet. Detection theory is then applied to determine decision thresholds, and some experiments are conducted that show that HMMs are very promising in the field of signature verification.

HMMs APPLIED TO THE MATCHING PROBLEM An excellent description of the application of the theory of HMMs to the matching problem can be found in [1]. Here, we give a very short summary of the theory. A HMM is defined as a doubly stochastic process with an underlying stochastic process that is not observable, but can be observed through another set of stochastic processes that produce the sequence of symbols [1]. HMMs are fully determined by

- The state transition matrix: **A**
- The symbol generation matrix: **B**
- The initial probability vector: π

At each time instant t, the model emits a symbol, and moves to the next state on the basis of the state transition probabilities. Only the symbol sequence can be observed, so the underlying state sequence is unknown ('hidden'). Principally, from each state a transition to any other state is possible; however, left-to-right models, i.e. models that only allow transitions to higher numbered states (see fig. 1), have shown to combine well with the temporal order of the strokes of the handwriting, and are used in this work.

Optimization Through a process of feature extraction, observation sequences of the signatures are generated. During training, models are optimized on observation sequences

of known classification. Let **O** ($\mathbf{O}=O_1,O_2,O_3,...,O_T$) denote the vector containing the observation sequence, then $Pr(\mathbf{O}|M)$ is optimized. This optimization can be done using classical methods, like with Lagrange Multipliers, but we used the so-called Baum-Welch algorithm. In order to capture the variability within the signatures, training on multiple observation sequences is necessary. Let K denote the total number of observation sequences available belonging to one person, then we optimize

$$\prod_{k=1}^{K} Pr(O^{(k)}|M) \qquad (1)$$

Because all methods find a locally optimum point, multiple optimizations under different initial estimates followed by selecting the model that yields the highest $Pr(\mathbf{O}|M)$ are necessary. Resulting, we have for each person a model of his/her signatures stored in a database.

Classification During classification, observation sequences of unknown classification are 'scored' on the reference model for that person from the database. Thus, the probability $Pr(\mathbf{O}|M)$ that the signature belongs to this reference model is calculated. In verification systems, we set a threshold for each model, and compare the $Pr(\mathbf{O}|M)$ that the test signature was generated by the model to this threshold. In this way, a decision (Acceptance / Rejection) is made. Inherently, due to the practical class overlap, two types of errors are introduced: False Acceptance (FA), and False Rejection (FR) [2].

The training procedure is computationally very intensive, unlike the classification, which can be done in real-time. So, in a practical system, training can be done on a central host, for instance by transmitting the signatures using Differential Chain Coding (DCC). The optimized models can be transmitted back, and verification can be done locally.

FEATURE EXTRACTION The requirements on the features are defined as follows: they should reduce the intraclass variability (i.e. the within-person variability) as much as possible, while on the other hand, they should have a sufficiently discriminant power. Clearly, these

two requirements may be contradictionary. More concrete, these requirements lead to features that are independent of translation, rotation, and scaling of the curve.

We used an absolute angular description, similar to the definition as defined with GFDs [3]. By nature, angular descriptions are independent of translation. Writers were constrained in the angle of writing, thus yielding rotation-independency. Signatures are available as a set of N discrete (x,y) coordinates. So, no speed, acceleration or pressure information is used. From the (x,y) coordinates, we derive N-1 angular directions:

$$\phi(s_n) = \arctan\left(\frac{dY_n}{dX_n}\right) \qquad (2)$$

where s denotes the distance along the curve. The discrete $\phi(s_n)$-values are interconnected using linear interpolation, resulting in a piecewise-linear function $\phi(s)$. Next, the s-axis is divided into K equidistant sections, and for each k (k=1,2,3,...,K) $\phi[k]$ is determined. Finally, if L denotes the total length of the curve, then each value of $\phi[k]$ is quantized, and thus mapped into one of M quantization levels, i.e. the observable symbols:

$$\phi^*[k] = Q\left[\phi\left(\frac{k \cdot s}{L}\right)\right] = O_k \qquad (3)$$

In this way, a size normalization procedure is obtained, yielding scale-independency.

EXPERIMENTS AND CONCLUSION For the experiments, signatures belonging to 4 persons were digitized: 2 male, 2 female. Each person wrote 25 signatures; 10 for training, and the remaining 15 for test purposes. Further, the following parameters were used:

- Observation sequence length: T=330
- Number of states: N=6
- Number of symbols: M=16

Next, the system was tested against substitution forgery, i.e. forgery where the forger uses his/her own signature. The remaining 45 signatures belonging to the 3 remaining persons

were used as forgeries. So, over each of the 4 reference models, for 60 signatures $\log[\Pr(\mathbf{O}|M)]$ was computed. The occurrences of $\log[\Pr(\mathbf{O}|M)]$ over each of the 4 models were determined. Fig. 2 shows these occurrences over model male-2. The thresholds T_0 were chosen at a certain distance from the average of $\log[\Pr(\mathbf{O}|M)$ of signatures belonging to their real writer (see also Fig. 2). For two thresholds, FA and FR ratios were determined.

The results yielded 0% FA and 6.6% FR at T_0=5% of ave $\{\log[\Pr(\mathbf{O}|M)]\}$, and 2.8% FA and 1.6% FR at T_0=10% of ave $\{\log[\Pr(\mathbf{O}|M)]\}$, respectively

Conclusion The method used here to verify writer identity by signature verification appears to be very promising. Given the small model size, and the fact that for instance speed and acceleration information was not used, the results achieved with this method are very good.

REFERENCES

[1] LEVINSON, S.E., RABINER, L.R., and SONDHI, M.M., *An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition*, Bell Systems Technical Journal, Vol. 62, No. 4, 1983, pp. 1035-1074.

[2] PLAMONDON, R., and LORETTE, G., *Automatic signature verification and writer identification - the state of the art*, Pattern Recognition, Vol. 22, No. 2, 1989, pp. 107-131.

[3] VIEVEEN, J.W., and PRASAD, R., *Generalised Fourier Descriptors for use with line-drawings and other open curves*, Proc. 6th Scandinavian Conference on Image Analysis, Oulu, Finland, June 1989, pp. 820-827.

LIST OF FIGURE CAPTIONS

Figure 1

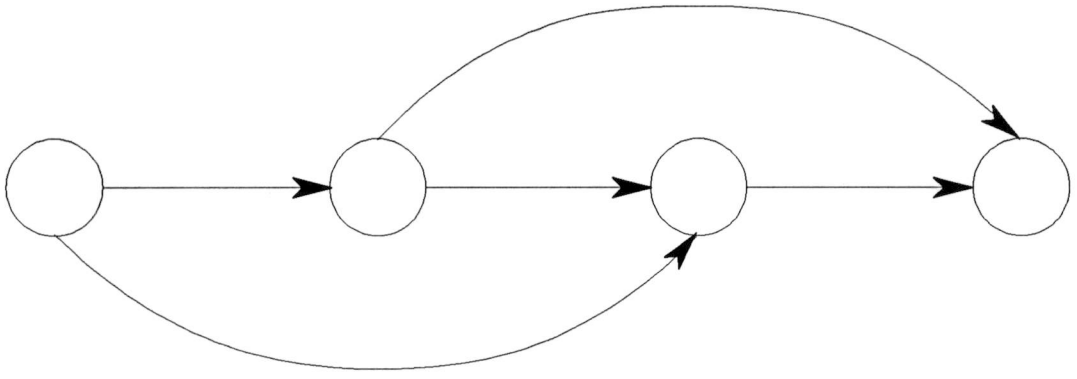
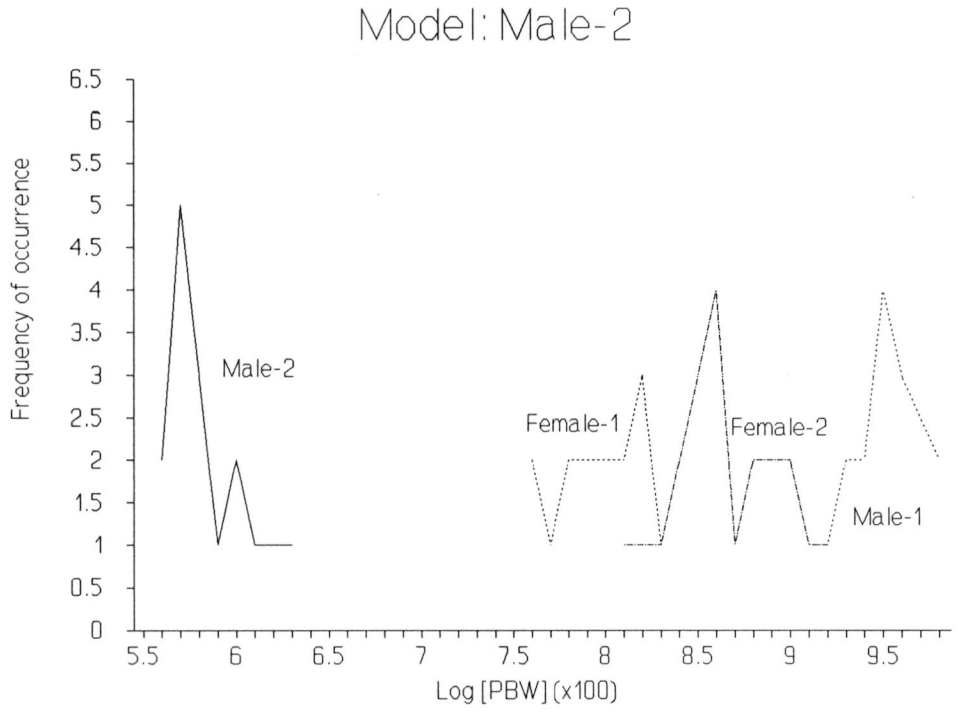


Figure 2


Model: Male-2
Frequency of occurrence
6.5
6
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
Male-2
Female-1
Female-2
Male-1
5.5
6
6.5
7
7.5
8
8.5
9
9.5
Log [PBW] (x100)

Appendix E: Paper prepared for publication in IEEE Journal

HIDDEN MARKOV MODELS APPLIED TO ON-LINE HANDWRITTEN ISOLATED CHARACTER RECOGNITION

Stephan R. Veltman and Ramjee Prasad

Delft University of Technology
Telecommunications and Traffic Control Systems Group
P.O. Box 5031, 2600 GA Delft, The Netherlands
Tel.: +31 15 782417, Fax:+31 15 781774

ABSTRACT:

Hidden Markov models are used to model the generation of handwritten, isolated characters. Models are trained on examples using the Baum-Welch optimization routine. Then, given the models for a vocabulary, unknown characters can be classified using maximum-likelihood classification. Experiments have been conducted, and an average error rate of 6.6% was achieved over a vocabulary consisting of the English alphabet.

I. INTRODUCTION

With the introduction of Integrated Systems Digital Network, multimedia applications are gaining importance. Due to the integration of several services into one single terminal, the user interface, or the man-machine part of the system, becomes a very important aspect of the system.

An interesting step in the evolution of the user interface was the introduction of the pen and a digitizing tablet. Since a pen enables the input of handwritten script, machine interpretation of handwritten text becomes an important field of research. If a machine can reliably interpret handwritten script, applications lie in a wide variety of systems, mainly those who require a high interactivity or the use of direct pointing and manipulation. (Remote) signature verification or person identification can also be an important application [1].

This paper focuses on character recognition. The performance of a system based on Hidden Markov Models (HMM) is evaluated. HMMs have been successfully applied to automatic speech recognition (ASR), and to some extent also to character recognition [2]-[3]. However, these were word-level approaches, while this paper focuses on an isolated character-based approach. Characters are inputted in a freely, unconstrained way using a tablet digitizer. HMMs for each character are obtained by training on example characters using optimization routines. Next, unknown characters are 'scored' on each model, after which the model that has the largest likelihood of having produced the unknown character is chosen (maximum-likelihood classification). The training procedure is computationally very intensive, but can be done off-line, while the classification procedure can be done in real-time. Special attention is given to the HMM symbol-parameter, and the features extracted from the characters.

The organization of this paper is as follows: first, the application of HMMs to the matching problem in general is discussed. Two major problems arise, that are treated consecutively in sections III and IV: the classification and the training problem. Next, tablet quantization is discussed in section V, followed by the formulation of requirements that features extracted

from the quantized characters should meet. Next, section VII discusses the symbol generation (i.e the feature extraction), followed by the discussion of an important design parameter and a preprocessing operation in sections VIII and IX, respectively. In section X the model structure is discussed, and in sections XI and XII empirical work is presented: first optimum model parameters are determined, followed by a complete system performance evaluation. Finally, section XIII presents the conclusions and recommendations for further research.

II. HIDDEN MARKOV MODELS APPLIED TO THE MATCHING PROBLEM

Basically, an HMM is defined as a doubly stochastic process with an underlying stochastic process that is not observable, i.e. hidden, but can only be observed through another set of stochastic processes that produce observable symbols [2]. When the symbols have a one to one correspondence to the states, we speak of a Markov chain. Here, we give a short summary of the theory of application of HMMs to the matching problem according to [4] and [5].

At each discrete time instant t, the HMM emits an observable symbol, and moves to the next state on the basis of state transition probabilities, contained in the **A** matrix. The emission of symbols from each state occurs on the basis of symbol emission probabilities, contained in the **B** matrix. Then, at time t+1, the next symbol is emitted, and so on. Formally, a HMM is defined by the following parameters, see also Table I.

In this paper, only discrete probability densities for the symbols will be discussed, however, it is possible to extend the theory to continuous probability density functions.

For each character, a HMM is formed by training on example characters. Now, two problems can be distinguished in th application of HMMs: the classification and the training problem.

III. CLASSIFICATION

The classification problem is the easier problem in the application of HMMs. For each character, a model is assumed. An unknown observation sequence is 'scored' on each model, and the character corresponding with the model that yields the highest probability of having emitted the sequence is chosen. So, the probability that the sequence **O** was emitted by a given model M, $Pr(\mathbf{O}|M)$, has to be calculated. Since the state sequence that has produced an observation sequence is 'hidden', i.e. not observable, we can either calculate $Pr(\mathbf{O}|M)$ over all possible state sequences, or over the most likely state sequence having emitted the observation sequence. In the present study, we used $Pr(\mathbf{O}|M)$ over all possible state sequences.

Given the observation sequence O and the model M, $Pr(\mathbf{O}|M)$ is given by:

$$\begin{aligned} Pr(O|M) &= \sum_{\text{all possible state seq.}} Pr(O|S,M) \\ &= \sum_{s_1,s_2,\ldots,s_T} \pi_{s_1} b_{s_1}(O_1)\, a_{s_1 s_2} b_{s_2}(O_2) \ldots a_{s_{T-1} s_T} b_{s_T}(O_T) \end{aligned} \tag{4}$$

$Pr(\mathbf{O}|M)$ can be computed very efficiently by the following procedure: A forward probability α_t is defined as the probability of the joint event of emitting the partial observation sequence $O_1,O_2,O_3,\ldots,O_t$ and occupying state s_j at time t:

$$\alpha_t(j) \triangleq Pr(O_1,O_2,\ldots,O_t,\ \text{state } s_j \text{ @ time } t|M) \quad , \quad j=1,2,\ldots,N \tag{5}$$

This leads to a recursive expression:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij} \right] b_j(O_{t+1}) \quad , \quad t=1,2,\ldots,T-1 \qquad (6)$$

In a similar manner, the backward partial probability β_t is defined as starting in state s_i at time t, followed by completing the observation sequence $O_{t+1},O_{t+2},\ldots,O_T$:

$$\beta_t(i) = \Pr(O_{t+1}, O_{t+2}, \ldots, O_T \text{ , state } s_i \text{ @ time } t \mid M) \qquad (7)$$

Again, a recursive relation can be found:

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \quad , \quad t=T-1,T-2,\ldots,1 \qquad (8)$$

Now, $\Pr(\mathbf{O} \mid M)$ can be found by the following expression:

$$\Pr(O \mid M) = \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \quad , \quad 1 \le t \le T-1 \qquad (9)$$

Finally, setting t=T-1 in eq. (6) yields

$$\Pr(O \mid M) = \sum_{j=1}^{N} \alpha_T(j) \qquad (10)$$

IV. TRAINING

The training problem is typically a constrained optimization problem. The likelihood that the example characters are generated by the given model is optimized. This can be done by classical optimization methods, like with Lagrange multipliers, but here we only consider the Baum-Welch algorithm [4].

An initial estimate of the model is made. Next, the parameters of A, B and π are reestimated using the following equations:

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \alpha_t(i)\, a_{ij} b_j(O_{t+1})\, \beta_{t+1}(j)}{\sum_{t=1}^{T-1} \alpha_t(i)\, \beta_t(i)} \qquad (11)$$

$$\bar{b}_{jk} = \frac{\sum_{t \ni O_t = v_k} \alpha_t(j)\, \beta_t(j)}{\sum_{t=1}^{T} \alpha_t(j)\, \beta_t(j)} \qquad (12)$$

$$\bar{\pi}(i) = \frac{1}{Pr(O|M)} \alpha_1(i)\, \beta_1(i) \qquad (13)$$

The model parameters are replaced by these reestimations, and this procedure is repeated until the increase in $Pr(\mathbf{O}|M)$ is arbitrarily small. The reestimation formulas eq. (8) to (10) have the remarkable property that they are guaranteed to increase $Pr(\mathbf{O}|M)$ until a critical point, from which $Pr(\mathbf{O}|M)$ does not change any more. A proof of this property can be found in [4].

The maximum value found for $Pr(\mathbf{O}|M)$ is typically a local maximum. So, it depends on the initial conditions how close the maximum is to the global optimum. Multiple optimizations per character, followed by selecting the model that yields the highest $Pr(\mathbf{O}|M)$, may be necessary for a good performance.

Some practical problems arise that must be solved before the re-estimation formulas can be used. First, the α and ß terms in the forward-backward procedure need to be scaled in order to prevent underflow on a computer. Secondly, if we look at the reestimation formulas we see that whenever a symbol is not observed often enough, a zero symbol probability is

produced for some states. If this is due to the small observation sequence length, care must be taken that all symbol probabilities are reasonable.

Finally, it should be mentioned that the reestimation formulas can be modified to handle multiple observation sequences. Now **K** training observation sequences are given, and

$$\prod_{k=1}^{K} \Pr(O^{(k)} \mid M) = \prod_{k=1}^{K} P_k \qquad (14)$$

is maximized. The Baum-Welch algorithm computes frequencies of occurrences, so each separate frequency may be computed and added together. However, since each individual observation sequence yields a different $\Pr(\mathbf{O} \mid M)$, weighting the numerator and denominator by this factor is necessary. Otherwise, the probabilities from the sequences with a low $\Pr(\mathbf{O} \mid M)$ contribute disproportionally low to the reestimation formulas eq. (8)-(10), resulting in a model that is only optimized for sequences with a high $\Pr(\mathbf{O} \mid M)$. The weighted reestimation formula for the a_{ij} is obtained in the following manner:

$$\bar{a}_{ij} = \frac{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T-1} \alpha_t(i)\, a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T-1} \alpha_t(i) \beta_t(i)} \qquad (15)$$

V. TABLET QUANTIZATION

Since the character representation depends on the digitizing method, we first shortly discuss the writing tablet. The writing tablet has a writable surface of 21x15 cm., with a resolution of 2100x1536 gridlines. So, the resolution requirements according to [7] are met. The sampling process involves the following steps (see fig. 1):

1) registration of pen-down

2) place a rectangular frame around the current pen position, and wait for the trace of the pen to intersect this frame
3) take the intersection the next sampling point, and repeat steps 2 and 3 until a pen-up is registrated.

The frame size can be chosen in terms of gridlines as the first, second or third rectangular frame of gridlines around the current point.

So, the tablet uses space sampling rather than time sampling . The result of the sampling process is an (x,y)-coordinate file of sampling points. The temporal order of the strokes and the dynamographical information of the character is preserved in this manner.We notice that this type of coding is very similar to Freeman coding of line drawings [12].

VI. SYMBOL REQUIREMENTS

The next issue of importance is the choice of features representing a character. The features, or symbols, are contained in the observation sequence **O**. The emission probabilities of the symbols are represented by the **B** matrix, and as we shall see, this will lead in a very logical way to the discussion of what model structure, the **A** matrix, to use.

The requirements imposed on the features are all inspired by the same thought: the interclass separation, i.e. the discriminant power between different characters, should be as large as possible. On the other hand, the intraclass variation, the variation between specimen of the same character, should be minimized. This leads to the following requirements on the features [2]: A good set of features should

1) be independent of translation, rotation, and linear scaling of the curve
2) be chosen so that features do not replicate each other
3) be easy computable
4) preferably (not necessarily) employ the dynamic information provided by the writing tablet

The writing tablet provides a time sequence of directional code vectors. We here use an angular curve description derived from the description used with the definition of Generalized Fourier Descriptors (GFDs) [9]-[12] for three reasons:

1) The two dimensional (x,y)-versus time format is reduced to one dimension; angular direction versus distance along the trajectory of the curve.
2) By nature, angular information is independent of translation of the curve
3) The description according to GFDs involves a size normalization procedure

VII. SYMBOL GENERATION

Let $\phi(s)$ denote the absolute angular direction of the curve as a function of the distance s along the trajectory of the curve. A reasonable independence of rotation is achieved by subtraction of the starting angle combined with a dehooking algorithm. Hooks are due to inaccuracies in pen-down detection and to rapid or erratic motion in placing the stylus on or lifting it off the tablet [7]. In any case, when subtraction of starting angle is used to achieve rotation-independency, these hooks must be removed.

Next, the s-axis is divided into K equidistant steps, and $\phi[k]$ is obtained by linear interpolation. This results in size-independency. Note that linear interpolation corresponds to a first-order reconstruction filter; in spatial domain this means that the curve is reconstructed by circle segments.

Now, the $\phi[k]$-samples can be quantized in order to obtain discrete symbols, resulting in $\phi^*[k]$, the quantized version, using the following mapping:

$$\phi^* = Q\left[\phi\left(\frac{k \cdot s}{L}\right)\right] \qquad (16)$$

VIII. QUANTIZATION

To determine the required accuracy of the quantizer, it is useful to recall the concepts from section VI: the intraclass and interclass variation. Using these concepts, we can formulate the following requirement on quantizer accuracy:

The difference between the quantized representation and its original may not be larger than the average intraclass variation according to some distance criterion.

Two relative error measures are defined, $\epsilon_{\text{quantization}}$ and $\epsilon_{\text{intraclass}}$:

$$\epsilon_{\text{quantization}} \triangleq \frac{\sum_{k=1}^{K} \| \phi[k] - \phi^*[k] \|}{\sum_{k=1}^{K} \phi[k]} \tag{17}$$

$$\epsilon_{\text{intraclass}} \triangleq \frac{\sum_{k=1}^{K} \| \phi_1[k] - \phi_2[k] \|}{\sum_{k=1}^{K} \phi_1[k]} \tag{18}$$

where $\phi[k]$ denotes the non-quantized description, and $\phi^*[k]$ its quantized counterpart. $\phi_1[k]$ and $\phi_2[k]$ are the normalized descriptions of two different specimen of the same character.

To get an indication of the magnitude of the required number of quantization levels, fig. 2 shows a plot of the quantization distortion. Also shown by the dotted line is the intraclass variation for two arbitrary characters.

IX. INTRACLASS-VARIATION REDUCING PREPROCESSING

One preprocessing item has not been discussed yet: a way to treat within-character pen-ups. In all cases, pen-up points are linearly interpolated with the consecutive pen-down points, see fig. 3. In this way, the intraclass variation is reduced, since some people retrace the pen in order to avoid lifting it from the paper.

Only characters 'i' and 'j' are treated differently: here, linear interpolation makes no sense, so an extra symbol ('dot') is added whenever a dot appears. Needless to say, that a dot is easy detectable.

Finally, fig. 4 shows a flowchart of the complete symbol generation procedure.

X. MODEL STRUCTURE

The tablet sampling method captures the temporal order of the handwriting. In a HMM, this temporal order can be imposed by constraining the model structure to left-to-right (LTR). Once an a_{ij} is set to 0, it remains 0 when using the Baum-Welch reestimation formulas. So, by initially setting $a_{ij}=0$ for a disallowed state transition, any model structure can be imposed. Left-to-right models must be trained on multiple sequences. Using one long observation sequence here makes no sense, since once the final absorbing state is reached, the further observations provide no information about earlier states.

Experiments have shown that indeed LTR models perform far better than non-LTR models.

XI. EMPIRICAL DETERMINATION OF OPTIMUM PARAMETERS

Extensive experimental work has been carried out in two stages: first, on a limited vocabulary (typically characters 'a'-'i'), an optimum set of parameters defining the HMMs

was determined. With these optimum parameters (model size, number of symbols, etc.), the performance of the system applied to the complete english alphabet was evaluated.

The main result of the first part of the experiments was that multiple optimizations per character, followed by choice of the model that yields the highest $Pr(\mathbf{O}|M)$, is an absolute necessity for a good system performance. Experiments showed that for some initial estimates no correct model is found, even when the observation sequence is generated by a Markov source using Monte Carlo simulation.

One way of varying the initial estimates is by dividing the unit interval of each parameter into μ sections, followed by optimizations for all possible combinations. For N states, M symbols, and μ sections, the number of combinations is given by

$$C=\mu^{\frac{N^2+N}{2}+N\cdot M} \qquad (19)$$

Only a LTR structure is assumed here. Setting $\mu=3$, N=4 and M=16 already yields $2\cdot 10^{35}$ combinations, clearly not a very attractive computational prospect.

An alternative method is to use a random generator to generate initial estimates. Experiments have shown that for about 50 optimizations per model, sufficiently strong local maxima are found.

Finally, the following model parameters gave the best trade-off between computational intensity and recognition accuracy, see Table II.

XII. SYSTEM PERFORMANCE EVALUATION

With the optimum set of parameters, recognition experiments were conducted on a vocabulary consisting of the complete English alphabet. Per character, 25 training characters were written. In addition, 25 test characters were inputted, resulting in total of $26\cdot 25=650$

test data. The overall error rate, i.e. the average error rate over all characters was determined.

Next, a second writer produced test and training characters. System performance was evaluated for two more cases, see Table III.

In this way, an indication of the ability of the system to capture the multiple-writer variability is achieved.

Finally, the results of the experiments are shown in Table IV.

XIII. CONCLUSIONS

Hidden Markov models provide a solid basis for the recognition of isolated, handwritten characters. The intraclass scatter for a 1-user system is effectively captured by the model. The error rate of 6.6% obtained for a system, trained on its user, can probably be improved by using models with more states. This, however, results in computation times that are not practical at the current state of technology [7]. A better solution would be to use ad-hoc decision rules for ambiguous characters, although this method also suffers from several disadvantages, such as the constraints that are put on the writing style in this way.

An error analysis showed that characters whose descriptions differ only in a small part of the observation sequence make the largest contribution to the error rate. For instance, an 'a' was misinterpreted as an 'd', and an 'q' as an 'g'. Thus, an alternative way of improving the recognition performance would be by either altering the writing style for the poorly recognizable characters, or increasing the discriminant power of the character description in some other way.

At this stage, it is clear that the main application lies in the 1-user system. As the number of users increases, the error rate also increases to a unpractical value.

REFERENCES

[1] PLAMONDON, R., and LORETTE, G., *Automatic signature verification and writer identification - the state of the art*, Pattern Recognition, Vol. 22, No. 2, 1989, pp. 107-131.

[2] KUNDU, A., and BAHL, P., *Recognition of handwritten script: A hidden Markov model based approach*, in Proc. Int. Conf. Acoust., Speech, Signal Processing, 1988, pp. 928-931.

[3] NAG, R., WONG, K.H., and FALLSIDE, F., *Script recognition using hidden Markov models*, in Proc. Int. Conf. Acoust., Speech, Signal Processing, 1986, pp. 2071-2074.

[4] LEVINSON, S.E., RABINER, L.R., and SONDHI, M.M., *An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition*, Bell Systems Technical Journal, Vol. 62, No. 4, 1983, pp. 1035-1074.

[5] LEVINSON, S.E., RABINER, L.R., and SONDHI, M.M., *On the application of vector quantization and hidden Markov models to speaker-independent, isolated word recognition*, Bell Systems Technical Journal, Vol. 62, No. 4, 1983, pp. 1075-1105.

[6] COX, S.J., *Hidden Markov models for automatic speech recognition: theory and application*, Br. Telecom Technol. Journal, Vol. 6, No. 2, April 1988, pp.105-115

[7] TAPPERT, C.C., SUEN, C.Y., and WAKAHARA, T., *The state of the art in on-line handwriting recognition*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 12, No. 8, August 1990, pp. 787-808.

[8] FREEMAN, H., *On the encoding of arbitrary geometric configurations, IRE Trans. Electronic Computers*, Vol. EC-10, June 1961, pp. 260-268.

[9] VIEVEEN, J.W., and PRASAD, R., *Generalised Fourier Descriptors for use with line-drawings and other open curves*, Proc. 6th Scandinavian Conference on Image Analysis, Oulu, Finland, June 1989, pp. 820-827.

[10] WEYLAND, N.B.J., and PRASAD, R., *Characterization of line-drawings using Generalised Fourier Descriptors*, Electronics Letters, Vol. 26, No. 21, October 1990, pp. 1794-1795.

[11] WEYLAND, N.B.J., and PRASAD, R., *Criterion for characterization of line-drawings using Generalised Fourier Descriptors*, Proc. 7th Scandinavian Conference on Image Analysis, Aalborg, Denmark, August 1991, pp. 48-55.

[12] YANG, L., and PRASAD, R., *Recognition of line-drawings based on Generalised Fourier Descriptors*, Proc. 4th IEE Int. Conf. on Image Processing and its Applications, Maastricht, Netherlands, April 1992, pp. 286-289.

LIST OF FIGURE CAPTIONS

Figure 1: Illustration of coding process

Figure 2: Intraclass scatter and scatter due to quantization

Figure 3: Intraclass-variability reduction through interpolation

Figure 4: Complete symbol generation procedure

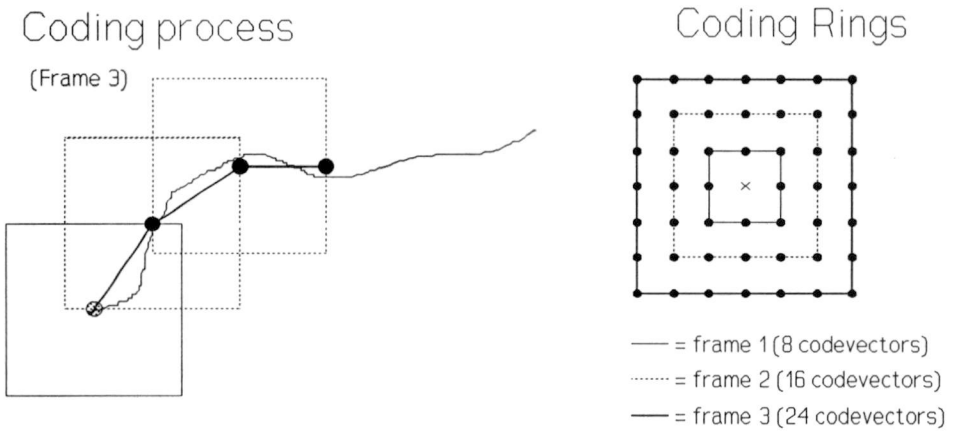

Coding process
(Frame 3)
Coding Rings
= frame 1 (8 codevectors)
= frame 2 (16 codevectors)
= frame 3 (24 codevectors)

Fig. 6

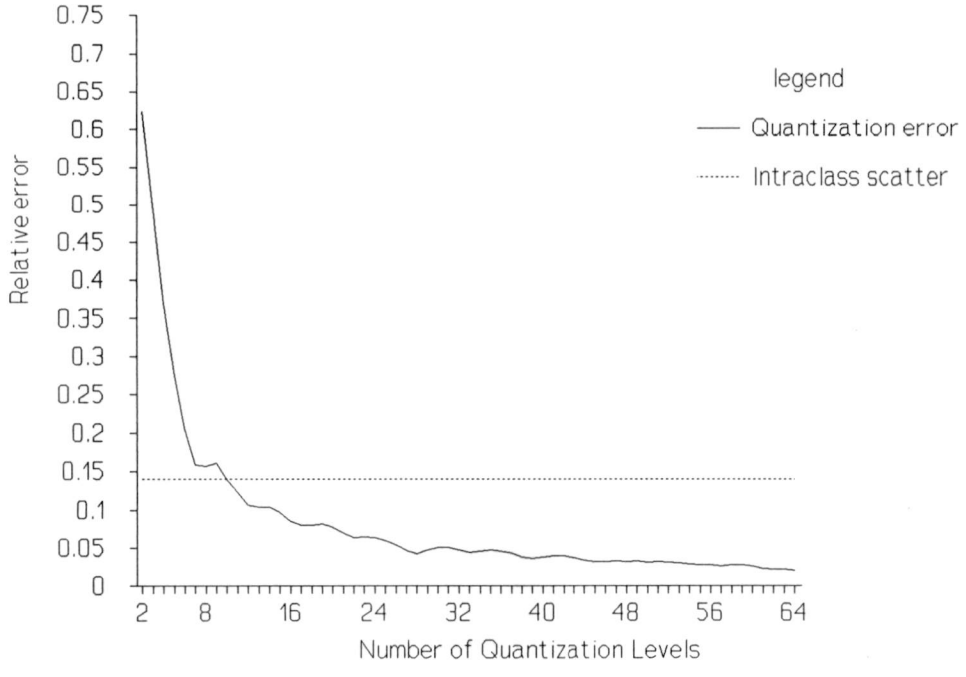

0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
Relative error
2
8
16
24
32
40
48
56
64
Number of Quantization Levels
legend
Quantization error
Intraclass scatter

Fig. 7

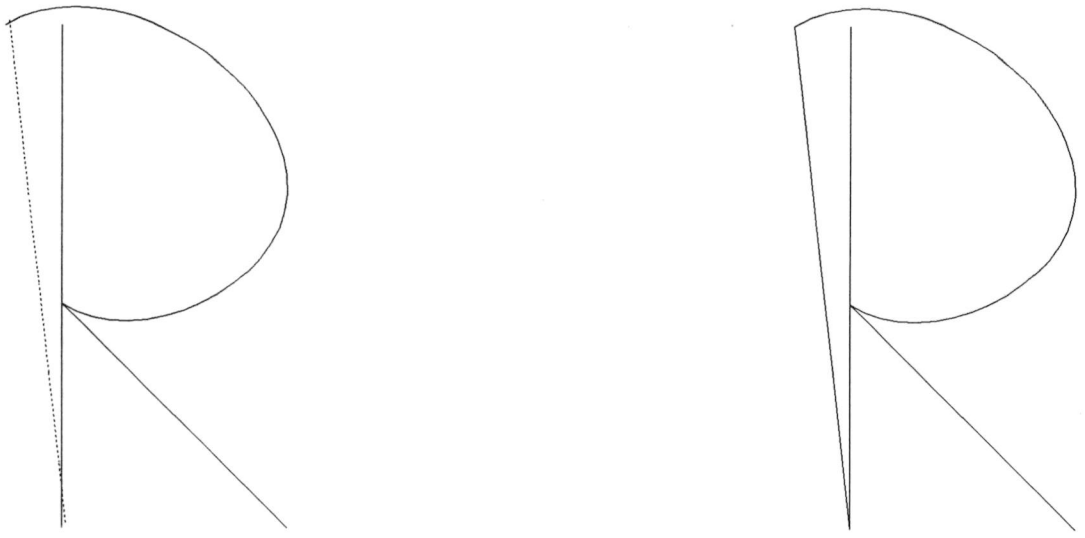


Fig. 8

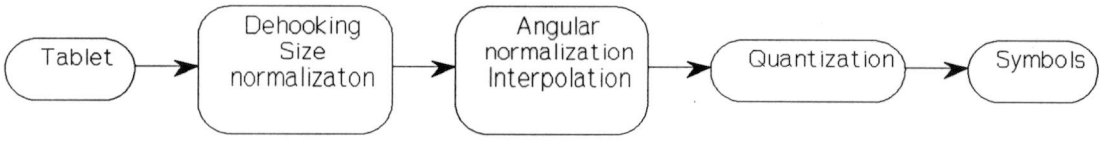

Tablet
Dehooking
Size
normalizaton
Angular
normalization
Interpolation
Quantization
Symbols

Fig. 9

LIST OF TABLE CAPTIONS

Table II

s_j	=	j^{th} state of the HMM	j = 1,2,...,N
v_k	=	k^{th} output symbol in the alphabet	k = 1,2,...,M
a_{ij}	=	Pr(state s_i @ t+1 |state s_i @ t)	i = 1,2,...,N
			j = 1,2,...,N
b_{jk}	=	Pr(output symbol v_k from state s_j)	j = 1,2,...,N
			k = 1,2,...,M
O_t	=	t^{th} observation (analysis vector)	t = 1,2,...,T
$b_j(O_t)$	=	Pr(output observation O_t from s_j)	
	=	b_{jk} when $O_t \rightarrow v_k$	
w_i	=	the i^{th} word in the recognition vocabulary	i = 1,2,...,W

Table III

N=6	,	Left-to-right structure
M=17	,	16 quantization levels and symbol 'dot'
T=64	,	observation sequence length
K=25	,	25 examples per character

50 optimizations per character

Table IV

TRAINING :	USERS :
1 writer	1 writer
1 writer	2 writers
2 writers	2 writers

Table V

TRAINING:	USERS:	ERROR RATE:
1 writer	1 writer	6.6%
1 writer	2 writers	11.4%
2 writers	2 writers	10.9%