Can Markov properties be learned by hidden Markov modelling algorithms?

Jean-Paul van Oosten

August 2010

Master Thesis
Artificial Intelligence
Department of Artificial Intelligence,
University of Groningen, The Netherlands

Supervisors:
Prof. dr. Lambert Schomaker (Artificial Intelligence, University of Groningen)
Dr. Marco Wiering (Artificial Intelligence, University of Groningen)

Abstract

Hidden Markov models (HMMs) are a common classification technique for time series and sequences in areas such as speech recognition, bio-informatics and handwriting recognition. HMMs are used to model processes which behave according to the Markov property: The next state is only influenced by the current state, not by the past. Although HMMs are popular in handwriting recognition, there are some doubts about their usage in this field.

A number of experiments have been performed with both artificial and natural data. The artificial data was specifically generated for this study, either by transforming flat-text dictionaries or by selecting observations probabilistically under predefined modelling conditions. The natural data is part of the collection from the Queen's Office (Kabinet der Koningin), and was used in studies on handwriting recognition. The experiments try to establish whether the properties of Markov processes can be successfully learned by hidden Markov modelling, as well as the importance of the Markov property in language in general and handwriting in particular.

One finding of this project is that not all Markov processes can be successfully modelled by state of the art HMM algorithms, which is strongly supported by a series of experiments with artificial data. Other experiments, with both artificial and natural data show that removing the temporal aspects of a particular hidden Markov model can still lead to correct classification. These and other critical remarks will be explicated in this thesis.

Contents

Chapter 1

Introduction

Hidden Markov models (HMMs) are a statistical machine learning technique, commonly used in areas such as sound recognition, bio-informatics and handwriting recognition (HWR) for classification of time series and sequences. In HWR, HMMs are used to compare unknown instances of written words to previously trained, labelled instances. In sound recognition, spoken words or syllables are used while bio-informatics typically classifies DNA sequences.

HMMs model processes which behave according to the Markov property: the next state is only influenced by the current state, not the past. When modelling natural data, the Markov property is generally assumed (it is therefore also referred to as the Markov assumption).

Another essential assumption is that the underlying processes which are being modelled are stationary. This means that the characteristics of the process do not change over time. Both these assumptions are needed to solve two problems: it is too difficult to deal with a large, unbounded number of past states and changing probabilities which, for instance, determine the next state (Russell & Norvig, 1995).

This thesis has two experimental parts: The first investigates some generic properties of HMMs by creating artificial time sequences with specific properties. The second part uses natural data from a handwriting recognition task to study the use of HMMs in recognition of handwritten text. Consequently, this thesis will have a focus on that area, although the theory is applicable to many other fields. Chapter 2, Theoretical background, will explain all assumptions made due to the focus on HWR.

Markov models are state machines: at each point in time, they are in a certain state. The change of state each time step is determined by the transition probability matrix $\mathbf{A}$. The number of discrete states is denoted with N, and thus the transition matrix is $N \times N$ in size. The transition from state S_i at time t to S_j at time $t+1$ is denoted with a_{ij}. This shows the Markov property: the state at time $t+1$ is only dependent on the state at time t. The probability of being in state S_i at time 1 is denoted with π_i.

When the current state is a physical, directly observable event, the above model suffices and we call it an observable (or explicit) Markov model. But, when the physical event is only partial evidence of the current state, the model needs to be extended to a hidden Markov model. This means that for each state, there exists a function which gives the probability of observing an event in that particular state. This function is usually denoted with $b_i(\ell)$ for the probability of observation ℓ in state S_i. Such situations are for example when sensors are unreliable.

In HWR, the observable events can be represented in a large number of ways. In some cases, the representation in pixel-space is used by moving over the image from left to right with a narrow window. Other cases compute higher order features, for instance by computing the contour of connected components. These representations are usually multi-dimensional and are not usable in the traditional HMM framework, where an observation is assumed to be a single discrete token of occurrence of an event. The problem of multi-dimensional features is solved by either transforming the multi-dimensional feature vector space to an alphabet of discrete tokens, for example by using K-means clustering or self-organising feature maps (Kohonen, 1987), or, alternatively, by allowing an observation x to be a point in $\Re^n$, for which a probability can be computed.

Hidden Markov Models were first introduced to the domain of speech recognition. The canonical reference is Rabiner (1989), which introduces the theory of HMMs as well as their application to speech recognition.

Early in the 1990's, HMMs were introduced to the domain of handwriting recognition. They were used to overcome the problem of segmenting words into characters: Oh, Ha, & Kim (1995) used Viterbi to find the most probable path through a network of character and ligature HMMs. Bengio, LeCun, Nohl, & Burges (1995) used a combination of artificial neural networks (ANNs) and HMMs: the HMMs were used in the post-processing stage to segment the output of the ANNs.

A slightly different approach was used in Bunke, Roth, & Schukat-Talamazzini (1995): a model was created for each word in the lexicon, and during recognition, the model which best described the unknown word would be selected using the techniques described later in this thesis. Dehghan, Faez, & Ahmadi (2000) used a similar approach, where Iranian city names were represented by separate HMMs. Features were first transformed into a discrete observation by using a self-organizing feature map.

HMMs were also used in the closely related field of recognising handwritten numeral strings (useful for, i.e., processing bank cheques): Britto Jr, Sabourin, Bortolozzi, & Suen (2001) represented separate, isolated digits as HMMs, but also used HMMs at the higher level of representing the entire string of digits as HMMs.

On-line handwriting recognition, where the input is from a digital pen or tablet-PC, is usually regarded as easier to solve than off-line handwriting recognition, where the input is from a static image. The temporal information is directly available as part of the input. HMMs are therefore also successfully applied in this field, as shown by Manke, Finke, & Waibel (1995) and Artières, Marukatat, & Gallinari (2007), which used HMMs to represent both strokes and characters.

Despite the abundance of HMM-based solutions in the HWR field, there are also doubts about the use of HMMs. A problem, although not unique to HMMs, is that it usually needs a large amount of training data. Furthermore, as pointed out in van der Zant, Schomaker, & Haak (2008), handwriting is 2-dimensional, containing ascenders, descenders (loops and sticks, protruding from the letters up and down respectively). The ascenders and descenders can extend to the left, where it overlaps other letters in the position on the x-axis. This proves to be a problem when the image is processed strictly from left to right, which is illustrated in Figure 1.1. In the same study, high performances (89%) for word classification are reported with non-Markovian, whole-word classification using Gabor features.

Furthermore, evidence was found that the temporal aspect of language may not

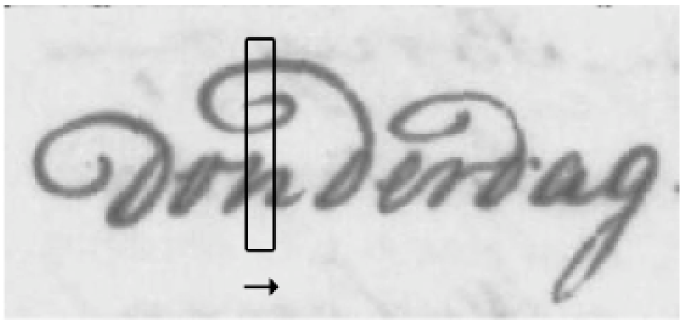


Figure 1.1: *Example of writing where the ascenders of the letters 'd' in the word 'donderdag' extend to the left. When processing this image strictly from left to right, the ascenders are processed before the letters they actually belong to. Image adapted from van der Zant et al. (2008).*

be as distinctive as assumed by most HMM-based solutions. Schomaker (2010) shows an experiment using a simple position-independent coding. The histogram of letter occurrences in each word of the dictionary proves to be a good predictor of word class. For Dutch, in a lexicon of 1144029 words, 95% of the letter histograms are unique.

This last experiment was the motivation for this master's project. It raised the following question: If classification of words in 'ASCII' space can be done with 95% accuracy with a simple position-independent code, how important is the Markov assumption in language and recognition of handwritten text?

A possible way to investigate the importance of the Markov property is by removing the temporal information from an HMM, by 'flattening' the transition matrix $\mathbf{A}$. This means that the probability of moving from any state to any other state always has the same probability, or: with N hidden states, the transition matrix becomes $\mathbf{A} = \{a_{ij} = \frac{1}{N}\}$. If the Markov property is important in, e.g., handwriting, the performance of an HMM-based recognition system should drop dramatically when using such a flattened model.

The experiments will study flattened models, as well as some other types of models, to see whether the classification performance drops when using both artificially generated data and natural data from a handwriting recognition task.

Another approach is that if we *know* that a certain time sequence is generated by a particular (artificial) Markov process, the properties of that process should be detectable by HMMs. A number of experiments in this thesis will investigate this problem by controlling the properties of sequences used to train and test HMMs.

1.1 Thesis structure

After first discussing the theory behind HMMs in chapter 2 and some implementation issues in chapter 3, the two types of experiments are discussed. Chapter 4 discusses the experiments where artificial data is generated to control the properties of the input sequences. Chapter 5 shows the experiments with data from a historical handwriting recognition task. Finally, chapter 6 concludes the thesis with a discussion of the results from the experiments and their consequences.

Chapter 2

Theoretical Background

In this chapter, the theoretical background of hidden Markov models is reviewed. The main body of this review is based on Rabiner (1989), a tutorial on HMM basics and applications in Speech Recognition. The theory is reviewed here step by step, to introduce the notation and the assumptions each implementation has to make (for example: while the most common topology in handwriting recognition is Bakis, it is not the only one, and others might have benefits in certain situations).

There are a few HMM packages available on the internet, which are built with applications for Speech Recognition[1] and Bioinformatics[2] in mind. Since implementation of an HMM framework is not straightforward, the next chapter will introduce some issues which may arise when implementing the theory discussed in this chapter.

The theory can be divided in three sections. The first section introduces the theory of Markov processes and the Markov property, which is the basis of HMMs. This section formalises so-called observable (or explicit) Markov models, which are extended to hidden Markov models in section 2.2. This extension covers observation probabilities, for both discrete and continuous observations. Section 2.3 discusses classification using HMMs, or more specifically, training and testing models.

Finally, the chapter closes with two sections. One details the training of continuous models separate from the general training of HMMs in section 2.3, and the final section provides some methods to integrate HMMs in a handwriting recognition task.

2.1 Markov models

Stochastic processes for which the Markov property holds, are called Markov processes. Stochastic processes are systems which probabilistically evolve in time (Gardiner, 1985, ch. 3). In this thesis, time is regarded as discrete, and the processes are therefore discrete stochastic processes.

A Markov process then, is a stochastic process for which the Markov property holds: Observations in the future are only dependent on the knowledge of the present. More formally, when we describe a Markov process as a system that has N distinct states $S_1, S_2, \cdots, S_N$ and we denote the state at time t with q_t, the Markov property

[1]See for example the HTK Speech Recognition Toolkit, `http://htk.eng.cam.ac.uk/`

[2]For example, the GHMM Library, `http://ghmm.sourceforge.net/`

states that

$$\begin{aligned} & P(q_t = S_j | q_{t-1} = S_i, q_{t-2} = S_k, \cdots) \\ = \quad & P(q_t = S_j | q_{t-1} = S_i) \qquad 1 \leq i, j, k \leq N \end{aligned} \tag{2.1}$$

This system is used as the basis of the explicit Markov models, as well as the hidden Markov models.

The transition matrix $\mathbf{A} = \{a_{ij}\}$ contains all the transition probabilities a_{ij}: the probability of moving from state i to state j, i.e.,

$$a_{ij} = P(q_t = S_j | q_{t-1} = S_i) \qquad 1 \leq i, j \leq N \tag{2.2}$$

Since $\mathbf{A}$ is a probability matrix, it has the following properties:

$$a_{ij} \quad \geq \quad 0 \tag{2.3a}$$

$$\sum_{j=1}^{N} a_{ij} \quad = \quad 1 \qquad 1 \leq i \leq N \tag{2.3b}$$

The probability of being in state i at time 1 is given by the initial state probability π_i:

$$\pi_i = P(q_1 = S_i) \qquad 1 \leq i \leq N \tag{2.4}$$

The transition matrix and initial state probability together make up a Markov model. We call it an *explicit*, or observable, Markov model since the output of the process at time t is its state q_t, which corresponds directly to a physical, observable event.

We can calculate the probability a certain observation sequence O will occur, given an explicit Markov model $(\mathbf{A}, \pi)$. If we take the observation sequence $O = \{S_2, S_3, S_1\}$, we can calculate the probability of this sequence as follows: $P(O|model) = P(S_2, S_3, S_1|model) = \pi_2 \cdot a_{23} \cdot a_{31}$.

Let us examine a simple example. Imagine a model with two states: S_{rain} and S_{dry}, which indicate the weather on a certain day. Furthermore, imagine the probability of having rain on a day after a rainy day is 0.6, while the probability of having a dry day after a rainy day is 0.4 (these probabilities have to add up to 1). The probability of having a dry day follow a dry day is 0.5, which means a dry day is followed by a rainy day half of the time. The probability of having rain the first day is 0.3, while the probability of having a dry first day is 0.7.

Formalised, we have the following transition matrix:

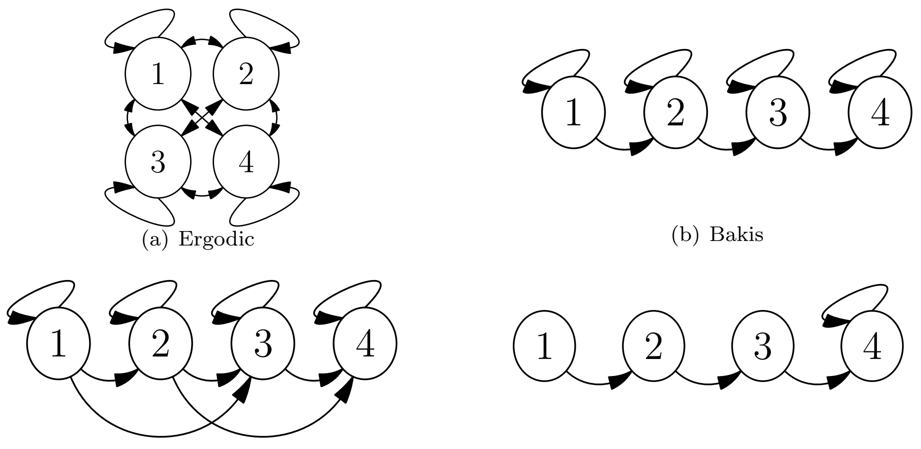
$$\mathbf{A} = \left(\begin{array}{ccc} & S_{rain} & S_{dry} \\ S_{rain} & 0.6 & 0.4 \\ S_{dry} & 0.5 & 0.5 \end{array} \right)$$

and the following initial probabilities:

$$\pi = (0.3 \quad 0.7)$$

So, the probability of having two days of rain followed by a dry day is:

$$P(S_{rain}, S_{rain}, S_{dry} | \mathbf{A}, \pi) = 0.3 \cdot 0.6 \cdot 0.4 = 0.072$$



(a) Ergodic (b) Bakis (c) Skip (d) Forward

Figure 2.1: *Topology examples. The images show all possible transitions in the Ergodic, Bakis, Skip and Forward topologies. The arrows give no indication of the probability of each transition. The flat topology is not displayed since it is equivalent to Ergodic, with equal transition probabilities to all states. The probabilities of all outgoing transitions from a state have to sum up to* 1*, and thus it is easy to see that the transitions in a Forward model are always* 1.

2.1.1 Model topology

So far, no constraints have been placed on the transition matrix. When every state can be reached from every other state in the model in a finite number of steps without repetitive cycles, we call the model *Ergodic*. Strictly speaking, when every state can be reached in a *single* step, i.e., when all transitions have a non-zero probability, the model is 'fully connected ergodic', but is still commonly referred to as being 'ergodic'. See also Figure 2.1(a).

The family of left-right models have the property that all states must be visited in succession (S_2 can not be visited after S_4). This means that the transition matrix is constrained by $a_{ij} = 0$ for all $j < i$.

Bakis models are left-right models as well, with the extra constraint that only recurrent transitions or transitions to the next state are allowed: $a_{ij} = 0$ for all $j < i, j > i + 1$. Figure 2.1(b) shows this relation between states. The Bakis model has its origin in speech recognition, where the modelling of variable phoneme duration is essential.

Models which can skip a few states as well, are constrained by $a_{ij} = 0$ for all $j < i, j > i + s$ where s is the largest jump a state change can make. In this thesis, s is usually 2, and we call these models simply *Skip* models. See Figure 2.1(c). Skip models are more usable in handwriting recognition than in, e.g., speech recognition, since noise is frequently in the form of specks or smudges. Skip models can theoretically jump over this noise by skipping a state.

The *Forward* model is a simple model with no recurrent transitions and only a transition to the next state. The only exception is the last state transition, which must be recurrent. Thus: $a_{ij} = 1$ for $j = i + 1$ and $j = i = N$. $a_{ij} = 0$ in all other cases. This topology is shown in Figure 2.1(d).

Finally, a *flat* model has a transition matrix with $a_{ij} = 1/N$ for all i, j. This means that the probability to move from one state to another is for all states equal. In other words, there is no temporal relation between states.

Models with a Bakis, Skip or Forward topology have an *absorbing* state. The absorbing state is the state where no other transitions are possible than the recurrent transition. So, once in the absorbing state, the model cannot leave that state. The Forward topology always reaches the absorbing state after $N - 1$ transitions.

2.2 Hidden Markov Models

The previous section discussed explicit Markov models, where each state corresponds to a physical, observable event. However, in many cases an observation is only partial evidence for the current state (which means one can not decide which state a system is in by looking at the observation alone). We have to extend the explicit Markov models to hidden Markov models, to accommodate for such cases.

Recall the example from the previous section, with the two states for rainy and dry days. In this example, the state was directly observable: just look outside and see whether it rains or not. However, if we want to monitor the weather in a place far away, and we only have a sensor which measures humidity, we cannot directly observe the weather. The humidity level reported by the sensor is only an indication of the kind of weather.

This 'indication' is modelled by having a probability function for each state in the model. This function returns the probability of observing a particular symbol in that state. In the case of having M distinct observation symbols, we can enumerate the symbols by $v_1, v_2, \cdots, v_M$ (the set V). The probability of producing symbol ℓ in state j is then defined as

$$b_j(\ell) = P(v_{\ell,t} | q_t = S_j) \qquad 1 \leq j \leq N, 1 \leq \ell \leq M \tag{2.5}$$

where $v_{\ell,t}$ is symbol v_ℓ at time t. This is the observation probability function for discrete observations. Continuous observations will be discussed below.

The transition matrix $\mathbf{A}$, the observation probabilities $\mathbf{B}$ and the initial probabilities π make up an HMM, usually abbreviated to $\lambda = (\mathbf{A}, \mathbf{B}, \pi)$.

If the sensor in the example above produces a discrete set of observations, say $Low, Middle, High$, indicating humidity levels, and we know the probability of each observations per state, we can formalise the HMM in this example:

$$\mathbf{B} = \left(\begin{array}{cccc} & Low & Middle & High \\ S_{rain} & 0.1 & 0.2 & 0.7 \\ S_{dry} & 0.6 & 0.3 & 0.1 \end{array}\right)$$

Now, observing Low on a rainy day is very improbable, while it is very probable on a dry day. The probability of observing $High, High, Low$ and the state sequence $S_{rain}, S_{rain}, S_{dry}$ is therefore:

$$\begin{aligned} & P(High, High, Low, S_{rain}, S_{rain}, S_{dry} | \lambda) \\ = \; & \pi_{rain} \cdot b_{rain}(High) \cdot a_{rain,rain} \cdot b_{rain}(High) \cdot a_{rain,dry} \cdot b_{dry}(Low) \\ = \; & 0.3 \cdot 0.7 \cdot 0.6 \cdot 0.7 \cdot 0.4 \cdot 0.6 \end{aligned}$$

Section 2.3.1 will go into more detail on how to efficiently calculate how well a particular model explains a certain observation sequence.

$$\vec{f} \rightarrow \begin{cases} \text{VQ} \quad \rightarrow \text{token } k & \rightarrow \tilde{p}(\vec{f}\,) \approx p(k) \\ \text{Naive Bayes assumption} & \rightarrow \tilde{p}(\vec{f}\,) \approx \prod_i^D p(f_i) \\ \text{Single Gaussian} & \rightarrow \tilde{p}(\vec{f}\,) \approx \mathcal{N}^D(\vec{f}, \vec{\mu}, \Sigma) \\ \text{Gaussian mixtures} & \rightarrow \tilde{p}(\vec{f}\,) \approx \sum_m^M p_m \mathcal{N}^D(\vec{f}, \vec{\mu}_m, \Sigma_m) \end{cases}$$

Figure 2.2: *Illustration of different methods of calculating the probability of a feature vector. The methods are listed from generic to specific: the first method does not make assumptions about the data and models have less parameters to estimate. The first method shows the traditional method of using vector quantization to transform $\vec{f}$ to a discrete token k. The second method shows using the Naive Bayes assumption to combine the probabilities of each feature by a product of probabilities. D denotes the number of dimensions, $|\vec{f}\,|$. The third method shows calculating the probability using a single Gaussian. Σ denotes the full covariance matrix, but might be a single scalar for isotropic Gaussians. The final method shows using a mixture of M Gaussians.*

2.2.1 Continuous observations

The traditional HMM framework allows for discrete observation symbols only, as discussed above. When the feature vector contains continuous observations, there is a number of options, summarised in Figure 2.2. The figure shows four different methods, of which we will discuss two: either quantize the feature vector (i.e., transform the continuous data into discrete codebook indices for which histograms can be tallied), or use continuous probability density functions such as a mixture of Gaussians.

The naive Bayes method has the assumption that each feature is conditionally independent. Each feature can have a different method of computing the probability of that feature. Combination is done by a product of the probabilities for each feature. This method will not be explored further.

While the Gaussian methods show a full covariance matrix Σ, in the specific case of isotropic Gaussians, which can only model circular Gaussian distributions, only a single standard deviation σ is needed. Further explanation can be found below.

With vector quantization, a multidimensional feature vector is transformed into a single discrete token. Frequently used methods include K-means clustering and self-organizing feature maps (SOFM), such as in Schomaker (1993) and Dehghan et al. (2000). A SOFM consists of a (usually two-dimensional) grid of nodes (Kohonen, 1987, 1998). Each node is a model of observations and can be represented with a vector. The best matching node for an unknown observation can be established by calculating the distance (usually Euclidean) to each node. The node with the smallest distance is then the best matching node.

In Schomaker (1993), stroke-based features are described, consisting of 30 (x, y) pairs, which were then converted to a single point: the index of a Kohonen SOFM. Dehghan et al. (2000) used a similar method in conjunction with HMMs, using features based on the word contour.

A large feature vector with continuous features can now be represented by this *discrete* index value, which means the traditional framework can be used without modification for continuous feature vectors.

However, using a SOFM, and vector quantization in general, introduces an extra

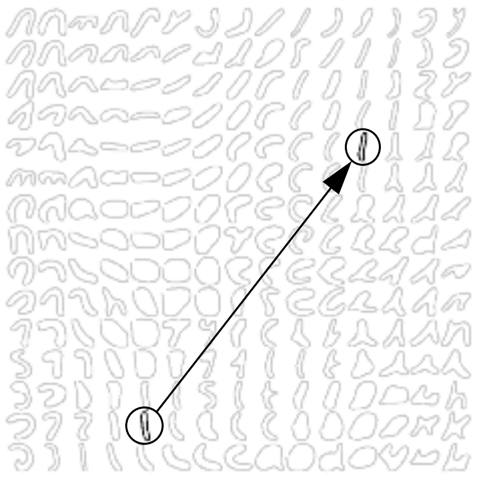


Figure 2.3: *This figure illustrates how a small change in shape leads to a large jump in the discrete representation. Shown is a* 15×15 *self-organizing feature map (SOFM) trained on fragmented connected component contours (FCO^3) extracted from handwritten text of multiple writers. Image adapted from Schomaker et al. (2007)*

step in the classification which requires separate training. Furthermore, a small shift in the feature vector may lead to a large jump in the discrete representation, see Figure 2.3 for an illustration of this effect. The performance of the recogniser will degrade if this jump happens infrequently. This brittleness problem can be solved with sufficient amount of training data. However, the amount of data is usually limited, and other solutions are preferred.

A common solution is replacing the output probability matrix with a probability density function, such as (mixtures of) Gaussians. This removes the separate training of a quantization method. The assumption is that, since there is a gradual transition from one observation to another, continuous observation probabilities will remove the brittleness introduced by the large jumps in discrete observations.

In the continuous observation case, the output probability function will have the following form:

$$b_j(\vec{O}) = \sum_{m=1}^{M} p_{jm} \mathcal{N}(\vec{O}, \vec{\mu}_{jm}, \sigma_{jm}) \qquad 1 \le j \le N \tag{2.6}$$

where M denotes the number of Gaussians in the mixture, $\vec{O}$ is the observation (possibly multi-dimensional), p_{jm} is the mixing probability for state S_j and Gaussian m. $\vec{\mu}_{jm}, \sigma_{jm}$ are the means (a D-vector, D denoting the number of dimensions) and standard deviation. $\mathcal{N}$ is the Gaussian PDF:

$$\mathcal{N}(\vec{O}, \vec{\mu}_{jm}, \sigma_{jm}) = \frac{1}{(\sqrt{2\pi}\sigma_{jm})^D} \exp\left(-\frac{1}{2}\left(\frac{\|\vec{O} - \vec{\mu}_{jm}\|}{\sigma_{jm}}\right)^2\right) \tag{2.7}$$

The observations regarded in this thesis are all 1-dimensional, although the theory should be applicable to multidimensional observations as well. In a handwriting recognition task, examples of multidimensional observations include features with pixel-position and time (x, y, t) or multiple features combined at the same time (and therefore grouped as a single 'observation').

The specific Gaussian PDF in (2.7) can only model Gaussians which have the same standard deviation in all dimensions (i.e., isotropic Gaussians). This means it cannot model elongated clusters, which can be a serious limitation. For non-isotropic Gaussians, a complete covariance matrix needs to be estimated, not just a single standard deviation. In that case, the PDF becomes:

$$\mathcal{N}(\vec{O}, \vec{\mu}_{jm}, \Sigma_{jm}) = \frac{1}{(2\pi)^{D/2}|\Sigma_{jm}|^{1/2}} \exp\left(-\frac{1}{2}(\vec{O} - \vec{\mu}_{jm})^T \Sigma_{jm}^{-1} (\vec{O} - \vec{\mu}_{jm})\right) \tag{2.8}$$

where $|\Sigma|$ is the determinant of Σ, a $D \times D$ covariance matrix.

However, a complete $D \times D$ covariance matrix requires a lot of data to be reliably estimated (there are a lot of parameters to estimate). Furthermore, as can be seen from the PDF, the covariance matrix needs to be invertible, which may not always be the case. Because of these limitations, and for simplicity, the theory of re-estimating Gaussian mixtures in section 2.4 deals only with isotropic Gaussians.

A completely different approach is the use of conditional random fields (CRFs), which are a generalisation of HMMs. CRFs are undirected graphs, where each edge represents a dependency between the nodes. When represented as a chain, each hidden state has a dependency on the previous hidden state, as well as a dependency on the node which represents the input. As a chain, CRFs can be seen as HMMs where the transition probabilities are replaced by functions depending on both the input and the position in the sequence. See for more information Do & Artières (2006).

2.3 Using Hidden Markov Models

Rabiner (1989) describes three basic problems. In this section, the two problems directly related to classification are described: training and testing.

The first problem is the evaluation problem: given a model $\lambda = (\mathbf{A}, \mathbf{B}, \pi)$ and a sequence of observations $O = O_1, O_2, \cdots, O_T$, how do we compute the probability of the observation sequence given the model $P(O|\lambda)$?

The second problem (Problem 3 in Rabiner, 1989) is how we train an HMM, which means adjusting the model parameters $\mathbf{A}, \mathbf{B}, \pi$ to maximise $P(O|\lambda)$.

Problem number 2 in Rabiner (1989) deals with finding the state sequence $q_1 q_2 \cdots q_T$ which best describes the observation sequence. This problem is usually solved with the Viterbi-algorithm and is not discussed in this thesis.

2.3.1 Evaluation

Rabiner (1989) describes two ways of solving the evaluation problem. The first solution is naive, and based on enumerating all possible state sequences. This method is infeasible since it requires too many computations. It does however give insight into the problem. The solution is given as follows:

$$P(O|\lambda) = \sum_{\text{all } Q} P(O|Q, \lambda) P(Q|\lambda) \tag{2.9}$$

where Q is a state sequence. The probability of observing O when Q is the state sequence is: $P(O, Q|\lambda) = P(O|Q, \lambda)P(Q|\lambda)$ with

$$P(O|Q,\lambda) = b_{q_1}(O_1)\cdots b_{q_T}(O_T) \tag{2.10}$$
$$P(Q|\lambda) = \pi_{q_1} a_{q_1 q_2} \cdots a_{q_{T-1} q_T} \tag{2.11}$$

Since this is computationally too heavy (in the order of 10^{72} computations for 5 states and 100 observations, according to Rabiner, 1989), there is another method, called the forward-backward procedure.

If we define the forward variable $\alpha_t(i)$ as the probability of the partial observation sequence up to time t and state S_i at time t, given the model ($\alpha_t(i) = P(O_1 O_2 \cdots O_t, q_t = S_i|\lambda)$), summing over all states at time T gives the desired probability: $P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$.

The forward variable can be solved inductively by the following steps:

1. Initialisation:
$$\alpha_1(i) = \pi_i b_i(O_1) \qquad 1 \leq i \leq N \tag{2.12}$$

2. Induction:
$$\alpha_{t+1}(j) = \left(\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right) b_j(O_{t+1}) \qquad 1 \leq t \leq T-1, 1 \leq j \leq N \tag{2.13}$$

The forward-backward procedure also introduces a backward variable β, which is not used in this problem, but in re-estimating the HMM parameters. β is therefore explained in the next section.

2.3.2 Training

The second problem is how we adjust the model parameters $\mathbf{A}, \mathbf{B}$ and π. This is done using iterative procedures. Here we describe the Baum-Welch method, which is a form of Expectation Maximisation (EM). Unfortunately, there is no known analytical solution.

The model parameters are estimated (or chosen randomly[3]) at first, and then iteratively re-estimated. The re-estimation formulas for the model parameters are defined as

$$\overline{\pi}_i = \text{expected number of times in state } S_i \text{ at time } t = 1 \tag{2.14a}$$
$$\overline{a}_{ij} = \frac{\text{expected number of transitions from } S_i \text{ to } S_j}{\text{expected number of transitions from } S_i} \tag{2.14b}$$
$$\overline{b}_j(\ell) = \frac{\text{expected number of times in } S_j \text{ and observing symbol } v_\ell}{\text{expected number of times in } S_j} \tag{2.14c}$$

[3]although estimation of the initial values of the parameters will yield better results, the experiments in this thesis all chose the initial parameters randomly for efficiency reasons

To compute the re-estimation formulas, we first need to introduce the backward variable. The backward variable $\beta_t(i)$ gives the probability of the partial observation sequence from time $t+1$ to T, given state S_i at time t and the model: $\beta_t(i) = P(O_{t+1}O_{t+2}\cdots O_T|q_t = S_i, \lambda)$. As with the forward variable, we can solve it inductively:

1. Initialisation:
$$\beta_T(i) = 1 \qquad 1 \leq i \leq N \tag{2.15}$$

2. Induction:
$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1})\beta_{t+1}(j) \qquad t = T-1, T-2, \cdots, 1, 1 \leq i \leq N \tag{2.16}$$

Since $\beta_T(i)$ is not defined according to the definition of $\beta_t(i)$ (since the sequence of $T+1$ to T is not defined), a specific initialisation step is needed. The initialisation is somewhat arbitrary, since the backward variable is always explicitly normalised.[4]

Then, with both the forward and backward variable, we can calculate the probability of being in state S_i at time t and S_j at time $t+1$:

$$\begin{aligned} \xi_t(i,j) &= P(q_t = S_i, q_{t+1} = S_j|O, \lambda) \\ &= \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)} \end{aligned} \tag{2.17}$$

The denominator is merely used to keep the desired probability properties $0 \leq \xi_t(i,j) \leq 1$ and $\sum_{i=1,j=1}^{N} \xi_t(i,j) = 1$. The numerator can be explained by looking at the meanings of $\alpha_t(i)$, $a_{ij}b_j(O_{t+1})$ and $\beta_{t+1}(j)$: $\alpha_t(i)$ accounts for all possible paths leading up to state S_i at time t. $a_{ij}b_j(O_{t+1})$ gives the probability of moving from state S_i to S_j when observing the symbol O_{t+1}, and $\beta_{t+1}(j)$ accounts for all the future paths from $t+1$ to T leading from S_j. This is illustrated by Figure 2.4, adapted from Rabiner (1989).

Now, to compute the probability of being in S_i at time t, we simply sum $\xi_t(i,j)$ over all states j:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) \tag{2.18}$$

The reasoning behind this summation is simple: since $\xi_t(i,j)$ defines the probability of being in state S_i at time t and S_j at time $t+1$, we can see that if we take all possible states for j, we get the probability of being in state S_i at time t and any other state at time $t+1$.

[4]However, using a value for the initialisation of $\beta_T(i)$ other than 1, will have a small side effect. Normally, $P(o_1o_2\ldots o_T) = \sum_{j=1}^{N}\alpha_T(j) = \sum_{j=1}^{N}\pi_j b_j(o_1)\beta_1(j)$ must hold, but no longer does when the initial $\beta_T(i) \neq 1$.

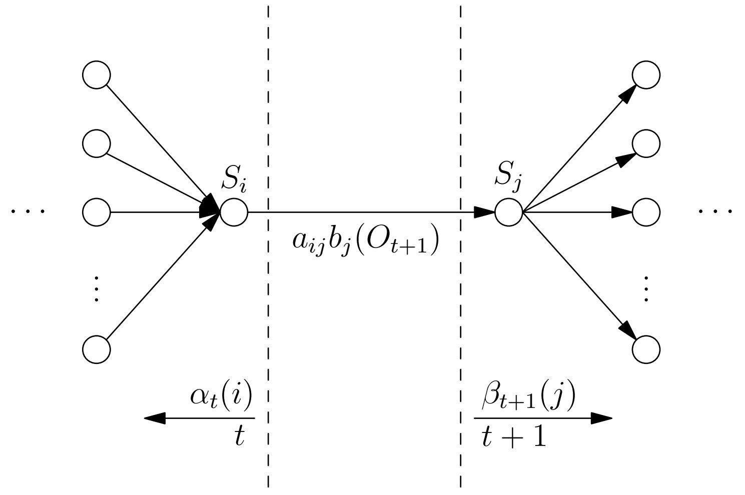


Figure 2.4: *Illustration of the meaning of $\xi_t(i,j)$: The $\alpha_t(i)$ term accounts for the past, leading up to S_i at time t. The term $\beta_{t+1}(j)$ accounts for the future leading from S_j at time $t+1$, while the $a_{ij}b_j(O_{t+1})$ term accounts for the transition from state S_i to S_j. This figure was adapted from Rabiner (1989)*

Another way to calculate $\gamma_t(i)$ is by realising that $\sum_{j=1}^{N} a_{ij}b_j(O_{t+1})\beta_{t+1}(j) = \beta_t(i)$, which means that $\gamma_t(i)$ can also be calculated by

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} \tag{2.19}$$

This is illustrated by Figure 2.5. The relation between $\xi_t(i,j)$ and $\gamma_t(i)$ should now be clear, since the sum over j would account for the paths leading from S_i at time t.

Now, if we sum both γ and ξ over t, we get the expected number of transitions:

$$\text{expected number of transitions from } S_i = \sum_{t=1}^{T-1} \gamma_t(i) \tag{2.20a}$$

$$\text{expected number of transitions from } S_i \text{ to } S_j = \sum_{t=1}^{T-1} \xi_t(i,j) \tag{2.20b}$$

Note that we cannot sum to T, since $\xi_t(i,j)$ has a $\beta_{t+1}(j)$ term, and $\beta_{T+1}(j)$ is obviously not defined.

Using these facts, we can now formalise the re-estimation formulas (2.14):

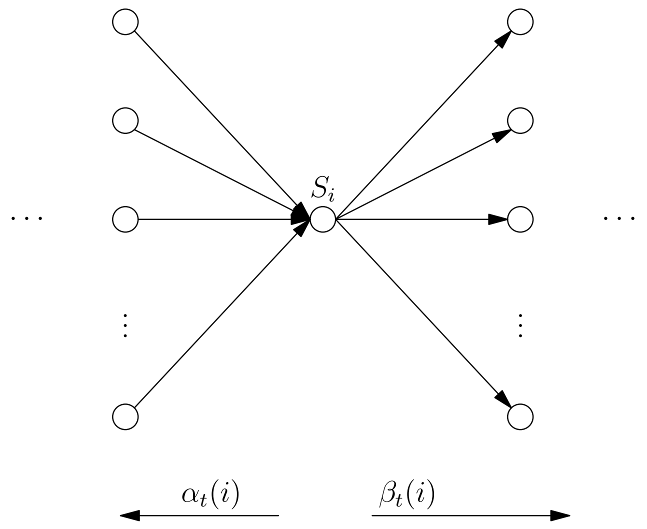


Figure 2.5: *Illustration of $\gamma_t(i)$, the probability of being in state S_i at time t. Again, the $\alpha_t(i)$ term accounts for all the paths in the past up to time t and state S_i, while the $\beta_t(i)$ term accounts for all the paths leading from S_i into the future from time t.*

$$\overline{\pi}_i = \gamma_1(i) \tag{2.21a}$$

$$\overline{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{2.21b}$$

$$\overline{b}_j(\ell) = \frac{\sum_{t\in[1,T]\wedge O_t=v_\ell} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{2.21c}$$

The re-estimated model is $\overline{\lambda} = (\overline{\mathbf{A}}, \overline{\mathbf{B}}, \overline{\pi})$, and we can use this re-estimated model instead of λ in the next iteration. Repeat the re-estimation procedure until no changes occur for a number of iterations, or a total number of iterations is reached (to catch non-converging situations).

It has been proven by Baum & Sell (1968) that either $\overline{\lambda} = \lambda$ or $P(O|\overline{\lambda}) > P(O|\lambda)$. This is a desirable property, since the new model is more likely (or just as likely) to have produced the observation sequence. It also has a drawback, in the sense that this leads to *local maxima*. This means that the initial model (the randomly selected model) has a (potentially) high impact on the final model.

Note that the model topology is preserved, since transitions with a zero probability will remain zero. However, a Flat model is essentially an Ergodic model, with the additional constraint that all transition probabilities must be $a_{ij} = \frac{1}{N}$. In order to retain this property, the transition probabilities of Flat models are not re-estimated.

The following experiment can be performed to test the implementation of the algorithms above. A single observation sequence $[0, 1, 0, 1, 0, 1, 0, 1]$ is learned by both an Ergodic model and a model with a Bakis transition matrix. The Ergodic model will have no trouble learning the sequence and will result in a transition matrix with a probability of 1 for switching to the other state. The observation probabilities will show a probability of 1 for a symbol in state 0 and a probability of 1 for the other symbol in state 1. See the probability matrices in (2.22).

$$\mathbf{A} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2.22a}$$

$$\mathbf{B} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2.22b}$$

The Bakis model will result in different probabilities, since it will not be able to jump back to a previous state. The transition matrix will thus show a probability of 1 to move to the last state, and the observation probability matrix will show a clear preference for one symbol (0 in the case of the sequence above) in the first state, the starting state for all Bakis models, and a probability of 0.5 for both symbols in the second state. See the illustration of these probability matrices in (2.23).

$$\mathbf{A} = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \tag{2.23a}$$

$$\mathbf{B} = \begin{pmatrix} 1 & 0 \\ 0.5 & 0.5 \end{pmatrix} \tag{2.23b}$$

The probabilities shown are of course theoretical probabilities, the actual learned probabilities will most likely not be as accurate. This is due to the EM-algorithm and arriving at a local optimum. Performing this experiment with the implementation used in this thesis, the learned probabilities approach the theoretically correct ones.

2.4 Continuous density observation probabilities

As discussed in section 2.2.1, observation probabilities are represented as Gaussian mixtures in the case of continuous observations. Just as the discrete observation density function, these Gaussian mixtures need to be estimated using Expectation Maximisation as well.

First, estimating Gaussian mixtures from a single set of observations will be discussed, outside the context of HMMs. The adjustments necessary to estimate Gaussian mixtures in a particular state will be introduced in section 2.4.2.

2.4.1 Estimating Gaussian mixture densities with EM

Given T observations $O = (\vec{o}_1, \vec{o}_2, \ldots, \vec{o}_T)$ (possibly multi-dimensional[5]), we can estimate the parameters of M *isotropic* Gaussian functions (in D dimensions). Only isotropic Gaussians are considered; This means this method might not be suitable

[5]As mentioned earlier, the observations in the experiments in this thesis are all 1 dimensional

for all data sources, but it has the advantage that the covariance matrix does not have to be estimated completely.

The final estimated function (mixture) is given by

$$f(O;\theta) = \sum_{k=1}^{M} p_k \mathcal{N}(O;\vec{\mu}_k,\sigma_k) \tag{2.24}$$

with M the number of Gaussians in the mixture, p_k the mixing probability and θ the set of parameters of all the Gaussians:

$$\theta = (\theta_1,\ldots,\theta_M) = ((p_1,\vec{\mu}_1,\sigma_1),\ldots,(p_M,\vec{\mu}_M,\sigma_M))$$

The mixing probabilities are subject to the usual probability properties: $p_k > 0$ and $\sum_{k=1}^{M} p_k = 1$.

The probability that the observation at time t was generated by component k is

$$p(k|t) = \frac{p_k \mathcal{N}(\vec{o}_t;\vec{\mu}_k,\sigma_k)}{\sum_{m=1}^{M} p_m \mathcal{N}(\vec{o}_t;\vec{\mu}_m,\sigma_m)} \tag{2.25}$$

This probability measure has an explicit normalisation component, which means that $\sum_{k=1}^{M} p(k|t) = 1$.

Before the iterative EM-procedure can be started, the set of parameters $(p_k,\vec{\mu}_k,\sigma_k)$ for $1 \le k \le M$ must be (randomly) initialised.

The EM-procedure then consists of two steps: the Expectation, or E-step and the Maximisation, or M-step. The following functions are derived in Tomasi (2005).
E-step:

$$p(k|t) = \frac{p_k \mathcal{N}(\vec{o}_t;\vec{\mu}_k,\sigma_k)}{\sum_{m=1}^{M} p_m \mathcal{N}(\vec{o}_t;\vec{\mu}_m,\sigma_m)} \qquad 1 \le k \le M \tag{2.26}$$

M-step:

$$\hat{\vec{\mu}}_k = \frac{\sum_{t=1}^{T} p(k|t)\vec{o}_t}{\sum_{t=1}^{T} p(k|t)} \tag{2.27}$$

$$\hat{\sigma}_k = \sqrt{\frac{1}{D}\frac{\sum_{t=1}^{T} p(k|t)\|\vec{o}_t - \hat{\vec{\mu}}_k\|^2}{\sum_{t=1}^{T} p(k|t)}} \tag{2.28}$$

$$\hat{p}_k = \frac{1}{T}\sum_{t=1}^{T} p(k|t) \tag{2.29}$$

The re-estimated parameters $\hat{\vec{\mu}}_k, \hat{\sigma}_k, \hat{p}_k$ are then used in the next iteration.

2.4.2 Estimating Gaussian mixture densities in HMMs

In order to use the re-estimation formulae above in HMMs, each state needs a set of parameters: $\vec{\mu}_{jk}, \sigma_{jk}$ and p_{jk} with $1 \le j \le N$. In the re-estimation procedure, the probability of being in state S_j needs to be modelled as well. In section 2.3.2 the measure $\gamma_t(j)$ was introduced as the probability of being in state S_j at time t. $\sum_{t=1}^{T} \gamma_t(j)$ then represents the probability of being in state S_j.

The E-step remains practically the same, apart from a notational difference (namely, it is computed per state). The $\sum_{t=1}^{T} \gamma_t(j)$ term is incorporated in the parameter estimation. This time, $O = \vec{o}_1, \ldots, \vec{o}_T$ is regarded as a single observation sequence.
E-step:

$$p_j(k|t) = \frac{p_{jk}\mathcal{N}(\vec{o}_t; \vec{\mu}_{jk}, \sigma_{jk})}{\sum_{m=1}^{M} p_{jm}\mathcal{N}(\vec{o}_t; \vec{\mu}_{jm}, \sigma_{jm})} \qquad 1 \leq j \leq N, 1 \leq k \leq M \tag{2.30}$$

M-step:

$$\hat{\vec{\mu}}_{jk} = \frac{\sum_{t=1}^{T} p_j(k|t)\vec{o}_t\gamma_t(j)}{\sum_{t=1}^{T} p_j(k|t)\gamma_t(j)} \tag{2.31}$$

$$\hat{\sigma}_{jk} = \sqrt{\frac{1}{D}\frac{\sum_{t=1}^{T} p_j(k|t)\|\vec{o}_t - \hat{\vec{\mu}}_k\|^2\gamma_t(j)}{\sum_{t=1}^{T} p_j(k|t)\gamma_t(j)}} \tag{2.32}$$

$$\hat{p}_{jk} = \frac{1}{T}\sum_{t=1}^{T} p_j(k|t)\gamma_t(j) \tag{2.33}$$

2.5 HMMs in Handwriting Recognition systems

This section provides a quick overview of the ways to use HMMs in Handwriting Recognition (HWR) systems. It will show the most widely used approaches.

Hidden Markov models can be used to classify words as a whole, or character by character. When classifying words as a whole, the evidence (features) is collected on the word-level, and each model is associated with a class. When classifying an unknown word, the model which has the highest probability of having produced the unknown observation sequence is returned as classification. In short:

$$class_k = \operatorname*{argmax}_{\lambda \in \Lambda} P(O_k|\lambda) \tag{2.34}$$

where O_k is the observation sequence for (unknown) word k. Λ denotes the list of all word models. The observation sequence is usually produced by moving over the word with a sliding window.

As discussed in section 2.4, a SOFM can be used to quantize the feature vector. An example of this approach was also already mentioned, Dehghan et al. (2000) used a SOFM to quantize a sliding window over the static image. A sequence of indices of the SOFM (interpreted as discrete tokens) was then classified as a word, using (2.34).

Instead of using a SOFM, an artificial neural network can be used to output sequences of ASCII characters. The input of the neural network is either a sliding window of a static image or a time sequence, e.g., direct output of a pen tablet. The output-sequence of ASCII characters is not clean in the sense that the network will produce a lot of character hypotheses, which are not necessarily correct. This garbled sequence of ASCII characters is then classified using HMMs. An example of such an approach can be found in Schenkel, Guyon, & Henderson (1995). Here, hidden Markov modelling is considered a post-processing method. Another post-processing method, closely related to HMMs, are the variable memory length Markov models, introduced in Guyon & Pereira (1995), which are actually observable Markov models, with a memory of a variable number of characters.

A different approach is to create HMMs per character and create a *lexical graph*. Each node in the graph is a character HMM, while the entire graph represents the word. By finding the optimal path through all character models, the final classification can be obtained. However, segmentation into separate characters is a difficult problem, therefore a stroke or syllable based approach is sometimes preferred.

A variation of this approach is used by Manke et al. (1995), where word models are series of concatenated character models, which in turn consist of three states. An artificial neural network is used to process the input (again, using a sliding window over a static image), and provide estimates of the state probabilities. Based on these estimates, the word is then classified by finding a path through the character models, using the Viterbi algorithm.

Both the list of word-models and the tree of character models can be combined to create word-level HMMs consisting of a tree of character models. The character models represent the states of the higher level word-model, and during classification the most likely sequence of character models can be found using, i.e., the Viterbi algorithm. For examples of this last method, see Oh et al. (1995); Koerich, Sabourin, & Suen (2003).

Finally, the continuous density hidden Markov model approach is used by, e.g., Bertolami & Bunke (2008): each character is modelled with a separate HMM, and an observation is modelled by a mixture of twelve Gaussians in each state. A statistical bigram language model is then used to concatenate all character models.

Chapter 3

Implementation

When trying to implement hidden Markov models with the knowledge presented in the previous chapter, one will encounter at least two main issues, both described in Rabiner (1989). However, the solutions offered to these issues are not sufficient for implementation. Both the solutions offered by Rabiner (1989) and final solutions which were used in the implementation for the experiments are presented in this chapter.

3.1 Scaling α and β

The re-estimation procedure of the transition matrix and the observation probabilities makes use of the forward and backward variables $\alpha_t(j)$ and $\beta_t(j)$. The former gives the probability of the partial observation sequence up to time t and state S_j at time t, given the model λ: $\alpha_t(j) = P(O_1O_2 \cdots O_t, q_t = S_j|\lambda)$. The latter gives the probability of the partial observation sequence from time $t+1$ to time T, given state S_j at time t and the model λ: $\beta_t(j) = P(O_{t+1}O_{t+2} \cdots O_T|q_t = S_j, \lambda)$.

Recall the original definition of $\alpha_t(j)$ (2.13):

$$\begin{aligned}\alpha_1(i) &= \pi_i b_i(O_1) \\ \alpha_{t+1}(j) &= \left(\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right) b_j(O_{t+1})\end{aligned}$$

The calculation of $\alpha_t(j)$ can potentially consist of many products of both transition and observation probabilities. The product of probabilities starts to approach 0 for large t (since $0 \leq p \leq 1$). In theory, this is not a problem, but implementations are usually bound by the precision of floating-point numbers offered by the computer and programming language.

To keep $\alpha_t(j)$ within the range of a floating-point variable, a scaling procedure is needed. The proposed scaling factor is $c_t = \frac{1}{\sum_{i=1}^{N} \alpha_t(i)}$ which is independent of the state and normalises so that $\sum_{i=1}^{N} \hat{\alpha}_t(i) = 1$ (with $\hat{\alpha}_t(i)$ being the scaled variant of $\alpha_t(i)$).

The same scaling factor is used when scaling $\beta_t(i)$. This makes the re-estimation procedure much easier, since most scaling factors will be cancelled out by each other.

We begin scaling at the base case: $\alpha_1(j)$:

$$\hat{\alpha}_1(j) = \frac{\alpha_1(j)}{\sum_{i=1}^N \alpha_1(i)} = c_1\alpha_1(j)$$

where $\hat{\alpha}_1(j)$ is the scaled variant of $\alpha_1(j)$.

In the original version, the induction step uses the previous time step $(\alpha_t(j))$, but in the scaled variant, it uses the scaled version of the previous time step:

$$\hat{\alpha}_{t+1}(j) = c_{t+1}\sum_{i=1}^N \hat{\alpha}_t(i)a_{ij}b_j(O_{t+1}) = c_{t+1}\overline{\alpha}_{t+1}(j)$$

where $\overline{\alpha}_{t+1}(j)$ means that it is based on the scaled variants of the previous time step (see also Rahimi, 2000). We can move the scaling factors for $\hat{\alpha}_t(i)$ out of the summation, since they are not dependent on i:

$$\begin{aligned}\hat{\alpha}_{t+1}(j) &= c_{t+1}\Pi_{\tau=1}^t c_\tau \sum_{i=1}^N \alpha_t(i)a_{ij}b_j(O_{t+1}) \qquad (3.1)\\ &= \frac{\Pi_{\tau=1}^t c_\tau \alpha_{t+1}(j)}{\Pi_{\tau=1}^t c_\tau \sum_{i=1}^N \alpha_{t+1}(i)}\\ &= \frac{\alpha_{t+1}(j)}{\sum_{i=1}^N \alpha_{t+1}(i)}\end{aligned}$$

This is exactly what we wanted to achieve, since now $\sum_{j=1}^N \hat{\alpha}_{t+1}(j) = 1$.

The important thing to remember during implementation is that each iteration in the calculation of $\alpha_t(j)$, one needs to use the scaled version of the previous step, which means after each step the scaling factor needs to be applied.

From (3.1), one can also write:

$$\hat{\alpha}_t(j) = C_t\alpha_t(j) \qquad (3.2)$$

where $C_t = \Pi_{\tau=1}^t c_\tau$.

The same scaling factor c_t is used in scaling $\beta_t(i)$, where, again, the scaling needs to be applied after every step:

$$\begin{aligned}\hat{\beta}_T(i) &= c_T\beta_T(i)\\ \hat{\beta}_t(i) &= c_t\sum_{j=1}^N a_{ij}b_j(O_{t+1})\hat{\beta}_{t+1}(j)\end{aligned}$$

Moving the scaling factors out of the summation as in the $\alpha_t(j)$ case, results in

$$\hat{\beta}_t(j) = D_t\beta_t(j) = \Pi_{\tau=t}^T c_\tau\beta_t(j) \qquad (3.3)$$

We can no longer calculate $P(O|\lambda)$ by summing over $\hat{\alpha}_T(i)$ however, since it now includes a scaling factor as well. We can use the following property:

$$\sum_{i=1}^N \hat{\alpha}_T(i) = C_T\sum_{i=1}^N \alpha_T(i) = 1$$

which is equivalent to $\Pi_{t=1}^{T} c_t P(O|\lambda) = 1$. So $P(O|\lambda) = \frac{1}{\Pi_{t=1}^{T} c_t}$.

However, since this also runs out of the range of a floating-point variable, it is common to compute the log probability:

$$\log\left(P(O|\lambda)\right) = -\sum_{t=1}^{T} \log c_t$$

Classification of a sequence of an unknown word k can be done using argmax (see also (2.34)):

$$class_k = \operatorname*{argmax}_{\lambda \in \Lambda} \left[\log\left(P(O_k|\lambda)\right)\right] \tag{3.4}$$

The scaling procedure has a small effect on the re-estimation procedure, but since we also change the re-estimation procedure in the next section, this effect will be dealt with below. Rahimi (2000) derives the same re-estimation formulae, by first deriving two intermediate variables.

3.2 Multiple observation sequences of variable duration

In order to reliably train an HMM, one needs an appropriate number of instances of observation sequences. More observation sequences means that the HMM can generalise better and has a better chance of successfully classifying new, unknown words. Rabiner (1989) suggests the following changes to the re-estimation formulas (2.21) to train with multiple observation sequences of variable duration:

$$\bar{a}_{ij} = \frac{\displaystyle\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \alpha_t^k(i) a_{ij} b_j(O_{t+1}^{(k)}) \beta_{t+1}^k(j)}{\displaystyle\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \alpha_t^k(i) \beta_t^k(i)} \tag{3.5a}$$

$$\bar{b}_j(\ell) = \frac{\displaystyle\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t \in [1, T_k-1] \wedge O_t = v_\ell} \alpha_t^k(j) \beta_t^k(j)}{\displaystyle\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \alpha_t^k(j) \beta_t^k(j)} \tag{3.5b}$$

K is the number of observation sequences, $\mathbf{O}^{(k)}$ is the kth observation sequence and $\alpha_t^k(i)$ is the $\alpha_t(i)$ based on observation sequence k. Finally, $P_k = P(\mathbf{O}^{(k)}|\lambda)$. Note the use of T_k, implying that each sequence can have a different duration.

Rabiner describes that π does not need re-estimation, since in a Bakis model $\pi_1 = 1$ and $\pi_i = 0$ for $i \neq 1$. However, this only applies to left-right models, such as Bakis and Skip. Below, a more generic definition of π will be provided, which can also be used in Ergodic models.

Even though this is theoretically sound, implementing these formulas will prove difficult. One source of difficulty lies in the fact that we cannot calculate P_k, since we use scaling (Section 3.1). We can however rewrite these formulas to no longer use P_k.

Rahimi (2000) takes a different approach by first deriving two intermediate variables, but achieves the same formulae in the end, expressed in the scaled forward and backward variables $\hat{\alpha}_t(j)$ and $\hat{\beta}_t(j)$.

For rewriting the re-estimation formulae we use the following equalities:

$$\prod_{s=1}^{t} c_s^k = C_t^k \tag{3.6a}$$

$$\prod_{s=t+1}^{T_k} c_s^k = D_{t+1}^k \tag{3.6b}$$

$$C_t^k D_{t+1}^k = \prod_{s=1}^{T_k} c_s^k = C_{T_k}^k \tag{3.6c}$$

$$P(\mathbf{O}^{(k)}|\lambda) = \frac{1}{\prod_{t=1}^{T_k} c_t^k} = \frac{1}{C_{T_k}^k} \tag{3.6d}$$

where c_t^k is the scaling factor $\frac{1}{\sum_{j=1}^{N} \overline{\alpha}_t^k(j)}$.

We now substitute P_k with $\frac{1}{C_{T_k}^k}$:

$$\overline{a}_{ij} = \frac{\sum_{k=1}^{K} C_{T_k}^k \sum_{t=1}^{T_k-1} \alpha_t^k(i) a_{ij} b_j(O_{t+1}^{(k)}) \beta_{t+1}^k(j)}{\sum_{k=1}^{K} C_{T_k}^k \sum_{t=1}^{T_k-1} \alpha_t^k(i) \beta_t^k(i)} \tag{3.7}$$

We can now substitute $\alpha_t^k(i)$ and $\beta_t^k(i)$ by their scaled variants, and use the fact that $\alpha_t(i) = \frac{1}{C_t}\hat{\alpha}_t(i)$ and $\beta_t(i) = \frac{1}{D_t}\hat{\beta}_t(i)$ (see (3.2) and (3.3)):

$$\overline{a}_{ij} = \frac{\sum_{k=1}^{K} C_{T_k}^k \sum_{t=1}^{T_k-1} \frac{1}{C_t^k}\hat{\alpha}_t^k(i) a_{ij} b_j(O_{t+1}^{(k)}) \frac{1}{D_{t+1}^k}\hat{\beta}_{t+1}^k(j)}{\sum_{k=1}^{K} C_{T_k}^k \sum_{t=1}^{T_k-1} \frac{1}{C_t^k}\hat{\alpha}_t^k(i) \frac{1}{D_t^k}\hat{\beta}_t^k(i)} \tag{3.8}$$

Note that $C_t D_t = C_t D_{t+1} c_t$. This leads to the final form:

$$\overline{a}_{ij} = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T_k-1} \hat{\alpha}_t^k(i) a_{ij} b_j(O_{t+1}^{(k)}) \hat{\beta}_{t+1}^k(j)}{\sum_{k=1}^{K} \sum_{t=1}^{T_k-1} \hat{\alpha}_t^k(i) \hat{\beta}_t^k(i) \frac{1}{c_t^k}} \tag{3.9}$$

Rewriting the re-estimation formula for $b_j(\ell)$ is similar and leads to the final form:

$$\bar{b}_j(\ell) = \frac{\displaystyle\sum_{k=1}^{K} \sum_{t\in[1,T_k-1]\wedge O_t=v_\ell} \hat{\alpha}_t^k(i)\hat{\beta}_t^k(i)\frac{1}{c_t^k}}{\displaystyle\sum_{k=1}^{K}\sum_{t=1}^{T_k-1} \hat{\alpha}_t^k(j)\hat{\beta}_t^k(j)\frac{1}{c_t^k}} \tag{3.10}$$

Since we also want to be able to train Ergodic models, we can use the following formula to re-estimate π:

$$\overline{\pi}_i = \frac{\sum_k^K \gamma_1^k(i)}{\sum_j^N \sum_k^K \gamma_1^k(j)} \tag{3.11}$$

where $\gamma_1^k(i) = \frac{1}{c_1^k}\hat{\alpha}_1^k(i)\hat{\beta}_1^k(i)$ which is the probability of being in state S_i at time 1. It can be derived from the non-scaled version:

$$\gamma_1^k(i) = \alpha_1^k(i)\beta_1^k(i)\frac{1}{P(O|\lambda)} \tag{3.12}$$

$$= \hat{\alpha}_1^k(i)\hat{\beta}_1^k(i)\frac{1}{P(O|\lambda)}\frac{1}{C_1^k}\frac{1}{D_1^k} \tag{3.13}$$

$$= \hat{\alpha}_1^k(i)\hat{\beta}_1^k(i)\frac{1}{c_1^k} \tag{3.14}$$

3.3 Continuous observation probabilities with multiple observation sequences

The continuous observation probabilities also need to be updated with multiple sequences. The following formulae show how we can adjust the original formulae (2.30) $\cdots$ (2.33) for updating with multiple observation sequences (since k now represents the sequence, we represent the current Gaussian with n):

E-step:

$$p_j^k(n|t) = \frac{p_{jn}\mathcal{N}(\vec{o}_t^{\,k};\vec{\mu}_{jn},\sigma_{jn})}{\sum_{m=1}^{M} p_{jm}\mathcal{N}(\vec{o}_t^{\,k};\vec{\mu}_{jm},\sigma_{jm})} \qquad 1\le j\le N, 1\le n\le M, 1\le k\le K \tag{3.15}$$

M-step:

$$\hat{\vec{\mu}}_{jn} = \frac{\displaystyle\sum_{k=1}^{K}\sum_{t=1}^{T_k} p_j^k(n|t)\vec{o}_t^{\,k}\gamma_t^k(j)}{\displaystyle\sum_{k=1}^{K}\sum_{t=1}^{T_k} p_j^k(n|t)\gamma_t^k(j)} \tag{3.16}$$

$$\hat{\sigma}_{jn} = \sqrt{\frac{\displaystyle\sum_{k=1}^{K}\sum_{t=1}^{T_k} p_j^k(n|t)\|\vec{o}_t^{\,k} - \hat{\vec{\mu}}_n\|^2\gamma_t^k(j)}{\displaystyle D\sum_{k=1}^{K}\sum_{t=1}^{T_k} p_j^k(n|t)\gamma_t^k(j)}} \tag{3.17}$$

$$\hat{p}_{jn} = \frac{1}{KT}\sum_{k=1}^{K}\sum_{t=1}^{T_k} p_j^k(n|t)\gamma_t^k(j) \tag{3.18}$$

3.4 Summary

This chapter has shown a number of possible issues which might arise when implementing the hidden Markov model algorithms from the previous chapter: α and β, when computed without scaling, quickly become zero due to a large product of probabilities and the limitations of floating point representations on digital computers. The other issue deals with the need of multiple observation sequences to reliably estimate transition, observation and initial state probabilities.

Both issues are mentioned in Rabiner (1989), but the solutions offered are incomplete at best. This makes the implementation not straightforward. One of the issues was the order of computing the scaled variants of α and β: each iteration needs to use the scaled variant of the previous iteration. Another issue, not addressed by the author is the use of $P(O|\lambda)$ in the re-estimation formulae for multiple observation sequences. Both issues have been addressed above, which resulted in formulae equivalent to those derived in Rahimi (2000).

In the next two chapters, a series of experiments will be described and their results shown. The next chapter uses artificially generated data to answer the questions how important the Markov property is in language at a character level, and whether the properties of a Markov process can be reliably learned by the algorithms described in the previous and current chapter. Chapter 5 uses data from a handwriting recognition task to determine the importance of the Markov assumption in handwriting.

Chapter 4

Artificial data experiments

This chapter describes a number of experiments done with artificial data. The experiments try to answer two questions: (i) What is the impact of removing temporal information from an HMM and what does that mean? (ii) Can the properties of Markov processes be reliably learned and modelled by HMMs?

The data sets used in these experiments are artificially created. Some data sets are created by adding noise to words from a flat-text dictionary and then representing these words as series of integers. Other data sets are generated by an artificial Markov process with randomly instantiated transition and observation probabilities.

By generating sequences based on transition and observation probabilities, the nature of data can be manipulated. Two experiments will use this to investigate the role of both the transition matrix and the observation probabilities.

4.1 Modelling a noisy, flat-text dictionary

In order to test the algorithm and perform an initial test, an experiment was performed with simple linguistic data: words consisting of a sequence of ASCII characters. The first experiment also investigates the importance of the Markov assumption in language at the character level, i.e., can the next letter of a word be reliably predicted when either knowing the previous letter or ignoring the past observations?

The experiment uses artificially generated data: Random noise is added to words from a flat-text dictionary. When classifying these 'dirty' words with models with different topologies, the impact of these topologies can be studied. Temporally insensitive models (such as those with a flat transition matrix) should perform badly when temporal ordering is the decisive information source.

This experiment uses data from a Dutch dictionary: a text file containing a large list of Dutch words, although any language should suffice. The words are unique and the string of letters can be seen as a series of observations, changing over time. These observations need to be represented in a numerical format. Since the alphabet (ASCII characters) is already discrete, no discretization is needed. The ASCII-value was taken as the numerical representation of each character, although one could also choose a different enumeration for all characters, as long as this scheme remains consistent throughout the experiment.

Noise was added by adding p random letters to the string, at random positions. The experiment was done for $p = 0, 1, 2, \cdots, 10$, where $p = 0$ is the case where no

Original word	Result
tegenstand	teguenstpanBd YtegenZstacnd
motief	moItbBief Bkmotietf

Table 4.1: *Two examples of words with added noise. p = 3 random letters were added at random positions in the words "tegenstand" (opposition) and "motief" (motive).*

noise was added. Two examples can be found in table 4.1 for $p = 3$.

To keep the duration of the training and testing cycle within reasonable limits, a random sample of 200 words was selected for each fold and each topology (it should be noted that this is a drastic reduction from the 114.029 total words in the Dutch dictionary file). During training, a cut-off of 3000 iterations was used to catch non-converging situations.

Each experiment was conducted 7 times with a random sample of 200 words. The training and test set were generated by randomly adding the p letters 500 times per set, and thus generating 1000 sequences for each word.

A model was trained on each word from the training set, with $N = 3$ hidden states. After training, for each sequence in the test set, the highest ranking model was selected ($\text{argmax}_\lambda[P(O|\lambda)]$, where O is the sequence) as the classification. If this model corresponded with the correct word, the trial was a success.

It should be noted that, due to the way the noise was added, the last letter was untouched and remained at its original, last position. This might give a small boost to performance since more information is available, but there were almost no unique last letters per sample of 200 words.

4.1.1 Results

The low error rates in the presence of substantial noise (Figure 4.1) give a good indication that the algorithm is working properly. The amount of noise effects the performance, although the traditional, temporally sensitive models are less affected by the noise than the Forward and Flat models. Performances for models with a Bakis, Skip or Ergodic topology stay above 99%, even at 10 randomly inserted letters.

The performances of the models with a flat and forward transition matrix show a similar pattern. A possible explanation might be that for both topologies the transition probability values are fixed.

It is remarkable that the models with a flat topology still has a performance of 98.5% correctly classified words, words with $p = 10$ randomly inserted letters. Performances of up to 4 added letters are comparable with the other topologies. The drop in performance for the Forward models can be explained by the fact that after 3 letters, the model has reached the absorbing (final) state, since $N = 3$.

Also consider the average length of a word in the dictionary: 10.4 characters. This means that with $p = 10$, a word doubles in size in the average case.

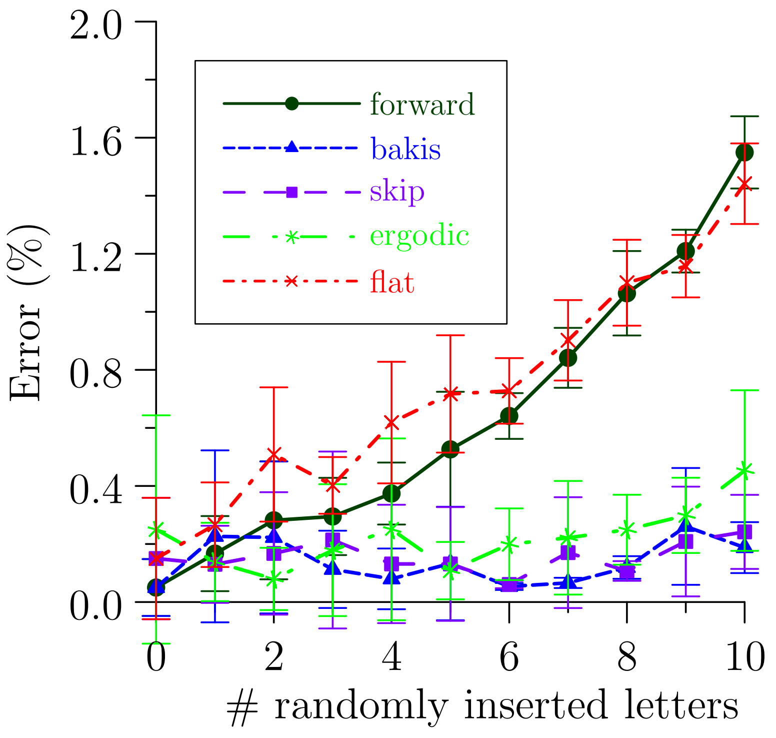


Figure 4.1: *Results of the flat-text dictionary-experiment. Performance of each topology measured in error rate. The x-axis shows the amount of added noise in letters. Reported performances are averages over 7 folds. For each fold and each topology a new random sample of 200 words was selected, for which 1000 sequences were generated, distributed evenly over test and training set. The bands show a 95% confidence interval.*

4.2 Noisy, flat-text dictionary, first and last letters untouched

A commonly known phenomenon, which was getting a lot of attention on the internet a few years back, is that reading is not drastically impaired when letters are rearranged, as long as the first and last letters retain their position. This phenomenon was first researched in 1976 by Rawlinson (for a summary, see Rawlinson, 2007), an example is shown in Figure 4.2.

Since human readers are apparently apt at such a task, it is also interesting to see whether our dictionary-experiment is helped by retaining the first and last letters. Although the task is a bit different from the experiment in Rawlinson (2007) (we add letters, instead of rearranging the existing letters), the performance should improve with regards to the previous dictionary experiment.

Note that new data has to be generated. Again, for each of the 7 folds, a random sample of 200 words was taken. For each word, 1000 sequences were generated by adding random noise. These sequences were split equally over training and test sets. Two examples of words with $p = 3$ added random letters are shown in table 4.2. The

Aoccdrnig to a rscheearch at Cmabrigde Uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoetnt tihng is taht the frist and lsat ltteer be at the rghit pclae. The rset can be a toatl mses and you can sitll raed it wouthit porbelm. Tihs is bcuseae the huamn mnid deos not raed ervey lteter by istlef, but the wrod as a wlohe.

Figure 4.2: *Illustration of the phenomenon found by Rawlinson (2007): reading is not drastically impaired when letters are rearranged, as long as the first and last letters retain their position.*

Original word	Result
eervol	eerASvZol
	ebxehrvol
pret	prasCet
	pGYrTet

Table 4.2: *Two examples of words with added noise. $p = 3$ random letters were added at random positions between the first and last letter in the words "eervol" (honourable) and "pret" (fun).*

methods of training and testing are equivalent to those described in section 4.1.

4.2.1 Results

The results are summarised in Figure 4.3. As can be seen, the performance for the Flat and Forward topologies are much improved. The other topologies already showed an error-rate of about 0.3% at 10 inserted letters. The simple, extra information of the first letter apparently has a positive effect on the Flat and Forward models.

The 95% confidence bands show overlap for all topologies for almost all experiments, except the final experiments with the most noise inflicted words, i.e., at $p = 10$. The confidence bands for the Flat topology do not overlap with the confidence bands of the other topologies. However, this is only one sample of the 55 total experiments, and still the performance is above 99.5%.

Also note that the error rate at 0 inserted random letters (i.e., no noise) is lower than before (0%, instead of 0.1%). This might indicate that the classes in the random samples were separated more (i.e., words were less similar) in the current experiment, which made the classification task easier.

4.3 Artificial Markov processes: varying transition topologies

The previous experiments were based on data from a dictionary. The data in this experiment was generated by an artificial Markov process. The goal of this experiment was to further investigate the effect of topology and whether the underlying properties of the Markov process can be learned and modelled by HMMs. With an artificial Markov process, the properties of the generated data can be controlled.

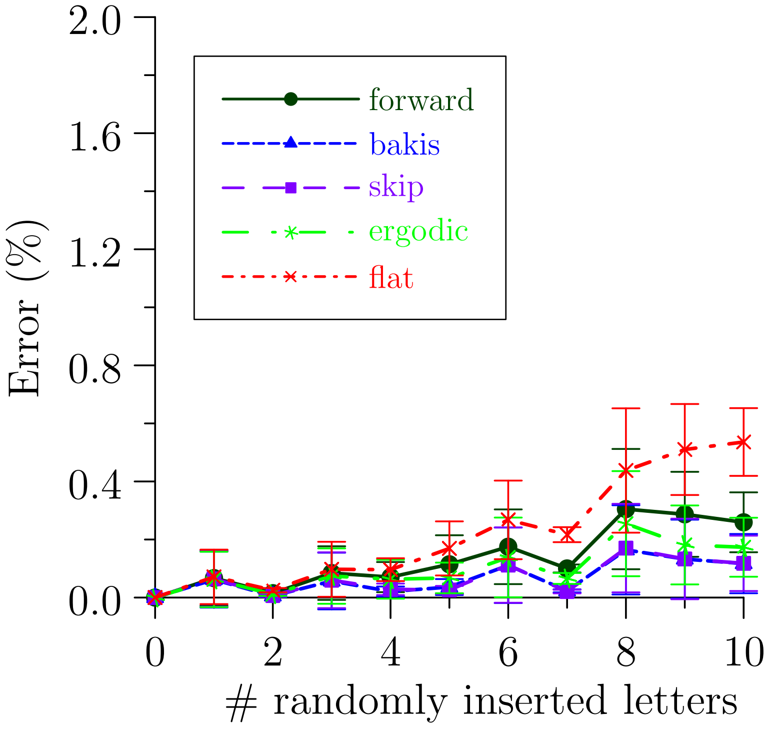


Figure 4.3: *Performance of each topology measured in error rate on the dictionary-task with the first and last letters intact. The x-axis shows the amount of added noise in letters. Reported performances are averages over 7 folds. For each fold and each topology a new random sample of 200 words was selected, for which 1000 sequences were generated, distributed evenly over test and training set. The bands show a 95% confidence interval. The scaling of the y-axis is made the same as in Figure 4.1 for comparison purposes; turn to page 29.*

In this experiment, five data sources were used; each having a different transition matrix with a particular topology. A data source can be thought of as a 'word' generator: According to the transition and observation probabilities, sequences of observations can be generated. The observations are generated randomly according to the distribution in the current state, which is selected solely on the basis of the previous state. This is formalised in Algorithm 4.1 and means that the generated sequences actually have the Markov property and should be very suitable for classification with HMMs.

Each data source consists of (i) a unique transition matrix **A**, generated according to one of the five topologies: Forward, Bakis, Skip, Ergodic and Flat, (ii) a randomly generated, shared observation probability matrix **B** and (iii) an initial state probability distribution π. For the Forward, Bakis and Skip data sources, the initial state probabilities were fixed to $\pi = [1, 0, 0, \cdots]$, while for the Flat and Ergodic data sources they were fixed to $\pi = [\frac{1}{N}, \frac{1}{N}, \cdots]$ (with N the number of hidden states). The observation probability matrix **B** is the same for each data source.

After generating the data sources, individual sequences were generated using Al-

Algorithm 4.1 Generating random sequences based on a data source with **A** a transition matrix, **B** an observation probability matrix and π an initial state probability vector. Random is a hypothetical function which takes a probability vector (distribution) and returns a randomly selected index according to the provided distribution.

```
$q_0 \leftarrow$ Random($\pi$)
$O_0 \leftarrow$ Random($\mathbf{B}[q_0]$)
for $t = 1$ to $T$ do
    $q_t \leftarrow$ Random($\mathbf{A}[q_{t-1}]$)
    $O_t \leftarrow$ Random($\mathbf{B}[q_t]$)
end for
```

Generator model	Resulting sequence											
Ergodic	14	47	42	11	74	16	1	84	49	89	37	⋯
	18	0	57	33	93	49	32	23	94	68	32	⋯
Flat	22	88	46	55	16	79	48	92	93	85	99	⋯
	1	2	12	19	15	55	40	44	14	26	86	⋯

Table 4.3: *Two examples of sequences generated by algorithm 4.1, using models with an Ergodic and Flat transition matrix.*

gorithm 4.1 (Rabiner & Juang, 1993). These sequences then, were used to train and test HMMs. Some example sequences are shown in table 4.3.

Each of the five classes in this experiment corresponded to a topology. Classification then, was done using the argmax function in equation (3.4), where $class_k \in \{\text{Forward}, \text{Bakis}, \text{Skip}, \text{Ergodic}, \text{Flat}\}$.

The experiment was conducted in 7 folds, with $N = 27$ states, $M = 100$ symbols and each sequence having a length of 100 observations. The training and test set for each data source contained 500 sequences. The parameters used for the training of the HMMs were equal to those of the data sources: Number of hidden states, size of the discrete alphabet and topology. So models with a Bakis transition matrix, for instance, were trained on data generated by the Bakis data source. Again, during training, a cut-off of 3000 iterations was used to catch non-converging situations.

4.3.1 Results

Table 4.4 shows how often the test sequences of a data source are classified as one of the five models. As discussed above, the models are named after the generator which generated its training sequences. The percentages which are highlighted in a bold typeface are the classifications which are most common for the given data source. The table shows that 3 out of the 5 data sources are most frequently classified as "Flat", except for the Bakis and Skip sources, which are most frequently classified as "Skip".

In a smaller font, χ^2 distances are reported for the model which was used to generate the sequences and the model in the corresponding column. The distance was the sum of the distance between both transition matrices and both observation probability distributions. Generally, the smallest distances are for models trained on data generated by a source with the same topology, except for the Ergodic data

		Model				
		Forward	Bakis	Skip	Ergodic	Flat
Data Source	Forward	7.5% (77)	0.0% (106)	0.1% (113)	7.5% (132)	**84.9%** (121)
	Bakis	0.0% (94)	40.1% (89)	**59.7%** (98)	0.0% (132)	0.1% (119)
	Skip	0.0% (103)	40.6% (98)	**59.3%** (98)	0.0% (131)	0.1% (116)
	Ergodic	7.2% (132)	0.0% (134)	0.0% (135)	7.4% (133)	**85.4%** (84)
	Flat	6.9% (126)	0.1% (128)	0.0% (128)	7.2% (126)	**85.9%** (72)

Table 4.4: *Results of the experiment for distinguishing between different topologies. The cells first display the percentages how much of the test data generated by the data source was classified as the corresponding model. The second measure shows the χ^2 distance between the data source and the trained model. All numbers are averages over 7 folds with 1000 instances generated per generator and split evenly over test and training set each fold. Each generator had 27 hidden states and 100 symbols. Each sequence was 100 observations long. The highest percentages are displayed in bold.*

source. In that case, the Flat model had a smaller χ^2 distance to the Ergodic data source.

Also interesting to note is that the χ^2 distance for the Flat data source and the Flat model only portrays the distance of the observation probabilities (the same holds for the Forward data source and model) since the transition probabilities for these topologies are fixed.

4.4 Artificial Markov processes: varying observation probabilities

In line with the previous experiment, this experiment also generates data with an artificial Markov process. However, the transition matrix is now fixed across the data sources, while the output probabilities vary. This is interesting since it can tell more about the role of the output probabilities.

The set-up of the experiment is largely the same as the previous experiment, with the noted differences. 5 data sources are generated by generating an output probability matrix and a flat transition matrix (so all 5 sources have the same transition probabilities). Each data source represents a pseudo-word.

1000 sequences are then generated per word, split evenly between train and test set. Each generator has $N = 27$ hidden states and $M = 100$ discrete symbols. Each generated sequence is 100 observations long. Some examples of sequences generated for this experiment can be seen in table 4.5.

For each pseudo-word, a model is trained on the generated training sequences. After all models have been trained, the test sequences are classified as before. This procedure is repeated for each topology we are interested in: Forward, Bakis, Skip, Ergodic and Flat. Again, this experiment was performed 7 times.

It should be noted that no effort has been made to separate the classes. Therefore, this experiment can prove very difficult since the classes might be very close to each other. This will lead to overlap in observations, which is not easily detectable. Contrast this with natural language, where the general rule is to avoid ambiguity and

Generator model	Resulting sequence											
class 1	77	69	13	54	11	78	75	52	78	57	91	···
	70	57	32	32	38	53	20	80	37	80	78	···
class 2	39	8	57	1	24	67	12	7	78	13	5	···
	48	0	97	41	29	16	3	4	33	41	90	···

Table 4.5: *Two examples of sequences generated with different observation probabilities, but with the same flat transition matrix, so the classes can only be distinguished by the observation probabilities.*

Topology	Performance	stdev
Forward	36.7%	0.33
Bakis	36.6%	1.20
Skip	38.8%	1.59
Ergodic	30.3%	1.14
Flat	47.9%	1.80

Table 4.6: *Results of varying observation probabilities, while fixing the generator-topology to Flat, creating 5 different pseudo-words. The reported performances are averaged over 7 folds, with 1000 instances generated per word, divided equally over training and test set. Each model had 27 hidden states and 100 symbols; Each sequence consisted of 100 observations.*

therefore separate classes as much as possible.

4.4.1 Results

The results of this experiment are summarised in Table 4.6. The topology which performs best on this data is the Flat topology, which can be explained by the fact that the trained transition matrix was exactly the same as in the generated samples. The other topologies score between 30% and 40%.

These results can be contrasted by a small experiment where a histogram of token-occurrences is classified using a non-temporal 1-nearest neighbour algorithm. Using an Euclidean distance, this simple method can classify 30% of the test sequences correctly. This result is comparable to the performance of the HMMs with an ergodic topology, while the models with other topologies perform better.

Chapter 5

Natural data experiments

While the previous experiments were based on artificial data, the following experiments are performed with natural data, extracted from a set of handwritten pages. The images used to extract the features from are scans from a single volume of the National Archive of the Netherlands (see also van der Zant et al., 2008, Section 2), all originating from a single writer.

The volume is part of the collection from the Queen's Office ("Kabinet der Koningin" in Dutch, abbreviated as KdK) which contains documents with royal decrees. The writing used in this experiment is part of an index and therefore relatively structured. An example page from this index is shown in Figure 5.1.

The book is annotated by specialist volunteers through a web-based system, line by line. The lines are extracted by an automated procedure using horizontal ink-pixel density and white pixel run-length histograms (see van der Zant et al., 2008).

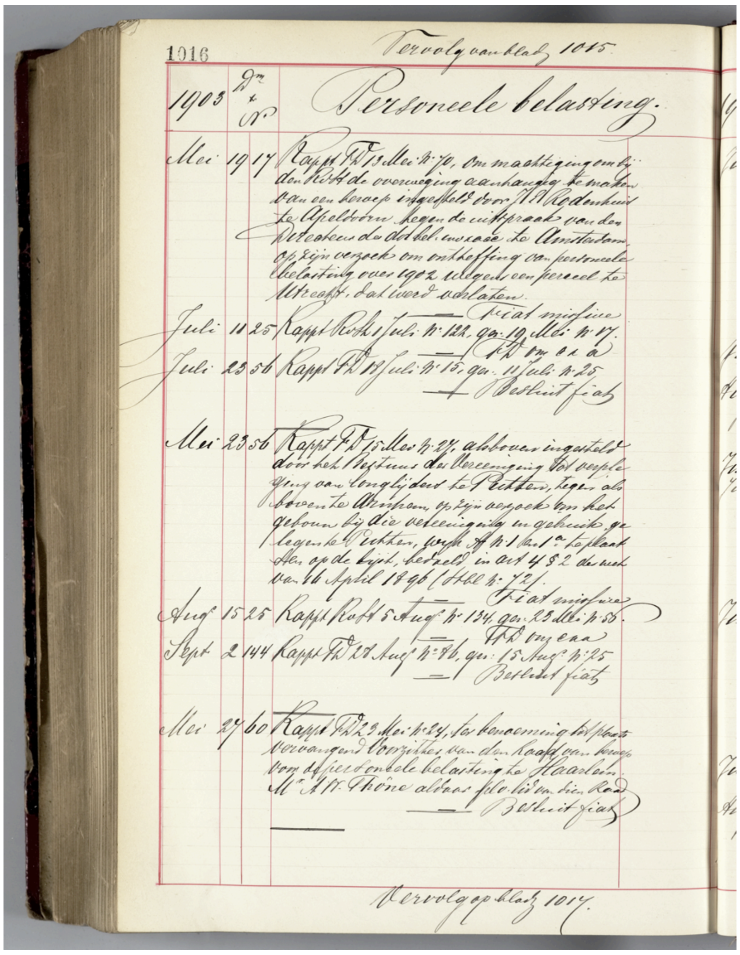
Lines are then segmented into word zones by using white space between connected components. Word zones are conservative estimations of words and can contain, for instance, non-text and parts of ascenders and descenders from neighbouring lines. These word zones are then presented to the previously mentioned volunteers, which label the words.

This chapter discusses two experiments, each using a different method to extract features from the images, which are detailed in Section 5.1. The actual experiments are discussed in Section 5.2, their results summarised in Section 5.3.

5.1 Features

This chapter describes two experiments, which differ on the type of feature used. The first experiment uses fragmented connected component contours (FCO^3) (Schomaker et al., 2007; Schomaker & Bulacu, 2004), while the second experiment uses pixel positions in a sliding window. Both features are quantized using a Kohonen SOFM.

The type of feature influences certain parameters of the experiments, which are discussed in more detail below. For example, since the sequences in the FCO^3 experiment are much shorter, a small amount of hidden states is used.



1016 Vervolg van blad 1015.

1903	Dm. & N°.		Personeele belasting.
Mei	19	17	Rappt F.D. 13 Mei N° 70, om machtiging om bij den RvH de overweging aanhangig te maken van een beroep ingesteld door J.H. Rodenhuis te Apeldoorn tegen de uitspraak van den Directeur der dir. bel. enz. [illegible] te Amsterdam op zijn verzoek om ontheffing van personeele belasting over 1902 wegens een perceel te Utrecht, dat werd verlaten.
Juli	11	25	Rappt RvH 1 Juli N° 122, gev. 19 Mei N° 17. Fiat missive
Juli	23	56	Rappt F.D. 18 Juli N° 15, gev. 11 Juli N° 25. F.D. om [illegible] Besluit fiat
Mei	23	56	Rappt F.D. 15 Mei N° 27, als boven ingesteld door het Bestuur der Vereeniging tot verpleging van longlijders te Putten, tegen als boven te Arnhem, op zijn verzoek om het gebouw bij die vereeniging in gebruik, gelegen te Putten, wijk A N° 1 en 1a, te plaatsen op de lijst, bedoeld in art 4 § 2 der wet van 16 April 1896 (Stbl N° 72).
Aug	15	25	Rappt RvH 5 Aug N° 134, gev. 23 Mei N° 56. Fiat missive
Sept	2	144	Rappt F.D. 28 Aug N° 86, gev. 15 Aug N° 25. F.D. [illegible] Besluit fiat
Mei	27	60	Rappt F.D. 23 Mei N° 24, tot benoeming tot plaatsvervangend Voorzitter van den Raad van beroep voor de personeele belasting te Haarlem, Mr. A.W. Thöne aldaar [illegible] lid van dien Raad. Besluit fiat

Vervolg op blad 1017.

Figure 5.1: *Example page from the collection of the Queen's Office. The experiments in this chapter use data extracted from images like this, which are first converted into black and white images, then cut into line strips. Each line is segmented into word zones. Features are then extracted from these word zones.*

5.1.1 FCO^3

Connected component contours (CO^3) were introduced by Schomaker & Bulacu (2004), and designed with writer identification rather than writing recognition in mind. After extracting the images, blurring and binarising them, connected components were extracted. For each of these connected components, the contour was computed using Moore's algorithm by starting at the left-most pixel and around the connected component in a counter-clockwise direction. Inner contours are not included in the resulting CO^3. The CO^3 is represented as a vector containing 300 X and Y coordinates, so the vector has 600 dimensions.

In cursive writing, the connected components will span multiple characters. A heuristic will be used to segment the connected components into broken connected components (fraglets). The method used in this experiment is "SegM", which seg-

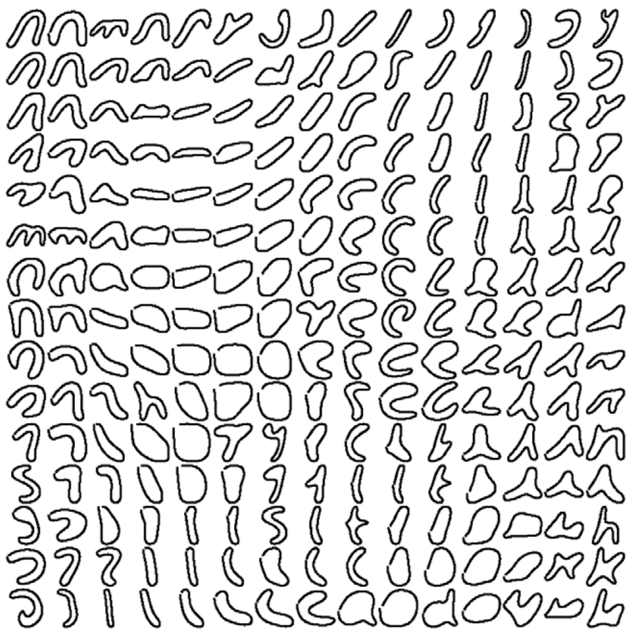


Figure 5.2: *A Kohonen self-organizing feature map (SOFM) of fragmented connected-component contours (FCO^3). Image from Schomaker et al. (2007). While the size of the map shown is 15×15, the map in the experiments was 70×70. Also, this image shows the map based on 300 pages by 100 different writers, while the SOFM used in the experiments was solely based on a single writer.*

ments at 'each vertical lower-contour minimum which is one ink-trace width away from a corresponding vertical minimum in the upper contour' (Schomaker et al., 2007, Section 1.1). This segmentation makes the CO^3 feature usable in a handwriting recognition task.

The 600 dimensions are then reduced to a single index by classification with a Kohonen SOFM, like in Dehghan et al. (2000). For the FCO^3 feature, the dimensions of the SOFM were 70×70, resulting in an alphabet of 4900 discrete symbols. An example of a SOFM with a reduced size (15×15) is shown in Figure 5.2.

5.1.2 Sliding window

The other feature uses a sliding window technique. A window is 4 pixels wide and 51 pixels high, centred around the centroid of black pixels over the entire word zone. A window is moved one pixel to the right each time and all windows are sent to a Kohonen SOFM, returning a single index.

Centering around the centroid with a height of 51 pixels means that parts of big ascenders and descenders are not part of the window. The ascenders and descenders are not essentially removed, but truncated with preservation of the outstanding sticks and (partial) loops.

The images from each window are then discretised by presenting it to a SOFM, with dimensions 25x25. The alphabet is then 625 discrete symbols.

5.2 Experiments

We now have two datasets, corresponding with both features: the first dataset contains sequences of indices from the 70x70 SOFM, applied to the FCO^3 feature, the second dataset contains sequences of indices from the 25x25 SOFM, applied to the sliding windows.

Each dataset is split up into 7 parts, for 7-fold cross-validation. This means that a single part is the test set, the other 6 parts are considered training set.

Since training an HMM requires enough data, we excluded all classes with 50 instances or less in the training set. This left us with 130 classes in the FCO^3 case, with a total of 30 869 labelled instances. The sliding window data set has only 20 classes. The use of all classes with more than 50 instances would computationally be too intensive. The sliding window dataset contained 4 928 labelled instances.

The number of hidden states used in both experiments are chosen with consideration of the average length of each sequence. The average length of sequences in the FCO^3 experiment was 4.4 observations, while the sequences in the sliding window experiment consisted of 65.9 observations on average. The number of hidden states is respectively 3 and 27.

The number of discrete symbols was directly influenced by the dimensions of the SOFM: the size of the FCO^3 alphabet was 4900 symbols, while the sliding window experiment had an alphabet of only 625 symbols.

A cut-off was used to limit the number of iterations. While for the FCO^3 experiment, a maximum of 3000 iterations was chosen, it had to be lowered to 1000 for the sliding window experiment due to the long duration of training models in this experiment. The expectation is that a higher number of iterations might perform a little bit better, but will not result in a large performance improvement.

An HMM is trained for each label in each training set. Testing is then done by calculating $\log P(O|\lambda)$. The classification is

$$\operatorname*{argmax}_{\lambda \in \Lambda} \left[\log P(O|\lambda)\right] \tag{5.1}$$

where Λ is the set of all trained models and O is the test sequence.

Training and testing is done for each fold and for each topology (Forward, Bakis, Skip, Ergodic and Flat), the results are presented in the next section.

5.3 Results

5.3.1 FCO^3

The results of the experiments using the FCO^3 features are summarised in table 5.1. The table shows a performance of 59%–60% for all topologies. There are no significant differences (ANOVA, $p \gg 0.05$) between the topologies. This means that there is also no performance difference between the Flat models and the more conventional topologies as Bakis and Skip.

5.3.2 Sliding window

The results of the sliding window experiment are summarised in table 5.2. Although these results show a significant difference between the topologies (ANOVA,

Topology	Correctly classified (%)	std.dev.
Forward	59.9%	0.9
Bakis	59.9%	0.7
Skip	59.7%	0.8
Ergodic	59.5%	0.9
Flat	59.1%	0.8

Table 5.1: *Results of the FCO^3 experiment. Performances reported are averages over 7 folds, with 130 labels, and at least 51 instances per label in training set. As can be seen, all topologies perform around 60%. Flat models do not perform significantly worse.*

Topology	Correctly classified (%)	std.dev.
Forward	65.0%	2.0
Bakis	75.2%	2.5
Skip	74.8%	1.8
Ergodic	78.5%	1.2
Flat	71.1%	1.3

Table 5.2: *Results of the sliding experiment. Performances reported are averages over 7 folds, with 20 classes, and at least 51 instances per class label in training set. Differences between the topologies are statistically significant ($p < 0.001$) although the difference between Flat and Ergodic is not as dramatic as expected.*

$p < 0.001$), the difference between Ergodic and Flat is not as dramatic as would be expected when temporal ordering was the decisive information source. The difference between the conventional topology Bakis and Flat is even smaller.

Chapter 6

Discussion

Hidden Markov Models are frequently used in fields as speech recognition, bio-informatics and handwriting recognition. HMMs were first used in speech recognition, and became popular in handwriting recognition in the early 1990's. This thesis looked at some generic properties of HMMs as well as their use in handwriting recognition. The questions addressed by the experiments were whether the properties of Markov processes could reliably be learned and modelled by HMMs and how important the Markov assumption is in handwriting recognition.

As discussed in the introduction, the motivation for this study was an experiment detailed in Schomaker (2010), where a position independent coding was used to classify words from a dictionary with the 1-nearest neighbour algorithm. This experiment lead to the first experiment in this study, where noise was added to words from a flat-text dictionary to assess how models with different topologies are affected by noise. The results show that all models perform over 98%. Given that the average word length is 10 letters, and on average the words are doubled in size, this is a high performance. At 10 randomly added letters, the temporally sensitive models such as Bakis and Skip only yield an advantage of around 1.5% on the Flat and Forward models. This suggests that the detailed structure of the transition matrix is maybe not as important as usually expected.

Adding a small bit of extra knowledge can boost the performance even more. Keeping the first letter of the original word intact keeps the performance above 99.5%, even for models with a Flat or Forward transition matrix.

The experiments with the flat-text dictionary data can not answer the question whether the properties of Markov processes can be reliably learned by the HMM-algorithms since the underlying process of generating the Dutch lexicon is not known. The next experiments address this question by using known, randomly sampled models to generate sequences. The underlying properties of these sequences are now exactly known and can be controlled to determine what can be reliably learned.

Five models were generated, each with a different topology but with shared observation probabilities. HMMs were trained on data generated by the models, and classification of all generated test-sequences showed that they were mostly classified as being generated by an incorrect model. Since the observation probabilities were shared by all the generator models, this means that it is hard to separate the sequences only on the basis of the underlying transition probabilities.

The next step is then to examine the role of observation probabilities and whether they can be learned reliably. This was studied by generating data from models which differed only in observation probabilities. The transition matrix was fixed to a Flat topology. The results show that models with a Flat transition matrix have the highest classification performance with 48%. Forward, Bakis and Skip come second with around 37%, while Ergodic models, with 30%, perform at the same level as a simple, non-temporal 1-nearest-neighbour classifier on the observation histogram. This shows how important the role of observation probabilities is. The high performance of the models with a flat transition matrix can be explained by the fact that the generator models used a flat transition matrix as well and that all transition probabilities are fixed. This means that effectively no transition matrix has to be estimated.

While the previous experiments investigated the question whether it is possible to learn underlying properties of Markov processes, the final experiment is designed to investigate the role of the Markov property in handwriting recognition. We used data from a handwriting recognition task to train models for all topologies. When temporal modelling is important in handwriting recognition, models with a flat transition matrix should have a dramatic decrease in performance compared to more conventional and temporally sensitive topologies such as Bakis and Skip.

Using fragmented connected component contours (FCO^3, see also Schomaker et al., 2007), quantized using a SOFM, the classification performed, in all cases, around 60%. Flat models do not perform significantly worse than models with other topologies.

Even though the experiment with data from a sliding window resulted in significant differences between the topologies, the performance drop when using a Flat transition matrix is not as dramatic as might have been expected. It should be noted that the number of lexical classes for the sliding window task was relatively small due to time restrictions, which has a positive effect on the performance. Also, in the FCO^3 experiments, the length of the sequences was small, which has a negative impact on reliably learning properties of the underlying model. These two critical remarks might be indications that the features used in the experiments are not optimal for use with HMMs. However, the conclusions of the experiments with artificial data remain: it is hard to find the underlying properties of a Markov process. Also, if the temporal (left-to-right) ordering would have been the decisive information source in handwriting, a bigger difference between recognition performances would be expected, especially in the FCO^3 experiments.

It should also be noted that the usage of 7-fold cross validation, although it is a commonly accepted evaluation method, can lead to a very small test set. In the worst case, where only 51 instances are available per class, the test set for that class only contains 7 instances. This might have a negative impact on the performance estimation. This is not the case in the artificial experiments, since we generated enough data for both training and testing; each set contained 500 sequences.

An implementation of the HMM algorithms was developed to have complete control over the topology of the transition matrix. Standard implementations, which can be found on the internet, have assumptions, such as topology, hard-coded. Hard-coding assumptions was avoided during implementation. The implementation is easily extended with new topologies and new observation probability functions. That said, implementation was not straightforward: The canonical paper on hidden Markov models (Rabiner, 1989) was not sufficient to implement the algorithms. Chapter 3 discussed the shortcomings as well as their solutions.

Furthermore, each experiment needs to make a number of assumptions on the data. For example, the number of hidden states must be established beforehand. Some studies use different number of hidden states per class, others choose N to be fixed. Also, the topology of the transition matrix must be chosen, which might, or might not have a positive effect on the performance. The type of observation modelling must be determined as well: will the feature vector be quantised using a vector quantisation method, or model the observations using a mixture of Gaussians in each state? Each decision will introduce a different set of parameters to be chosen, and makes not only the implementation, but also the use of HMMs not straightforward.

No extensive experiments were conducted using continuous HMMs, or the Viterbi algorithm. A start has been made with the implementation of estimating mixtures of isotropic Gaussians. Some preliminary tests have been performed, but not to the extent of the other reported experiments. Furthemore, the method of classification only used lists of word models; a different classification method, using path finding with the Viterbi algorithm might provide a better comparison with other existing studies.

The results of the experiments with the artificial Markov process strongly suggest that the Baum-Welch algorithm is insufficient for learning the hidden state transitions. It remains to be seen whether this is a fundamental problem of the hidden Markov modelling paradigm, or is due to the EM-method. Pérez, Piccardi, García, Patricio, & Molina (2007) suggest using genetic algorithms to update the parameters of HMMs for classification of human activity in video. Bhowmik, Parui, Kar, & Roy (2007) show a similar approach for a limited lexicon of West Bengal town names. It would be interesting if the same performance gain could be achieved in the recognition of *unconstrained* handwritten text.

The results of the experiments with the artificial Markov process also suggest that the observation probabilities have an important role. Combined with the previous conclusion, we can conclude the role of the observation probabilities are maybe more important than that of the transition probabilities.

Mention of these issues in the literature is very rare. Artières, Dorizzi, Gallinari et al. (2001) just hint at the importance of the observation probabilities by using a Bakis topology with fixed, predefined transition probabilities. They do not mention the consequences; the hidden state transitions seem to be the heart of hidden Markov models after all. By predetermining the sequence of state transitions, there is no added value of having a *hidden* state.

The fact that the transition probabilities seem of lesser importance than the observation probabilities, is also supported by the results of the experiments with lingual data. Both the experiments with the ASCII-dictionary and the natural data show that the performance of a model with a flat transition matrix is not drastically lower than the performance of models with a more specific topology (such as Bakis or Skip). In the dictionary experiment, a small bit of extra knowledge is enough to bring the error rate down to 1% at most.

As for the experiments with the data from the handwriting task, the results show that using a flat transition matrix does not drastically degrade performance. Coupled with the fact that van der Zant et al. (2008) report high performances for word classification with a non-Markovian classifier, this suggests that finding a robust feature should be the top priority.

The idea of having a simple model of state transitions and observation probabilities is highly attractive. However, we have shown in this thesis that the use and implementation of the hidden Markov modelling algorithms is not straightforward.

The use of HMMs is not straightforward due to the number of parameters needed to be predetermined such as the number of hidden states and the modelling of observations. Moreover, the Baum-Welch algorithm has difficulty finding the properties of time sequences, and removing the temporal aspects of a particular model still yields an acceptable result. There have been promising results using genetic algorithms to learn the parameters of a model, instead of expectation maximisation. Finally, the fact that high recognition performances can be achieved without Markovian classifiers recommends a greater emphasis on finding robust features.

Acknowledgements

First of all, I would like to thank my supervisor professor Dr. Lambert Schomaker, for his great advise and enthusiasm and for always finding time to discuss interesting angles and new insights, despite his busy schedule. I also would like to thank my second supervisor Marco Wiering for his invaluable support.

My thanks go out to Wilmer Joling and Maurice Voogd for proofreading drafts of my thesis. I would also like to thank Richard, Auke-Dirk, Robert and all the other students working on their master thesis for creating a great working environment.

Finally, I would like to thank my parents for their unwavering support and of course my girlfriend Mitsai for supporting me, putting up with me and keeping me motivated during all my years at the university of Groningen.

Bibliography

Artières, T., Dorizzi, B., Gallinari, P., et al. (2001). From character to sentences: A hybrid Neuro-Markovian system for on-line handwriting recognition. *Hybrid Methods in Pattern Recognition", H. Bunke, A. Kandel (eds.), World Scientific Pub. Co*, (pp. 1–27).

Artières, T., Marukatat, S., & Gallinari, P. (2007). Online handwritten shape recognition using segmental hidden markov models. *IEEE transactions on pattern analysis and machine intelligence*, *29*(2), 205–217.

Baum, L., & Sell, G. (1968). Growth functions for transformations on manifolds. *Pac. J. Math*, *27*(2), 211–227.

Bengio, Y., LeCun, Y., Nohl, C., & Burges, C. (1995). Lerec: A NN/HMM hybrid for on-line handwriting recognition. *Neural Computation*, *7*(6), 1289–1303.

Bertolami, R., & Bunke, H. (2008). Hidden Markov model-based ensemble methods for offline handwritten text line recognition. *Pattern Recognition*, *41*(11), 3452–3460.

Bhowmik, T., Parui, S., Kar, M., & Roy, U. (2007). HMM parameter estimation with genetic algorithm for handwritten word recognition. *Pattern Recognition and Machine Intelligence*, (pp. 536–544).

Britto Jr, A., Sabourin, R., Bortolozzi, F., & Suen, C. (2001). A two-stage HMM-based system for recognizing handwritten numeral strings. In *Proceedings of the Sixth International Conference on Document Analysis and Recognition, 2001.*, (pp. 396–400).

Bunke, H., Roth, M., & Schukat-Talamazzini, E. G. (1995). Off-line cursive handwriting recognition using hidden markov models. *Pattern Recognition*, *28*(9), 1399 – 1413.

Dehghan, M., Faez, K., & Ahmadi, M. (2000). A hybrid handwritten word recognition using self-organizing feature map, discrete HMM, and evolutionary programming. *IEEE*, (pp. 515–520).

Do, T.-M.-T., & Artières, T. (2006). Conditional Random Fields for Online Handwriting Recognition. In *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft.

Gardiner, C. (1985). *Handbook of stochastic methods*. Springer Berlin.

Guyon, I., & Pereira, F. (1995). Design of a linguistic postprocessor using variable memory length Markov models. In *International Conference on Document Analysis and Recognition*, (pp. 454–457). Published by the IEEE Computer Society.

Koerich, A., Sabourin, R., & Suen, C. (2003). Lexicon-driven HMM decoding for large vocabulary handwriting recognition with multiple character models. *International Journal on Document Analysis and Recognition*, *6*(2), 126–144.

Kohonen, T. (1987). Adaptive, associative, and self-organizing functions in neural computing. *Appl. Opt.*, *26*(23), 4910–4918.

Kohonen, T. (1998). The self-organizing map. *Neurocomputing*, *21*(1-3), 1–6.

Manke, S., Finke, M., & Waibel, A. (1995). NPen++: A Writer Independent, Large Vocabulary On-Line Cursive Handwriting Recognition System. In *Proceedings of the Third International Conference on Document Analysis and Recognition, 1995.*, vol. 1.

Oh, S., Ha, J., & Kim, J. (1995). Context dependent search in interconnected hidden Markov model for unconstrained handwriting recognition. *Pattern Recognition*, *28*(11), 1693–1704.

Pérez, Ó., Piccardi, M., García, J., Patricio, M., & Molina, J. (2007). Comparison Between Genetic Algorithms and the Baum-Welch Algorithm in Learning HMMs for Human Activity Classification. *Applications of Evolutionary Computing*, (pp. 399–406).

Rabiner, L., & Juang, B. (1993). *Fundamentals of speech recognition*. Prentice hall.

Rabiner, L. R. (1989). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, *77*(2), 257–286.

Rahimi, A. (2000). An Erratum for "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition". `http://xenia.media.mit.edu/~rahimi/rabiner/rabiner-errata/rabiner-errata.html`, as of may 2010.

Rawlinson, G. (2007). The significance of letter position in word recognition. *IEEE Aerospace and Electronic Systems Magazine*, *22*(1), 26–27.

Russell, S., & Norvig, P. (1995). *Artificial intelligence: a modern approach*. Prentice hall Englewood Cliffs, NJ.

Schenkel, M., Guyon, I., & Henderson, D. (1995). On-line cursive script recognition using time-delay neural networks and hidden Markov models. *Machine Vision and Applications*, *8*(4), 215–223.

Schomaker, L. (1993). Using stroke-or character-based self-organizing maps in the recognition of on-line, connected cursive script. *Pattern Recognition*, *26*(3), 443–450.

Schomaker, L. (2010). Some problems with a Scotsman and a Russian: On the limitations of Bayesian and hidden-Markov models. In prep.

Schomaker, L., & Bulacu, M. (2004). Automatic writer identification using connected-component contours and edge-based features of uppercase western script. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *26*(6), 787–798.

Schomaker, L., Franke, K., & Bulacu, M. (2007). Using codebooks of fragmented connected-component contours in forensic and historic writer identification. *Pattern Recognition Letters*, *28*(6), 719–727.

Tomasi, C. (2005). Estimating gaussian mixture densities with EM: A tutorial. `http://www.cs.duke.edu/courses/spring04/cps196.1/handouts/EM/tomasiEM.pdf`, as of may 2010.

van der Zant, T., Schomaker, L., & Haak, K. (2008). Handwritten-word spotting using biologically inspired features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (pp. 1945–1957).