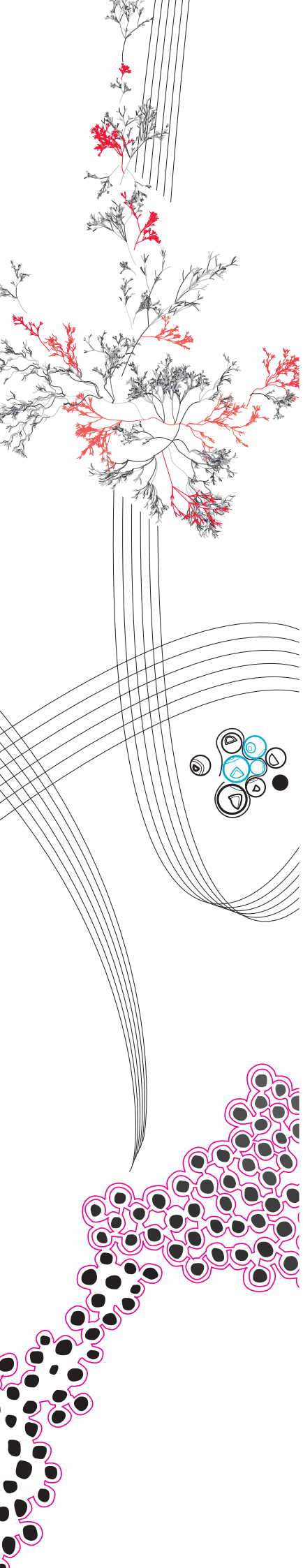


MASTER THESIS

Activity Classification through Hidden Markov Modeling

Thijs Tromper
March 1, 2016

Faculty of EEMCS
Department of Applied Mathematics
Chair Applied Analysis

Assessment committee:
prof. dr. S.A. van Gils (UT)
L.L. Zeune MSc (UT)
C. Baten MSc (RRD)
prof. dr. H.J. Zwart (UT)

UNIVERSITY OF TWENTE.

Abstract

The goal of this study is to improve upon an existing activity classifier software model called FusionAAC, previously developed at Roessingh Research and Development. FusionAAC applies a technique called Hidden Markov Modeling to automatically and independently classify human activities. It mainly uses kinematic motion sensor data as input, but other input data types are also possible. The FusionAAC software is based upon the Hidden Markov Toolkit 3.4, implemented in a Labview environment.

Until now, FusionAAC has been largely used as a 'black box' tool. This study looks to open up the black box by taking a closer look at the underlying algorithms. The deeper understanding is used to improve various aspects of the software, most notably the training process of the classifier. For the training process, the way the training data should be organized and different parameter initialization methods are discussed. Other areas of focus are preprocessing of the input data through application of principal components analysis, and the avoidance and removal of errors (most notably false positives) from the classification results. The improvements are tested on a data set containing various lifting activities.

Preface

This report covers my combined internship and master's research project, carried out towards receiving a master's degree in Applied Mathematics at the University of Twente. The main body of work was done at Roessingh Research and Development (RRD) in Enschede. I would like to thank my daily supervisor Chris Baten for putting his trust in me and giving me the opportunity to put my theoretical knowledge to practical use. I would further like to thank Leonie for her invaluable input which greatly helped me translate our research questions into mathematics.

Thijs Tromper
Enschede, March 1, 2016

Contents

List of symbols

HMMs

Symbol	Description
t	Clock time
T	Total number of clock times, $1 \leq t \leq T$
$\mathcal{S}$	State space of a Markov chain
N	Total number of states in a Markov chain
q_i	State i in a Markov chain, $1 \leq i \leq N$
X_t	State of a Markov chain at clock time t
$\mathcal{X}$	Markov chain state sequence $X_1 \cdots X_T$
$\boldsymbol{\pi}$	Initial state probability distribution of a Markov chain, see (2.3)
$\mathbf{A}$	Transition probability matrix $\{a_{ij}\}$ of a Markov chain, see (2.1)
$\mathcal{V}$	Space of observable signals
M	Total number of distinct observable signals
v_k	Discrete observable signal k, $1 \leq k \leq M$
O_t	Observable signal at clock time t
$\mathcal{O}$	Sequence of observations $O_1 \cdots O_T$
$\mathcal{O}^{t_1:t_2}$	Sequence of observations from time t_1 to t_2: $O_{t_1} \cdots O_{t_2}$
$\mathbf{B}$	Vector of observable signal distributions $\{b_j(\cdot)\}$, see (2.2)
λ	Compact notation for a HMM, $\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$
$\alpha_t(j)$	Forward variable, see (2.5)
$\beta_t(j)$	Backward variable, see (2.6)
$\gamma_t(j)$	Probability state q_j at time t, see (2.8)
$\delta_t(j)$	Probability partial best path ending at state q_j at time t, see (2.11)
$\Psi_t(j)$	Most likely state q_j at time t in partial best path, see (2.12)
$\xi_t(i,j)$	Probability of state q_i at time t and state q_j at time $t+1$, see (2.13)

HTK

Symbol	Description
S	Substitution error
D	Deletion error (also called false negative)
I	Insertion error (also called false positive)
p	Word insertion penalty

Acronyms	Meaning
HMM	Hidden Markov Model
MLE	Maximum-Likelihood Estimation
DMD	Duchenne Muscular Dystrophy
AT	Assistive Technology
ADL	Activities of Daily Life
HTK	Hidden Markov model Toolkit
TV	Total Variation
MAP	Maximum A-posteriori Probability

Formulas	Description and use
$P(A\|B) = \frac{P(A,B)}{P(B)}$	Conditional probability (Bayes' theorem): the probability that an event A occurs, given that another event B has occurred. Plays an important role in most derivations in this report.

Chapter 1

Introduction

1.1 Background

Over the past two decades, inertial sensing technology has seen considerable improvement. Accelerometers and gyroscopes have not only become much more accurate, it is now also possible to build cheap, highly miniaturized sensors. These developments give rise to a wealth of new research fields where motion sensing can be applied.

One of these fields is rehabilitation technology. In rehabilitation, a therapist often needs some way to quantify function and progress of a patient. To this end, it is important for the therapist to be able to observe patients in a setting that comes as close to their natural environment as possible, while at the same time the observation is as objective and accurate as possible. For example, it is better to observe patients walking normally compared to them walking on a treadmill, but it might be easier to quantify motion on a treadmill. Some rehabilitation therapists might have access to a motion analysis laboratory, but this is still not ideal. The walking space in laboratories usually is no more than ten meters, which makes it difficult to measure more than two gait strides in their entirety. On top of this, the setting (artificial floor, pressure to perform) is far from optimal. In other words, to make it easier to provide the best possible care, it would be much better to be able to observe patients accurately and objectively during normal daily life.

This is where modern sensors come into play. The new generation of inertial sensors show great potential to measure patients during normal daily life. Lightweight, easy to position sensors, that are easily worn underneath clothing, are becoming the standard. Not only are these sensors not affecting the wearer's ability to perform normal daily activities, the possibility to wear them underneath clothing greatly increases social acceptance. The challenge now is to achieve the same level of accuracy that is obtained in a laboratory setting.

In this light, and in the wake of increasing possibilities and applications of wearable sensing, the importance of finding smart ways to handle the recorded data grows accordingly. For example, eight hours of back load exposure data is not interpretable without highly detailed knowledge of the actual activities of the person

being analyzed. Typically, existing automated activity recognizers are only able to distinguish a few general activities. In practice this means that any recorded data has to be manually annotated to make sure it is interpretable at a later time. These constraints make data acquisition enormously time-consuming. And so there arises huge potential for accurate automated activity classification.

1.2 Activity classifiers

What is needed is an activity classifier that can distinguish between a series of activities that are very specific to for example rehabilitation exercises of a patient, or the job of a worker being analyzed. Moreover, after some initial learning process, the classifier is able to do this independently and automatically.

The sensor solution that provides the data for the classifier needs to be as 'minimal' as possible. That is, it needs to be lightweight so as to not affect the activities it tries to measure. It should consist of as few sensors as possible, positioned on the body in the least cumbersome locations, while at the same time it still provides enough data to capture the essence of any activities of interest.

1.2.1 Prior research

Over the past decade Roessingh Research and Development [26] (henceforth RRD) has been working on both ambulatory sensing solutions and activity classification. The work on ambulatory sensing resulted in the Fusion project (2009-2013) [2], where among other things the positioning and calibration of motion sensors on the human body was researched. The project culminated in the development of tools to accurately and objectively monitor motor function in physical and rehabilitation therapy, and sports. These tools were used to monitor the influence of fatigue on the coordination of subjects running a marathon, and to provide rowing coaches with comprehensive, accurate information about the movement, timing and behavior of competitive rowers.

The first work on activity classification was done in 2003 as part of the master's project of Ruben Wassink [38] and resulted in a software model called FusionAAC (Fusion Adaptive Activity Classification). FusionAAC applies a technique called Hidden Markov Modeling and has been further developed and tested in several pilot projects. For example, distinguishing between different lifting activities in a simulated workplace environment (see also [39] and [31]), but also monitoring arm and hand activities (eating, drinking, reaching, working on a laptop) of patients suffering from Duchenne muscular dystrophy [44] as part of the Fusion project. Primarily kinematic data recorded by specific motion sensor configurations serves as input to FusionAAC, but other input data types are possible.

1.2.2 Goals

After the initial groundwork done by RRD, some fundamental questions now have to be addressed. Maybe the most important part of designing an activity classifier is the training process. Without the right training, it will never be capable of high accuracy recognition. It is vital to assess whether a training set not only contains enough information, but also the right kind of data to effectively train the model. One of the main challenges addressed in this research will therefore be generalizing and optimizing the training effort. Some other aspects we will go into are:

- Data preprocessing: making sure that the amount of information in the data is maximized, while the amount of noise is minimized.
- The ability of the classifier to deal with classification (in)securities, e.g. dealing with classification doubt between multiple activities, most notably dealing with false positives.
- Minimizing the computational load.

1.3 Classifying arm and hand function in Duchenne patients

Duchenne muscular dystrophy (DMD) is a genetic disorder caused by a mutation in the dystrophin gene. This gene codes for the protein dystrophin, which strengthens muscle cells by attaching parts of the internal cytoskeleton to the cell membrane. Without it, the cell membrane becomes permeable, eventually causing cell burst due to excessive internal pressure. This results in a cycle wherein muscle fibers deteriorate and regenerate until the repair capacity of the tissue is no longer sufficient. The degeneration of the fibers becomes irreversible and muscle cells are replaced by fat cells and connective tissue.

The dystrophin gene is located on the X chromosome. It is a recessive gene. This means that both sexes can carry the mutation, but mainly male children are affected by DMD. Worldwide, one in 3500 male children [12] suffers from the disease. Only one percent of people with DMD is female. It is the most common inherited muscle disease in children.

Symptoms usually first appear in male children around age 6, although they may already be visible in early infancy. Patients first start to lose muscle mass in their legs and pelvis. Some less severe loss of muscle mass may also occur in the arms, neck, and other areas of the body. Patients experience problems with motor skills, like running and jumping, and walking starts to become problematic. From age 10, braces may be required to aid in walking. Most patients are wheelchair dependent by age 12. The progressive deterioration of muscle tissue leads to increasingly greater degrees of paralysis. Around age 20 the respiratory and heart muscles also start being affected, which eventually proves fatal.

At present, there is no cure for DMD. Treatment is generally aimed at controlling the symptoms to maximize the quality of life. Maintaining upper extremity function is very important in daily life, so physical therapy is vital to help maintain strength and flexibility. Inactivity can worsen the disease. Orthopedic appliances, such as wheelchairs and leg braces, may improve mobility and independence. As the disease progresses, it becomes necessary to support respiratory function.

Over the last couple of decades, new and improved ways of treatment have led to DMD patients living much longer. The average life expectancy for people with DMD currently is 27 years, but because of the individual variation in the progression of the disease, this differs from patient to patient. Nowadays half of the patients live past the age of 30. Some even live into their 40s and 50s. However, quality of life has not increased accordingly.

1.3.1 Adapt project

Loss of upper limb function has a major impact on the quality of life of patients. Many patients (not only those suffering from DMD, but for example also multiple sclerosis patients, or people recovering from a stroke or spinal cord injury) depend on assistive technology (AT) to support them during activities of daily life (ADL) like eating and drinking, reading or working on a laptop. Better assistance for patients means an increased level of independence and social participation, which greatly improves the quality of life. An example of an assistive device could be a wheelchair mounted (dynamic) arm support, see for example [36].

Unfortunately, current AT is not yet sufficiently matched to the capabilities and preferences of patients. This is why many patients stop using the technology [5]. Reasons for this are that users have to change numerous settings to switch between different activities, assistive devices do not know whether a user intends to be active, and the performance of devices only tends to be partially helpful.

With this in mind, the goal of the Adapt project [23] (part of the Symbionics Program [22]) is to research and develop new models and methods for future generations of AT. The project focuses on understanding the mutual interaction between patients and assistive devices. The goal is to provide an optimal physical support for the user and adjust the assistance according to the dynamics of the requirements set by task, user or environment. This personalized approach gives patients a central role in defining requirements and priorities in the development of assistive devices. Devices should continuously (co-)adapt to the user: according to a therapy plan (e.g. increased support when a patient gets tired), to compensate for user changes (e.g. in the case of disease progression), changing environments and changing tasks. Furthermore, the goal is to design assistive devices that completely fit underneath regular clothing, which is important for social acceptance.

To reach these goals, individual patient profiles need to be identified on multiple levels. Effective physical support solutions can be found through neuromusculoskeletal models. Co-adaptation of user and device, to cope with time-varying task requirements, can be facilitated through wearable sensors. An unobtrusive and

robust sensing solution needs to be developed to monitor the interaction between user and assistive device during activities of daily life. Ultimately, this should lead to an assistive device that is able to recognize and predict in real time what specific activity a user is performing. The device can use this information to optimally support the user.

1.3.2 Role of RRD

The real-time co-adaptive device as described in the previous section is still something for the (hopefully nearby) future. The first step towards such a device is highly accurate activity recognition. RRD plays an important role in this part of the Adapt project. Not only can the existing knowledge on activity classification be applied, there is already some prior experience with the classification of activities of DMD patients (see again [44], part of the Fusion project).

With accurate activity classification it becomes possible to get an idea of the typical daily activities of DMD patients. This knowledge can be used to identify where DMD patients could benefit most from AT. Ultimately, as the activity classifier becomes more advanced, it could in theory also be used for real-time recognition and prediction. This last part is beyond the scope of this research.

1.4 Overview of this report

Before we can start describing the modeling process for the classification of human activities, first it is necessary to explain the basics of hidden Markov modeling. Chapter 2 consist of an introduction to the theory of hidden Markov models. Chapter 3 describes the acquisition and processing of the data that will serve as input to the activity classifier. It also describes the process of modeling human activities and the software implementation of the theory from chapter 2, resulting in a working activity classifier.

The second part of this report covers the research methods. Chapter 4 discusses preprocessing of the data. Chapter 5 goes into the various aspects of the training process and how to improve upon them. Chapter 6 provides some strategies to improve on the results by trying to remove classification errors through postprocessing.

The final part of the report summarizes the results and conclusions of this research. The results of testing the improvements made to FusionAAC are shown in chapter 7. Chapter 8 presents the conclusion of this project and gives some suggestions for future research.

Chapter 2

Hidden Markov models

2.1 Introduction

Hidden Markov Models (henceforth HMMs) were first introduced and studied in the late 1960s and early 1970s by Leonard E. Baum and his coworkers. The technique is especially suitable for the modeling of processes that have a sequential quality, e.g. speech. Such processes typically produce time series with homogeneous zones, where the observed data within these zones are governed by similar distributions. In the case of speech, these zones can be phonemes that are characterized by a certain frequency. Starting from the mid-1970s, speech recognition was one of the first fields where HMMs were successfully applied [15] [24].

After the initial success realized in speech recognition, more research fields followed. Nowadays HMMs are widely known for their applications in describing or classifying a wide range of processes. Some examples closely related to speech recognition are handwriting [7] and sign language recognition [32]. The technique has also proven useful in modeling biological processes such DNA sequence analysis [18], automated musical accompaniment of human performers [20] and even computer virus detection [41].

To familiarize the reader with the modeling problem, this chapter will start with a toy example. In section 2.3 we will present a general hidden Markov model and discuss the various parameters of this model. Finally, section 2.4 describes how to estimate these model parameters.

2.2 Toy example

A hidden Markov model is a doubly stochastic process. It consists of an underlying stochastic process — more specifically a first-order Markov chain — that is not observable (hidden), and another set of stochastic processes (observable signals) through which the Markov chain can be indirectly observed. We will first illustrate this abstract concept with some simple coin toss experiments largely inspired by [25], and then extend the ideas to a slightly more complicated ball and urn model.

2.2.1 A hidden coin toss experiment

To understand the concept of HMMs, consider the following situation. Suppose you are in a room that is separated in two by a curtain. On the other side of the curtain, hidden from view, a coin tossing experiment is being performed. Because of the curtain, the exact nature of the experiment is unknown, but at regular time intervals the results are called out to you. A possible sequence of "observations" on your side of the curtain would look like

$$\begin{aligned}\mathcal{O} &= O_1 O_2 O_3 \cdots O_T \\ &= \mathcal{T}\mathcal{H}\mathcal{H}\cdots\mathcal{T}\end{aligned}$$

where $\mathcal{H}$ stands for "heads" and $\mathcal{T}$ stands for "tails".

The question now is, how to build a model that best explains the observed sequence of heads and tails? The first problem is deciding which part of the real-world process you want to describe through the hidden Markov chain states, and how many of these states are needed. Secondly, the observable stochastic process needs to be defined.

For the purpose of this discussion we will make things a little more interesting and skip the easiest explanation, namely that the person on the other side of the curtain is simply tossing a single coin and calling out the result to you. A simple model for explaining the sequence of coin tosses is given in figure 2.1. We will call it the "2 fair coins" model.

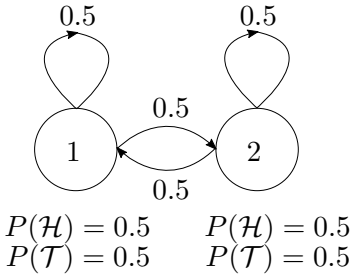


Figure 2.1: "2 fair coin" model

It consists of two states. Each state corresponds to a different coin being tossed. The observable signal in each state is the result of this coin toss, which makes the observable process stochastic. The state transitions between the different coins could in turn be described by another coin toss, independent of the first two. Each face of this third coin is associated with one of the coins corresponding to the states. Note that because of the fairness of all coins, the complete process is statistically no different from a single, completely observable, fair coin toss experiment.

A second possible model (figure 2.2) is a slight generalization of the first, the only difference with the previous example being the bias of the two coins corresponding to the states. It is called the "2 biased coins" model. Note that, because of the specific choice of the bias of these two coins, the long term statistical behavior is

still no different from a single, completely observable, fair coin toss experiment. The introduction of two different, biased coins does however present us with new possibilities to explain more complex statistical behavior of the observation sequence.

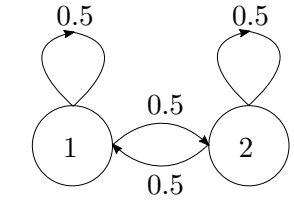
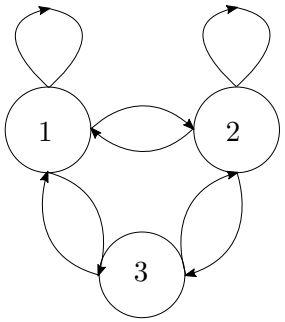


Figure 2.2: "2 biased coins" model

The third and final coin toss example we will present here is called the "3 biased coins" model (figure 2.3). It consists of three states, each corresponding to a different biased coin. The first coin is slightly biased towards "heads", the second is strongly biased towards "tails", and the third is slightly biased towards "tails". In this model, the behavior of the observation sequence depends strongly on the state transition probabilities. Consider for example the cases where the probability of staying in the second state is very large ($P_{22} > 0.95$) or very small ($P_{22} < 0.05$). Very different long term statistics will arise from these two extremes.



State	1	2	3
$P(\mathcal{H})$	0.6	0.25	0.45
$P(\mathcal{T})$	0.4	0.75	0.55

Figure 2.3: "3 biased coins" model

Summarizing, the above examples show that one of the main challenges of building a HMM is deciding on the number of states in the model. Typically, deciding on the value of this parameter will depend largely on the modeling problem at hand, careful study of any available observation data, and maybe some trial and error. On top of this, the size of the observation sequence also plays a major role. The larger

the model is, the more unknown parameters it has. If the observation sequence is too small compared to the number of unknown parameters, it becomes impossible to reliably estimate all these parameters, and any resulting HMMs of different sizes may not be statistically different.

2.2.2 A ball and urn model

To fix ideas, we will extend the ideas presented in the coin toss experiments to a slightly more complicated ball and urn model. Consider the following situation. A room contains a number of urns. Each urn contains a large amount of different colored balls. According to some unknown random process an urn is chosen at regular intervals. When an urn is selected, a ball from this urn is selected. The color of this ball is recorded as the observation and the ball is then replaced in its original urn. The process is repeated until some finite observation sequence of colors is generated.

We would like to model the observation sequence as the observable output of a HMM. A natural choice is to let the number of urns correspond to the states of the hidden Markov chain. The random process through which the urns are chosen is modeled as the transition probability matrix of the Markov chain. The distribution of the observed colors is modeled through a set of discrete observation distributions, one for each urn. The remaining problem then consists of estimating the parameters of both the transition probability matrix and the observation probability distributions.

2.3 A general hidden Markov model

This section will describe the basic elements of a general HMM. All examples treated in the previous section contained a discrete observation process. First we formally define the model notation for such discrete observation HMMs. We will go into the structure of the underlying Markov chain and its implications for the model. Lastly, we will say something about the distribution of the observable signals and extend the model notation to the continuous case.

2.3.1 Definitions and notation

A basic, discrete HMM is characterized by five elements, namely

1. A finite number of unobservable states N. Each state can broadcast an observable signal that possesses some measurable, distinctive properties. The individual states are elements of the state space $\mathcal{S} = \{q_1, q_2, \ldots, q_N\}$, and the state at time t is denoted as X_t.

2. The state transition probability matrix $\mathbf{A} = \{a_{ij}\}$, where

$$a_{ij} := P\left(X_{t+1} = q_j | X_t = q_i\right), \quad 1 \leq i, j \leq N. \tag{2.1}$$

That is, a transition to a new state only depends on the current state (the Markov property). Furthermore, the transition probabilities are independent of the time t (the process is stationary). If it is possible to go from one state to another in a single step, we have that $a_{ij} > 0$. Otherwise $a_{ij} = 0$. Note that the rows of the matrix should sum up to one:

$$\sum_{j=1}^{N} a_{ij} = 1.$$

3. The number of distinct observable signals M. For the coin toss experiments these signals were simply "heads" and "tails". For the ball and urn model, M is the total number of distinct colors. The individual signals are denoted as $\mathcal{V} = \{v_1, v_2, \ldots, v_M\}$, and the signal at time t is denoted as O_t.

4. The distribution of the observable signals in state j, $\mathbf{B} = \{b_j(k)\}$, where

$$b_j(k) := P\left(O_t = v_k | X_t = q_j\right), \ 1 \leq j \leq N, \ 1 \leq k \leq M. \tag{2.2}$$

   Again we assume that the process is stationary, i.e. $\mathbf{B}$ is independent of the time t.

5. The initial state distribution $\boldsymbol{\pi} = \{\pi_i\}$, where

$$\pi_i := P\left(X_1 = q_i\right), \quad 1 \leq i \leq N. \tag{2.3}$$

At each clock time $1 < t \leq T$, a new state is entered based upon the transition probability matrix $\mathbf{A}$. This state then generates some observable signal. From the above model, a sequence of these observations $\mathcal{O} = O_1 O_2 \cdots O_T$ is generated in the following way:

1. Generate an initial state $X_1 = q_i$, using the initial state distribution $\boldsymbol{\pi}$.
2. Set $t = 1$.
3. Generate $O_t = v_k$, using the distribution of observable signals in state q_i, $b_i(k)$.
4. Transit to a new state $X_{t+1} = q_j$, using the transition probability matrix $\mathbf{A}$.
5. Set $t = t + 1$. If $t \leq T$, return to step 3, otherwise terminate the procedure.

Summarizing, we note that a complete definition of a HMM requires specification of the three probability distributions $\mathbf{A}$, $\mathbf{B}$ and $\boldsymbol{\pi}$ (and implicitly the dimensions M and N). For the sake of convenience, we will henceforth use the compact notation

$$\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$$

to represent a HMM.

2.3.2 The underlying Markov chain

In section 2.2 we discussed some examples of simple HMMs. In all of the underlying Markov chains of these examples every state could be reached in a single step from any other state. A model of this type has the property that all elements of the transition probability matrix $\mathbf{A}$ are strictly positive. These kinds of Markov chains are also called ergodic.

Other architectures are also possible. Here we would like to discuss one particular alternative in more detail, namely the left-to-right model. It is called a left-to-right model because as time increases, the state index does not decrease, i.e. the states proceed from left to right. Architectures like these impose a temporal order to a HMM, since states with a lower index number are linked to observations occurring prior to those that are linked to states with a higher index. For this reason, in speech recognition often left-to-right Markov chains are used [24] [10, p. 33] [6, p. 614].

The transition probability matrix of a left-to-right model has the property that

$$a_{ij} = 0, \quad j < i,$$

i.e. $\mathbf{A}$ has an upper triangular form. Furthermore, since the sequence must begin in state 1 the initial state distribution is of the form

$$\pi_i = \begin{cases} 1, & i = 1 \\ 0, & i \neq 1 \end{cases}$$

Additionally, you might want to place extra constraints on the state transitions to make sure that large jumps in the state index do not occur. Such a constraint might be of the form

$$a_{ij} = 0, \quad j > i + \Delta.$$

So if for example Δ has value 2, jumps of more than two states are not possible. See figure 2.4 for a schematic representation of this example.

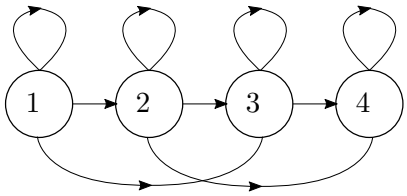


Figure 2.4: Four state left-to-right Markov chain with $\Delta = 2$

Left-to-right models have another useful quality. Where an ergodic chain will produce an infinitely long output sequence of states, a left-to-right model will only generate a sequence of finite, random length. This is true since at each clock time, the probability that the state index increases is positive. This means that the time until arrival at a state with a greater index has a geometric distribution with finite mean. The benefit of having a random length output sequence is that it will provide

some robustness to local stretching and compression of the time axis. In the case of speech recognition, this stretching and compression is associated with variations in the speed of the speech. A left-to-right model will to some extent be able to handle these distortions by inserting extra transitions from a state to itself.

2.3.3 Distribution of the observable signals

Earlier in this section, when we described a general HMM, we limited ourselves to a discrete distribution of the observable signals. However, in many applications the observation signals are continuous. This is also the case for our activity monitor, which makes use of continuous motion sensor data. Luckily, the extension to a continuous distribution is readily made.

In a continuous observation model, the $b_j(k)$ are replaced by a continuous density $b_j(\mathbf{x})$ ($1 \leq j \leq N$, one for every state in the Markov chain), where $\mathbf{x}$ is some observation vector that is being modeled. The most general probability density function, for which a re-estimation procedure has been formulated, is a finite mixture of the form

$$b_j(\mathbf{x}) := \sum_{m=1}^{M_j} c_{jm} \mathfrak{N}[\mathbf{x}|\boldsymbol{\mu}_{jm}, \mathbf{U}_{jm}]. \tag{2.4}$$

Here c_{jm} is the mixture coefficient for the mth mixture in state j. $\mathfrak{N}$ is any log-concave or elliptically symmetric density (e.g. a Gaussian density), with mean vector $\boldsymbol{\mu}_{jm}$ and covariance matrix $\mathbf{U}_{jm}$, for the mth mixture in state j. The mixture weights satisfy

$$\begin{aligned} \sum_{m=1}^{M} c_{jm} &= 1, \quad 1 \leq j \leq N \\ c_{jm} &\geq 0, \quad 1 \leq j \leq N, \quad 1 \leq m \leq M \end{aligned}$$

so that the probability density function is properly normalized, i.e.

$$\int_{-\infty}^{\infty} b_j(\mathbf{x}) d\mathbf{x} = 1, \quad 1 \leq j \leq N.$$

See figure 2.5 for an example of a (Gaussian) mixture distribution. A probability density function such as (2.4) can be used to approximate, arbitrarily close, any finite, continuous density function. This makes it suitable to be applied to a wide range of problems. The simplest example of a distribution like this, is a single component Gaussian distribution.

It turns out that it is possible to show that a HMM λ consisting of N states, where each state is described by a Gaussian mixture distribution, is (probabilistically) equivalent to another HMM λ' where each state is described by only a single Gaussian. Before we can prove this, we need some additional theory that will be

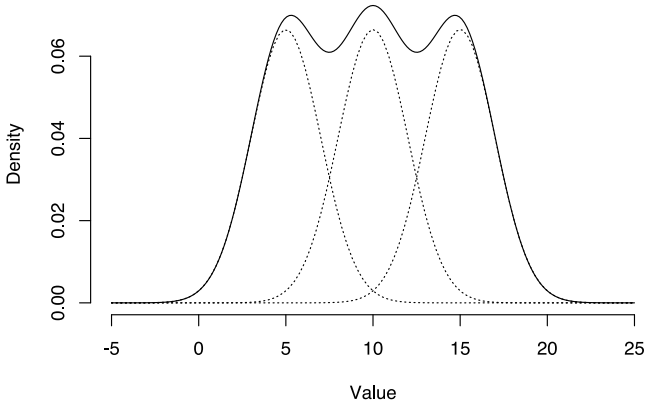


Figure 2.5: Density of a mixture of three normal distributions ($\mu = \{5, 10, 15\}$, $\sigma = 2$) with equal weights. Each component is shown as a weighted density (each integrating to 1/3) [30]

described in the remainder of this chapter. For the proof, see Appendix A. The main reason that we mention mixture distributions, is that most of the literature on HMMs seems to omit this equivalence result. We will not use mixture distributions in this research.

2.4 Estimating the model parameters

Up to this point, we have described the most important elements that make up a general HMM. Before we can use the HMM in real world applications, three key problems have to be solved. These problems are the following.

Problem 1: Given an observation vector $\mathcal{O} = O_1 O_2 \cdots O_T$ and a model $\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$, how to efficiently compute the probability $P(\mathcal{O}|\lambda)$ that this observation vector was generated by the model?

This is the evaluation problem. It can also be viewed as the problem of scoring how well a given model matches a given observation. The solution allows you to choose the best match among competing models.

Problem 2: Given an observation vector $\mathcal{O} = O_1 O_2 \cdots O_T$ and a model $\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$, how to choose a corresponding underlying hidden state sequence $\mathcal{X} = X_1 X_2 \cdots X_T$ that best explains the observations?

This problem is about uncovering the hidden part of the model. There is no "correct" solution to this problem. The best we can do is use some optimality criterion to find the hidden state sequence that is most likely to have generated a given observation.

Problem 3: How to adjust the model parameters $\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$ to maximize $P(\mathcal{O}|\lambda)$ for given data $\mathcal{O}$?

This is the learning problem. It can be viewed as training a model to best fit the observed data. The observation sequence used to adjust ("train") the model parameters is called a training sequence.

Figure 2.6 gives a schematic overview of why it is necessary to solve these problems and how the solutions to these three problems are applied to the classification process.

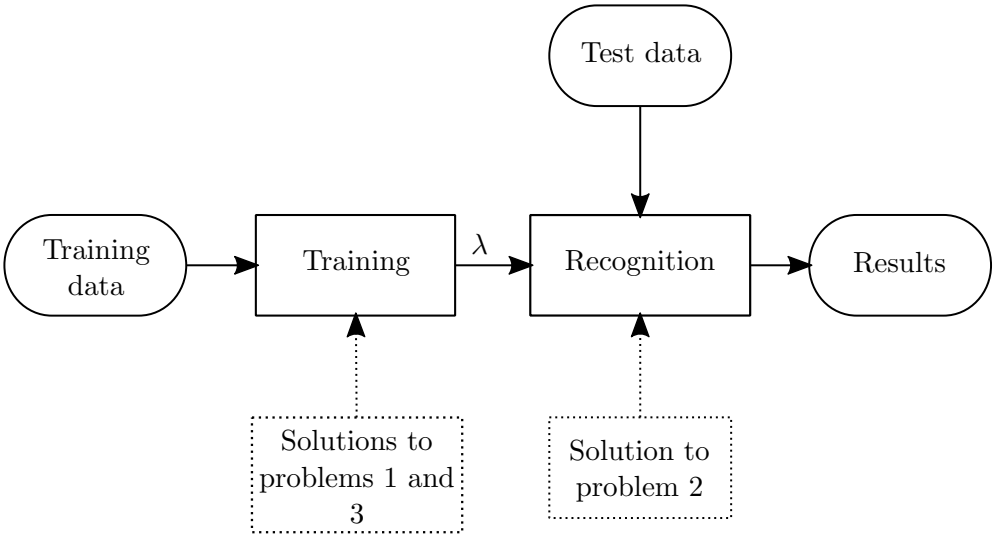


Figure 2.6: The training process alternates between applying the solutions to problem 1 and 3; to score how well a model fits the training data, and to update the parameters until some level of optimality is reached. The fully trained model then uses the solution to problem 2 to classify independent test data.

2.4.1 Solution to problem 1

Let $\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$ be a given model, let $\mathcal{O} = O_1 O_2 \cdots O_T$ be an observation sequence, and let $\mathcal{X} = X_1 X_2 \cdots X_T$ be a state sequence. By the definition of the distributions $\mathbf{A}$, $\mathbf{B}$ and $\boldsymbol{\pi}$, we have

$$P(\mathcal{X}|\lambda) = \pi_{X_1} a_{X_1,X_2} a_{X_2,X_3} \cdots a_{X_{T-1},X_T}$$
$$P(\mathcal{O}|\mathcal{X},\lambda) = b_{X_1}(O_1) b_{X_2}(O_2) \cdots b_{X_T}(O_T)$$

We are interested in finding $P(\mathcal{O}|\lambda)$. A direct approach to compute this quantity would be to sum over all possible state sequences $\mathcal{X}$:

$$\begin{aligned} P(\mathcal{O}|\lambda) &= \sum_{\mathcal{X}} P(\mathcal{O}, \mathcal{X}|\lambda) \\ &= \sum_{\mathcal{X}} P(\mathcal{O}|\mathcal{X}, \lambda) P(\mathcal{X}|\lambda) \\ &= \sum_{\mathcal{X}} \pi_{X_1} b_{X_1}(O_1) a_{X_1,X_2} b_{X_2}(O_2) \cdots a_{X_{T-1},X_T} b_{X_T}(O_T). \end{aligned}$$

Here the first step makes use of conditional probability. However, since there are N^T possible state sequences, this approach would require $(2T-1)N^T$ multiplications and $N^T - 1$ additions and is therefore computationally infeasible. Clearly a more efficient procedure is required. Such a procedure exists and is called the Forward-Backward algorithm [24].

The Forward-Backward algorithm

Let $\mathcal{O}^{1:t} = (O_1, \ldots, O_t)$ be the observation sequence until time t. For $1 \leq t \leq T$ and $1 \leq j \leq N$, define the forward variable in the following way:

$$\alpha_t(j) := P\left(\mathcal{O}^{1:t}, X_t = q_j|\lambda\right). \tag{2.5}$$

Now, omitting the λ's for the sake of brevity, and using conditional probability:

$$\begin{aligned} \alpha_t(j) &= P\left(\mathcal{O}^{1:t-1}, O_t, X_t = q_j\right) \\ &= \sum_{i=1}^{N} P\left(\mathcal{O}^{1:t-1}, X_{t-1} = q_i, X_t = q_j, O_t\right) \\ &= \sum_{i=1}^{N} P\left(X_t = q_j, O_t|\mathcal{O}^{1:t-1}, X_{t-1} = q_i\right) \cdot P\left(\mathcal{O}^{1:t-1}, X_{t-1} = q_i\right) \\ &= \sum_{i=1}^{N} P\left(X_t = q_j, O_t|\mathcal{O}^{1:t-1}, X_{t-1} = q_i\right) \alpha_{t-1}(i) \\ &= \sum_{i=1}^{N} P\left(X_t = q_j, O_t|X_{t-1} = q_i\right) \alpha_{t-1}(i) \\ &= \sum_{i=1}^{N} a_{ij} b_j(O_t) \alpha_{t-1}(i) \\ &= \left[\sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij}\right] b_j(O_t) \end{aligned}$$

See figure 2.7 for an illustration of the calculation of the forward variables.

So, for all t, j the $\alpha_t(j)$ can be computed recursively:

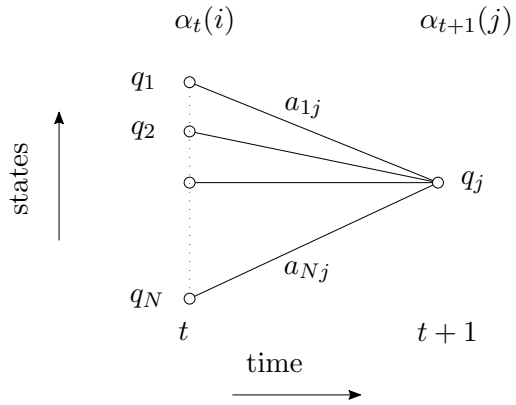


Figure 2.7: To calculate the value of the forward variable at time $t+1$, you only need the values of the forward variables for all states at time t.

1. Initial step:

$$\alpha_1(j) = \pi_j b_j(O_1), \quad 1 \leq j \leq N$$

2. For $1 \leq t \leq T-1$, and $1 \leq j \leq N$

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1})$$

3. Termination. From the definition of $\alpha_j(t)$, we find

$$P\left(\mathcal{O}|\lambda\right) = \sum_{j=1}^{N} \alpha_T(j)$$

To calculate $P\left(\mathcal{O}|\lambda\right)$ we need $N(N+1)(T-1)+N \approx N^2T$ multiplications and $N(N-1)(T-1)+N$ additions. In other words, using the forward algorithm has an enormous computational advantage over the direct approach as described above.

Let $\mathcal{O}^{t+1:T} = (O_{t+1}, \ldots, O_T)$ be the observation sequence from time $t+1$ until time T. Then in similar fashion, we can also define a backward variable:

$$\beta_t(i) := P\left(\mathcal{O}^{t+1:T}|X_t = q_i, \lambda\right). \tag{2.6}$$

Again omitting the λ's and using conditional probability:

$$\begin{aligned}
\beta_t(i) &= \sum_{j=1}^{N} P\left(\mathcal{O}^{t+1:T} | X_t = q_i, X_{t+1} = q_j\right) P\left(X_{t+1} = q_j | X_t = q_i\right) \\
&= \sum_{j=1}^{N} P\left(\mathcal{O}^{t+1:T} | X_{t+1} = q_j\right) a_{ij} \\
&= \sum_{j=1}^{N} P\left(O_{t+1} | X_{t+1} = q_j\right) \cdot P\left(\mathcal{O}^{t+2:T} | O_{t+1}, X_{t+1} = q_j\right) a_{ij} \\
&= \sum_{j=1}^{N} b_j(O_{t+1}) P\left(\mathcal{O}^{t+2:T} | X_{t+1} = q_j\right) a_{ij} \\
&= \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)
\end{aligned}$$

See figure 2.8 for an illustration of the calculation of the backward variables.

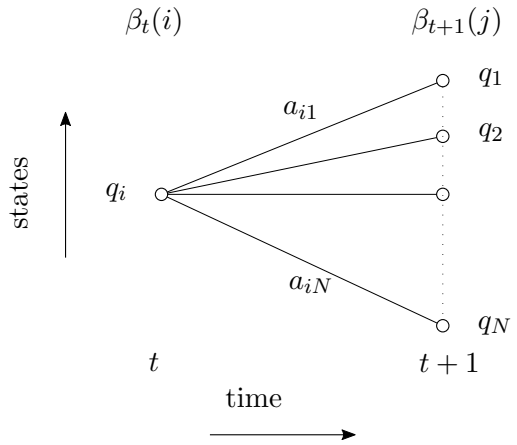


Figure 2.8: To calculate the value of the backward variable at time t, you only need the values of the forward variables for all states at time $t+1$.

So the $\beta_t(i)$ can then also be computed recursively:

1. Initial step:
$$\beta_T(i) = 1, \quad 1 \leq i \leq N$$

2. For $t = T-1, T-2, \ldots, 1$, and $1 \leq j \leq N$
$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)$$

3. Termination.

$$\begin{aligned} P(\mathcal{O}|\lambda) &= \sum_{i=1}^{N} P(\mathcal{O}|X_1 = q_i, \lambda)\, \pi_i \\ &= \sum_{i=1}^{N} P(O_1|X_1 = q_i, \lambda)\, P\left(\mathcal{O}^{2:T}|O_1, X_1 = q_i, \lambda\right) \pi_i \\ &= \sum_{i=1}^{N} b_i(O_1) P\left(\mathcal{O}^{2:T}|X_1 = q_i, \lambda\right) \pi_i \\ &= \sum_{i=1}^{N} b_i(O_1)\beta_1(i)\pi_i \end{aligned}$$

The computational cost of using the backward algorithm is comparable to that of the forward algorithm.

It is possible to combine both these approaches:

$$\begin{aligned} P(\mathcal{O}, X_k = q_j|\lambda) &= P\left(\mathcal{O}^{1:k}, X_k = q_j|\lambda\right) P\left(\mathcal{O}^{k+1:T}|\mathcal{O}^{1:k}, X_k = q_j, \lambda\right) \\ &= P\left(\mathcal{O}^{1:k}, X_k = q_j|\lambda\right) P\left(\mathcal{O}^{k+1:T}|X_k = q_j, \lambda\right) \\ &= \alpha_k(j)\beta_k(j) \end{aligned}$$

and hence

$$P(\mathcal{O}|\lambda) = \sum_{j=1}^{N} \alpha_k(j)\beta_k(j). \tag{2.7}$$

2.4.2 Solution to problem 2

For the solution to problem 2, [28, p. 261–263] proved very helpful. Let $\mathcal{O} = O_1 O_2 \cdots O_T$ be an observation sequence. Given this data we want to find the state sequence that best explains the observation. What exactly this best explanation is, depends on what we are trying to accomplish.

Individually most likely states

If the objective is to maximize the number of *individually* most likely states, we need to compute

$$\arg\max_j P(X_t = q_j|\mathcal{O}, \lambda), \quad 1 \leq j \leq N,$$

for all clock times $1 \leq t \leq T$. To do so, we use conditional probability and the forward and backward variables as defined in the previous subsection. For $1 \leq t \leq T$

and $1 \leq j \leq N$, define

$$\begin{aligned} \gamma_t(j) :&= P\left(X_t = q_j | \mathcal{O}, \lambda\right) \\ &= \frac{P\left(X_t = q_j, \mathcal{O}|\lambda\right)}{P\left(\mathcal{O}|\lambda\right)} \\ &= \frac{\alpha_t(j)\beta_t(j)}{\sum_i \alpha_t(i)\beta_t(i)}. \end{aligned} \tag{2.8}$$

So, the optimal predictors at time t of the index of the state i_t^* and the most likely state X_t^* itself are then given by

$$\begin{aligned} i_t^* &= \arg\max_j \gamma_t(j) \\ &= \arg\max_j \frac{\alpha_t(j)\beta_t(j)}{\sum_i \alpha_t(i)\beta_t(i)} \end{aligned} \tag{2.9}$$

$$X_t^* = q_{i_t^*} \tag{2.10}$$

A downside of considering only the individually most likely states, is that the obtained optimal state sequence might allow for impossible state transitions if $a_{ij} = 0$ for some individually optimal i, j. It simply provides us with the most likely state at each clock time without regard to network structure, neighboring states or the length of the observation sequence.

Viterbi algorithm

Alternatively, we can regard the sequence of states as a single entity. The objective is now, given some observation sequence, to choose the sequence of states (path) whose conditional probability as a whole is maximal. The algorithm that solves this problem is called the Viterbi algorithm. Let $\mathcal{X}^{1:t} = (X_1, \ldots, X_t)$ be the vector representation of the first t states, and let $\mathcal{O}^{1:t} = (O_1, \ldots, O_t)$ be the observation sequence until time t. The problem of interest is to find the sequence of states $X_1, \ldots, X_t$ that maximizes $P\left(\mathcal{X}^{1:t}|\mathcal{O}^{1:t}, \lambda\right)$. Since

$$P\left(\mathcal{X}^{1:t}|\mathcal{O}^{1:t}, \lambda\right) = \frac{P\left(\mathcal{X}^{1:t}, \mathcal{O}^{1:t}|\lambda\right)}{P\left(\mathcal{O}^{1:t}|\lambda\right)}$$

and $P\left(\mathcal{O}^{1:t}|\lambda\right)$ only depends on the model parameters (see previous section), this maximization problem is equivalent to finding the sequence of states $X_1, \ldots, X_t$ that maximizes $P\left(\mathcal{X}^{1:t}, \mathcal{O}^{1:t}|\lambda\right)$. For the sake of brevity, we will omit the λ term when denoting these probabilities.

Given an observation sequence, then for each intermediate and terminating state of the hidden Markov chain, there is a most probable path to that state. We will call these paths partial best paths. Each of these partial best paths has an associated probability which we will call δ. We define $\delta_t(j)$ to be the probability associated

with the partial best path ending at state q_j at time t. Formally:

$$\delta_t(j) := \max_{\mathcal{X}^{1:t-1}} \left[P\left(\mathcal{X}^{1:t-1}, X_t = q_j, \mathcal{O}^{1:t}\right) \right] \tag{2.11}$$

For $t = 1$, the most probable path to a state does not sensibly exist; there are no preceding states. However, we can use the probability of being in that state given $t = 1$ and the observed signal O_1:

$$\begin{aligned} \delta_1(j) &= P\left(X_1 = j, O_1\right) \\ &= \pi_j b_j(O_1) \end{aligned}$$

Recall that the Markov property says that the probability of transitioning to the next state, given a sequence of the previous states, depends only on the current state. The probability of the most probable path to state X_t can now be recursively calculated in the following way:

$$\begin{aligned} \delta_t(j) &= \max_{\mathcal{X}^{1:t-1}} \left[P\left(\mathcal{X}^{1:t-1}, X_t = q_j, \mathcal{O}^{1:t}\right) \right] \\ &= \max_i \left[\max_{\mathcal{X}^{1:t-2}} P\left(\mathcal{X}^{1:t-2}, X_{t-1} = q_i, X_t = q_j, \mathcal{O}^{1:t-1}, O_t\right) \right] \\ &= \max_i \left[\max_{\mathcal{X}^{1:t-2}} P\left(\mathcal{X}^{1:t-2}, X_{t-1} = q_i, \mathcal{O}^{1:t-1}\right) \right. \\ &\qquad \left. \cdot P\left(X_t = q_j, O_t | \mathcal{X}^{1:t-2}, X_{t-1} = q_i, \mathcal{O}^{1:t-1}\right) \right] \\ &= \max_i \left[\max_{\mathcal{X}^{1:t-2}} P\left(\mathcal{X}^{1:t-2}, X_{t-1} = q_i, \mathcal{O}^{1:t-1}\right) \cdot P\left(X_t = q_j, O_t | X_{t-1} = q_i\right) \right] \\ &= \max_i \left[P\left(X_t = q_j, O_t | X_{t-1} = q_i\right) \cdot \max_{\mathcal{X}^{1:t-2}} P\left(\mathcal{X}^{1:t-2}, X_{t-1} = q_i, \mathcal{O}^{1:t-1}\right) \right] \\ &= \max_i \left[a_{ij} \delta_{t-1}(i) \right] b_j(O_t) \end{aligned}$$

So, to calculate $\delta_t(\cdot)$, we only need $\delta_{t-1}(i)$ for all states $1 \leq i \leq N$. Having calculated these probabilities, it is possible to record which preceding state was the one to generate $\delta_t(i)$, i.e. in which state the process must have been at time $t-1$ if it is to arrive optimally at state q_i at time t. To keep track of the states that maximize the δ's, define

$$\Psi_t(j) = \arg\max_i \left[\delta_{t-1}(i) a_{ij} \right]. \tag{2.12}$$

The Ψ's answer the question "if I am here, by what path is it most likely I arrived?". Summarizing, the complete Viterbi algorithm now looks like this:

1. Initialization:

$$\begin{aligned} \delta_1(i) &= \pi_i b_i(O_1), \quad 1 \leq i \leq N \\ \Psi_1(i) &= 0. \end{aligned}$$

2. Recursion:

$$\begin{aligned}\delta_t(j) &= \max_{1\leq i\leq N}\left[\delta_{t-1}(i)a_{ij}\right]b_j(O_t), && 2\leq t\leq T, \quad 1\leq j\leq N\\ \Psi_t(j) &= \arg\max_{1\leq i\leq N}\left[\delta_{t-1}(i)a_{ij}\right], && 2\leq t\leq T, \quad 1\leq j\leq N\end{aligned}$$

3. Termination:

$$\begin{aligned}P^* &= \max_{1\leq i\leq N}\left[\delta_T(i)\right]\\ i_T^* &= \arg\max_{1\leq i\leq N}\left[\delta_T(i)\right]\\ X_T^* &= q_{i_T^*}\end{aligned}$$

4. Path backtracking:

$$\begin{aligned}i_t^* &= \Psi_{t+1}(i_{t+1}^*), \quad t=T-1,T-2,\ldots,1\\ X_t^* &= q_{i_t^*}\end{aligned}$$

Here P^* is the total probability of the most likely path given some observation sequence $\mathcal{O}$, i_t^* is the index of the most likely state at time t, and X_t^* is the most likely state at time t of the most likely path.

2.4.3 Solution to problem 3

The third problem is to adjust the parameters $(\mathbf{A},\mathbf{B},\boldsymbol{\pi})$ of a HMM λ to best fit an observation sequence $\mathcal{O}$. Formally:

$$\lambda^* = \arg\max_{\lambda} P\left(\mathcal{O}|\lambda\right)$$

There is no known way to analytically solve for such a λ. In fact, given any finite observation sequence, there is no optimal way of estimating the model parameters [24]. However, it is possible to use an iterative procedure called the Baum-Welch re-estimation algorithm to find a local maximum for $P\left(\mathcal{O}|\lambda\right)$.

Given some model λ and an observation sequence $\mathcal{O}$, define $\xi_t(i,j)$ to be the probability of being in state q_i at time t, and state q_j at time $t+1$:

$$\xi_t(i,j) := P\left(X_t=q_i, X_{t+1}=q_j|\mathcal{O},\lambda\right) \tag{2.13}$$

Using conditional probability, $\xi_t(i,j)$ can be expressed in terms of the forward and

backward variables:

$$\begin{aligned}
\xi_t(i,j) &= \frac{P\left(X_t = q_i, X_{t+1} = q_j, \mathcal{O}|\lambda\right)}{P\left(\mathcal{O}|\lambda\right)} \\
&= \frac{P\left(X_t = q_i, X_{t+1} = q_j, \mathcal{O}^{1:t}, \mathcal{O}^{t+1:T}|\lambda\right)}{P\left(\mathcal{O}|\lambda\right)} \\
&= \frac{P\left(X_{t+1} = q_j, \mathcal{O}^{t+1:T}|X_t = q_i, \mathcal{O}^{1:t}, \lambda\right) \cdot P\left(X_t = q_i, \mathcal{O}^{1:t}|\lambda\right)}{P\left(\mathcal{O}|\lambda\right)} \\
&= \frac{P\left(X_{t+1} = q_j, O_{t+1}, \mathcal{O}^{t+2:T}|X_t = q_i, \mathcal{O}^{1:t}, \lambda\right) \cdot P\left(X_t = q_i, \mathcal{O}^{t}|\lambda\right)}{P\left(\mathcal{O}|\lambda\right)} \\
&= \frac{1}{P\left(\mathcal{O}|\lambda\right)} \big[P\left(\mathcal{O}^{t+2:T}|X_t = q_i, X_{t+1} = q_j, \mathcal{O}^{1:t}, O_{t+1}, \lambda\right) \\
&\qquad \cdot P\left(X_{t+1} = q_j, O_{t+1}|X_t = q_i, \mathcal{O}^{1:t}, \lambda\right) \cdot P\left(X_t = q_i, \mathcal{O}^{1:t}|\lambda\right)\big] \\
&= \frac{1}{P\left(\mathcal{O}|\lambda\right)} \big[P\left(\mathcal{O}^{t+2:T}|X_{t+1} = q_j, \lambda\right) \cdot P\left(X_{t+1} = q_j, O_{t+1}|X_t = q_i, \lambda\right) \\
&\qquad \cdot P\left(X_t = q_i, \mathcal{O}^{1:t}|\lambda\right)\big] \\
&= \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P\left(\mathcal{O}|\lambda\right)}
\end{aligned}$$

Here the forward variable $\alpha_t(i)$ accounts for the first t observations, ending at state q_i at time t. The term $a_{ij}b_j(O_{t+1})$ accounts for the transition to state q_j at time $t+1$ along with the observation of signal O_{t+1}. Lastly, the backward variable $\beta_{t+1}(j)$ accounts for the remainder of the observation sequence.

Recall that in equation (2.8) we defined

$$\gamma_t(i) = P\left(X_t = q_i|\mathcal{O}, \lambda\right).$$

$\gamma_t(i)$ can be related to $\xi_t(i,j)$ by summing over j:

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

Define a counting variable

$$I_t := \begin{cases} 1, & X_t = q_i \\ 0, & \text{otherwise.} \end{cases}$$

Then

$$\begin{aligned}\mathbb{E}\left[\#\text{ visits to state } q_i\right] &= \mathbb{E}\left[\#\text{ transitions from state } q_i\right] \\ &= \mathbb{E}\left[\sum_{t=1}^{T-1} I_t\right] \\ &= \sum_{t=1}^{T-1} \mathbb{E}\left[I_t\right] \\ &= \sum_{t=1}^{T-1} \gamma_t(i)\end{aligned}$$

Similarly,

$$\mathbb{E}\left[\#\text{ transitions from state } q_i \text{ to state } q_j\right] = \sum_{t=1}^{T-1} \xi_t(i,j)$$

We now have the building blocks to write down the re-estimation formulas. The formula for the initial distribution is simply the probability of being in state q_i at time $t = 1$:

$$\hat{\pi}_i = \gamma_1(i), \quad 1 \leq i \leq N. \tag{2.14}$$

The re-estimation formula for a_{ij} is the expected number of transitions from q_i to q_j divided by the expected total number of transitions out of q_i:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}, \quad 1 \leq i, j \leq N. \tag{2.15}$$

The re-estimation formula for a discrete signal distribution $b_j(k)$ is the expected number of visits to state q_j where k is the observed signal, divided by the expected total number of visits to state q_j:

$$\hat{b}_j(k) = \frac{\sum_{t=1}^{T} I_t^k \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)}, \quad 1 \leq j \leq N, 1 \leq j \leq M \tag{2.16}$$

where

$$I_t^k := \begin{cases} 1, & O_t = v_k \\ 0, & \text{otherwise.} \end{cases}$$

For a continuous, Gaussian signal distribution $b_j(\mathbf{x})$, i.e. each state output is a single component Gaussian, the means and variances of the HMM need to be re-estimated.

If a HMM consists of just one state, the re-estimation would be easy. The estimators of μ and Σ would just be the averages:

$$\hat{\mu} = \frac{1}{T}\sum_{t=1}^{T} O_t,$$
$$\hat{\Sigma} = \frac{1}{T}\sum_{t=1}^{T}(O_t - \mu)(O_t - \mu)^\top.$$

In practice, there usually are multiple states and because the underlying state sequence is unknown there is no direct assignment of observation vectors to individual states. The solution is to assign each observation to every state in proportion to the probability of the model being in a certain state at a certain time. Let $\gamma_t(j)$ again denote the probability of being in state j at time t, the re-estimation formulas then become the following weighted averages:

$$\hat{\mu}_j = \frac{\sum_{t=1}^{t} \gamma_t(j) O_t}{\sum_{t=1}^{t} \gamma_t(j)}, \tag{2.17}$$
$$\hat{\Sigma}_j = \frac{\sum_{t=1}^{t} \gamma_t(j)(O_t - \mu)(O_t - \mu)^\top}{\sum_{t=1}^{t} \gamma_t(j)}. \tag{2.18}$$

One can check that all of the above estimators are the Maximum Likelihood Estimators (MLEs), see for example [6]. The complete iterative procedure is now as follows:

1. Initialization of the model $\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$.
2. Re-estimation. Compute $\hat{\mathbf{A}}$, $\hat{\mathbf{B}}$ and $\hat{\boldsymbol{\pi}}$ and define an adjusted model $\hat{\lambda} = (\hat{\mathbf{A}}, \hat{\mathbf{B}}, \hat{\boldsymbol{\pi}})$.
3. If $P(\mathcal{O}|\hat{\lambda}) > P(\mathcal{O}|\lambda)$, set $\lambda = \hat{\lambda}$ and return to step 1. Repeat until a limiting point is reached.

Chapter 3

Modeling activities through HMMs

3.1 Introduction

As mentioned earlier, HMMs were first successfully applied in the field of speech recognition. HMMs are a suitable modeling choice because they are able to incorporate the sequential quality of speech. Typical daily human activities share this sequential quality. For example, in the same way words are divided into phonemes, activities can be separated into a number of distinct parts.

The first section of this chapter is about the experimental data set we perform our experiments on. Here we go into the details of the acquisition of kinematic data, which includes a description of the sensors, the locations where they are placed, how many are needed and the type of data that is recorded. This is followed by a section on the application of HMMs to physical activities and a discussion of the initialization of the HMM parameters and the training and testing processes of the classifier. The final section is about the software implementation of the algorithms described in the previous chapter.

3.2 Experimental data

For our experiments, we consider a data set consisting of a series of lifting activities. The eight activities in this data set are 'walking' (1), 'standing still' (2) (standing upright with feet centered), 'sitting' (3), lifting an object by 'stooping' (4) (bending without using your knees) or 'squatting' (5) (bending using your knees), lifting an object positioned to the 'right' (6) or to the 'left' (7), and putting the object 'down' again (8). The typical order in which these activities are carried out would for example be: 'walking' to an object, 'standing still', picking up the object using one of the lifting techniques, 'standing still' again, 'walking' somewhere else and putting the object 'down'.

3.2.1 Data acquisition and processing

For the recording of data we use MTw motion tracking sensors [17], developed by the Enschede based company Xsens [8]. These sensors record three-dimensional angular velocity, acceleration and magnetic field data. A sensor cannot be placed in the exact same spot every time. Because of this, and to make the data comparable to data recorded by other sensors placed on the subject, some segment calibration measurements are performed. These calibration measurement are needed to determine the orientation of each sensor worn by a subject, and are used to rotate the sensor coordinate systems to a universal coordinate system. See figure 3.1 for a schematic representation of this process.

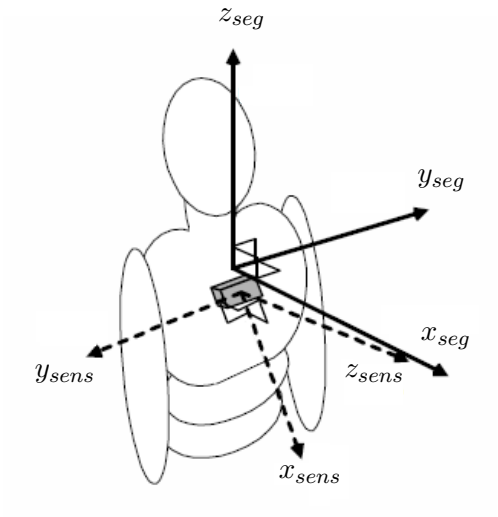


Figure 3.1: The dashed arrows represent the sensor orientation on the subject, and the solid arrows represent the segment orientation in the universal coordinate system.

The sensors are wireless and weigh approximately 27 grams, so they can be worn easily and unobtrusively by any subject. We want the sensor configuration to be minimal for the subject: as few sensors as possible, positioned as unobtrusively as possible. In the previous RRD study with patients with DMD [44], sensors were placed on the trunk, the head, the lower and upper dominant arm and the dominant hand. For the subject it would be much better if we could lose the sensor positioned on the head.

The recorded data is cleaned up and low pass filtered. The cut off frequency of the low pass filter for the acceleration data is 7 Hz, the cut off frequency for the gyroscopic data is 5 Hz. The filtering is performed to obtain a more smooth signal in order to apply down sampling. The resulting signal has a frequency of 25 Hz. The reason to perform down sampling is human movement cannot exceed a frequency of ~ 7 Hz.

3.3 Application of HMMs

It is possible to observe human activities as a sequence of smaller, basic movements. For a specific activity, the order of these basic movements is always the same. For example, the activity 'picking up an object' can be split into a sequence of five separate actions, namely (1) a downward acceleration; (2) a downward deceleration; (3) no movement; (4) an upward acceleration; and finally (5) an upward deceleration.

To understand a typical human activity in terms of a HMM, we will start with the hidden Markov chain and what it should represent. As a starting point, and to impose a temporal order to the model, we choose for a left-to-right architecture, with delta equal to 1 (see also 2.3.2). We let each state in this representation correspond to one of the basic movements of an activity. For the activity 'picking up an object' as described above, the number of states then will be five. Because of the chosen architecture each of these states must be visited, which also means that it is not possible to skip any of the basic movements of the activity. With this we have enough information to implicitly define the transition probability matrix $\mathbf{A}$. It should be an upper triangular matrix with all elements equal to zero, except those on and directly above the diagonal.

The second model parameter, namely the initial state distribution, is easy for the chosen Markov chain architecture. Since we are only considering left-to-right models, all chains must start from the left-most state. This means that the initial distribution will be of the form $\boldsymbol{\pi} = \{1, 0, \ldots, 0\}$.

The third model parameter is the signal probability distribution in each state. We have assumed that a state corresponds to a small, basic movement, which is characterized by the data being stationary in some sense (e.g. downward acceleration or no movement at all). In the theoretical ideal case, where specific human activities are always performed in the exact same way, and any resulting recorded data is very 'clean', it would be possible to observe these stationary parts across all parallel data channels. The resulting sequence of these stationary parts then makes up a complete activity.

To some extent it is indeed possible to observe these stationary parts in real life measurements. See for example figure 3.2. To account for noise in the signal and variation between different examples of the same activity, we model each state by a normal distribution with mean μ and covariance matrix Σ. Data points can then be probabilistically matched to whatever state was most likely to have generated them. We now have covered all parameters $\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$ that make up a HMM. For each activity that needs to be classified, a separate HMM needs to be defined. The total of these activities make up our dictionary.

3.3.1 Initialization

Before the training algorithms can be applied, the HMM parameters need to be initialized. The initial distribution $\boldsymbol{\pi}$ is always of the form $\{1, 0, \ldots, 0\}$ as described above. For the initialization of the transition probability matrix the only thing that

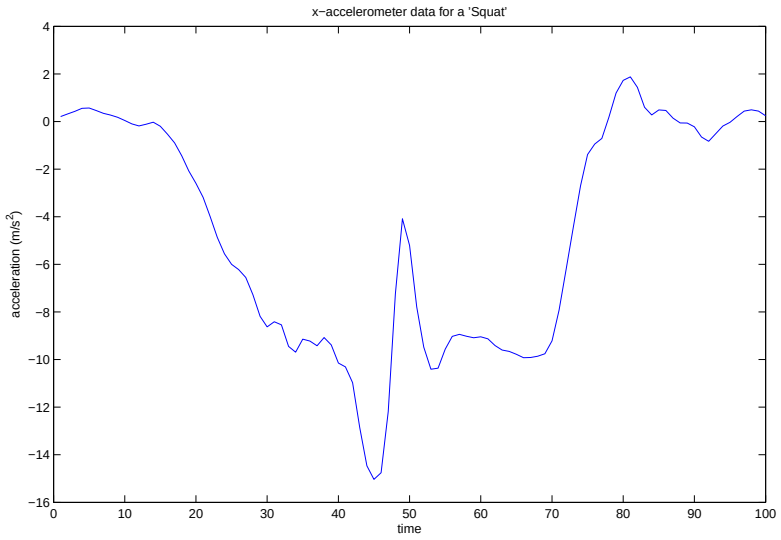


Figure 3.2: An example of an accelerometer data channel of a subject performing a 'squat'. The sensor was positioned on the spine. It is possible to observe stationary parts within the time series.

is important is that all illegal state transitions have probability zero. The nonzero elements can be given any positive value between zero and one, the only constraint is that each row of the matrix should sum up to one.

For the initialization of the signal probability distribution the values of μ and Σ need to be specified. In contrast to the matrix $\mathbf{A}$, here the initial values play a more important role because they determine the initial location of the hidden Markov chain states within the state space. Since the Baum-Welch algorithm is a local optimizer, and since it is as of yet unknown to us how 'mountainous' the space is where Baum-Welch operates, a smart choice of the initial values of μ and Σ seems essential.

However, for our preliminary testing, we will provide all HMMs with a 'flat' start. All μ_j and Σ_j will be given the value of the mean and variance of the complete training dataset. We will return to this question in chapter 5 and address it in more detail.

3.3.2 Training and testing

To train the HMMs, we need to provide a set of training data. This training set consists of a sequence of manually labeled examples of each of the activities from the dictionary. The activities are recorded in a random order, to make sure the classifier is not trained to recognize them in a certain order.

Each example of a certain activity is used to train the corresponding HMM

through application of the iterative Baum-Welch algorithm (see section 2.4.3). An open question is how often to iterate it. If a HMM goes through too many iterations, it may adjust to very specific random features of the training data, that have little or nothing to do with the target activity. When the model overfits the data, the performance on the training examples may still increases while at the same time the performance on unseen data becomes worse. For the case of speech recognition, the HTK manual [43] suggests that five iterations should be sufficient. Compared to speech, our data is of a higher dimension, so maybe a few more iterations will be needed.

The trained models can now be provided with an independent test data set. The test data set again consists of a sequence of randomly ordered activities. The Viterbi algorithm (see section 2.4.2) is applied to transcribe the data.

3.4 Software implementation

All the above algorithms and functionalities are implemented using the Hidden Markov Toolkit software package (HTK 3.4), which was developed at the Machine Intelligence Laboratory of the Cambridge University Engineering Department [1]. The toolkit is designed to run with a traditional command-line style interface. At RRD a shell was built around the HTK tools in a Labview environment. This was done to increase the user-friendliness of the software and make it possible to run large automated batches of train-and-test sessions with varying parameters.

An important aspect of the implementation that needs to be addressed here is the handling of probabilities. Both the Baum-Welch algorithm (through the Forward-Backward calculations) and the Viterbi algorithm need to calculate long products of probabilities. These products tend to zero exponentially fast, so there is the risk of numerical underflow. To avoid this problem, we use the logarithm of the probabilities instead. This means that all products of probabilities are transformed into sums.

3.4.1 HERest training tool

The training procedure is carried out by the HTK tool called HERest, which applies the Baum-Welch algorithm. HERest requires a transcripted sequence of continuous activities as its input. It concatenates all the HMMs corresponding to the training sequence to construct a single composite HMM. The training examples of the various activities are processed to re-estimate all parameters of each individual HMM within the composite HMM. In this process, the transcriptions are only used to identify the sequence of activities; it is not necessary to include the start and end times of individual activities.

As mentioned in section 3.3.1, all HMMs are initialized through a flat start: each hidden Markov state is assigned the values of the global mean and variance of the complete training data set. Note that the dimension of both the mean and variance

is determined by the number of input data channels, and hence we consider the values of the global mean and variance per data channel. The flat start procedure implies that during the first re-estimation cycle, each training example of an activity will be uniformly segmented. The hope then is that enough hidden Markov states align with their actual realizations (i.e. the smaller basic movements that make up activities), so that after the subsequent training iterations the HMMs align to the training data as intended. Prior studies suggest that this is indeed what happens.

3.4.2 HVite recognition tool

The classification process is handled by the HTK tool HVite. The recognition of a single activity can be achieved through straightforward application of the Viterbi algorithm as described in the chapter on HMMs. In theory this would work as follows. Recall again that for each activity a separate HMM is defined. For each of these HMMs the total probability that some observed data sequence was generated by one of these HMMs can be calculated through application of the Viterbi algorithm. The most likely HMM then must have been the one to have generated the observation and the classification is complete.

In practice however, test data will consist of a sequence of unknown activities, the start and end times of which are also unknown. HVite extends the Viterbi algorithm in the following way to handle sequential activity recognition [42]. All separate activity HMMs are connected in a looped composite model (see figure 3.3). It is again possible to apply the Viterbi algorithm to this model to calculate the most likely sequence of states through the composite network. The difference is that it now also operates on a macro (whole activities) level instead of just on a micro (hidden Markov state) level. The output of the HVite function is the most likely sequence of activities, including activity boundary information and the total log probability for each recognized activity.

Another internal HTK parameter that influences the recognition, is the word insertion penalty p. This parameter p influences the number of expected activities within an activity sequence by adding an offset to the log probability of a possible classified activity. A larger value of p means that you expect a smaller number of activities, which will result in less false positives, i.e. recognizing something where there is nothing to recognize. At the same time, a larger penalty also means that the number of false negatives, i.e. missing an activity, will increase. Within the Labview environment, some automated testing is performed to find the optimal value of the word insertion penalty. Optimal here means that the total number of errors is as small as possible.

3.4.3 HResults evaluation tool

The HTK tool HResults analyzes the performance of generated HMMs. It reads in the transcription generated by HVite and compares it with the corresponding reference transcription of the test data. The classifier can make three types of

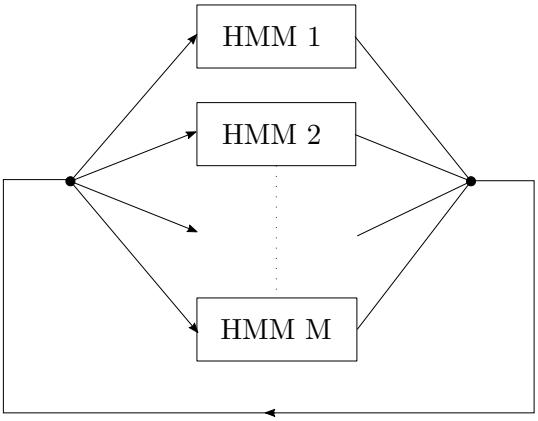


Figure 3.3: Given an unknown sequence of activity data, HVite connects all the HMMs in the dictionary in a looped composite model and calculates the path through the network that is most likely to have generated the data.

mistakes. It could recognize the wrong activity, which we will call a substitution error S. Also, it could miss an activity, which we will call a false negative or deletion error D. Lastly, it could recognize an activity where there is nothing to classify, which we will call a false positive or insertion error I.

The last error type is usually the worst. If the software classifies some activity, you want to be as certain as possible that there actually was something to classify. If the number of insertion errors becomes too large, it can make you doubt all your results. In case of a substitution error, at least you are certain that there is an actual activity to classify. In case of deletion error, you are missing an activity but the accuracy of the other results does not decrease. This is why we think avoiding insertion errors is the most important.

HResults uses the following output statistics:

$$\text{Correctness} = \frac{N - D - S}{N} \cdot 100\%$$
$$\text{Accuracy} = \frac{N - D - S - I}{N} \cdot 100\%,$$

Where N is the total number of activities in the test data set. Note that theoretically the Accuracy could become negative, but in some cases this number might be a more representative measure of the performance of the recognizer than the correctness.

3.4.4 HTK flowchart

A schematic representation of the HTK software implementation is given in figure 3.4. Compare also with the flowchart presented in figure 2.6.

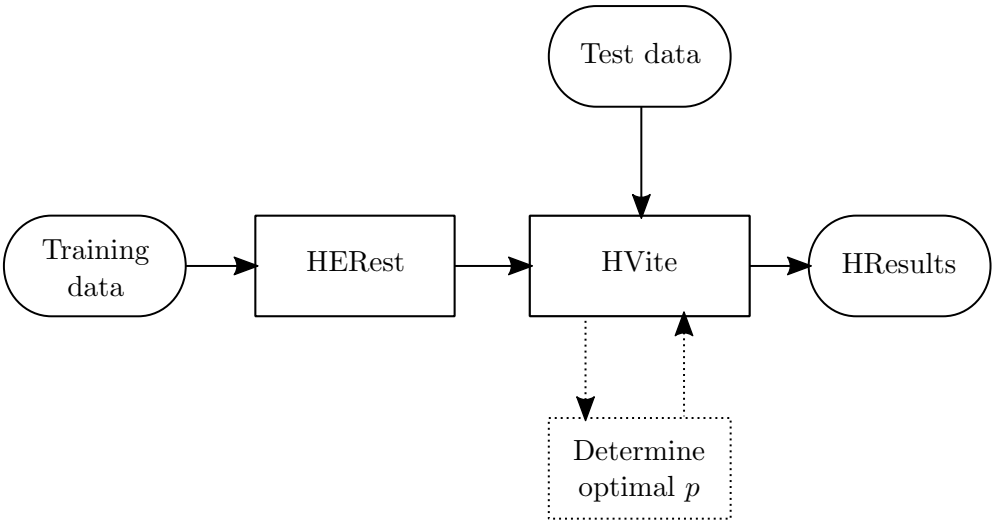


Figure 3.4: The training of the HMMs is carried out by HERest. Recognition of independent test data is performed by HVite. The results are evaluated by HResults.

Chapter 4

Data preprocessing

4.1 Introduction

To make the training process more efficient and to improve the classification results of the HMM-based classifier, we apply preprocessing to optimize the data that go into the model. Optimizing data means maximizing the amount of information and minimizing the amount of redundant data (noise) they contain. A previous study [31] also looked into this problem and considered several preprocessing techniques. The most promising of these techniques is Principal Components Analysis (henceforth PCA).

Other studies also combined HMMs with PCA. See for example [27] on face recognition or [37] on handwriting recognition. More relevant to our problem are for example [3] and [9], who applied PCA on motion data before training the HMMs. In this chapter we will explain the theory of principal components analysis, discuss its limitations and investigate its applicability to our problem.

4.2 Goals of data optimization

First we need to define what maximizing the amount of information and minimizing the amount of redundancy and noise exactly means. When constructing a HMM to optimally describe a certain activity, it is difficult to know beforehand which elements of that activity best capture its essence and need to be represented in the HMM. In this discussion, we will consider processed data that leads to better classification results to be more optimal data.

With each extra sensor a subject wears, six extra channels of data are recorded. While extra data may increase the ability of the classifier to differentiate between different activities, there is a tradeoff. Too much channels may result in poorer classification accuracy because more data also brings along more redundancy and noise. On top of this, extra data channels result in greater overall computation time in the training and testing phases. What is needed is a way to manage the complete recorded data set.

An important aspect of data optimization is dimensionality reduction. Not all of the recorded channels will contain the same amount of information. Dimensionality reduction is the process of reducing the number of variables under consideration, while preserving as much essential information as possible.

4.3 Theory of PCA

Principal Components Analysis (henceforth PCA) combines all of the aspects described in the previous paragraph. The idea of PCA is as follows. Imagine recording in three dimensions the motion of a ball hanging from a spring (example inspired by [29]). Most likely, the largest part of the variation in the recorded data will be in the direction of the spring (the vertical z-axis). Any movement of the ball in the direction of the x and y axes can be regarded as noisy perturbations around the z-axis.

Now imagine a more general experiment where it is not so clear beforehand how to choose the direction of the axes you are recording and how many dimensions you should take into consideration. PCA looks for a translation and rotation of the recorded axes to a new coordinate system where the amount of information in each dimension is optimized. This way, it becomes easier to discover underlying structure in the data. In case of the ball-on-a-string experiment, where you recorded three-dimensional data but did not know the best way to choose the axes along which to record, the underlying structure revealed by PCA will tell you the direction that contains most variation (or information) and hence the most likely direction of the spring.

In a more general sense, PCA transforms the data from the original space to a new space of the same dimension. Through this transformation, PCA may make it easier to see that the data are in fact grouped around a manifold of much lower dimension than the original space where the data were recorded. This could be helpful when you are trying to reduce the dimensionality of the data. Dimensions that contain little variation, and hence also little information, may be discarded before feeding the data set to the classifier.

There are two main interpretations of PCA. It can be defined as the orthogonal projection of the data onto a lower dimensional linear space such that the variance of the projected data is maximized [14]. An alternative way of looking at it is considering it as the linear projection that minimizes the average projection cost, defined as the mean squared distance between the data points and their projections [21]. Here we will explain the theory through examining the first interpretation, which we will call the maximum variance formulation. For the explanation of the theory, [6] proved very useful.

4.3.1 Maximum variance formulation

Consider a set of D-dimensional, observation vectors $\{\mathbf{x}_n\}$, $n = 1, \ldots, N$. The aim is to project the data onto a lower dimensional subspace having dimension $M \leq D$, while maximizing the variance of the projected data.

For simplicity, let us first consider projecting the data onto a one-dimensional space. We can describe this new space using a D-dimensional vector $\mathbf{u}_1$ with length 1 (i.e. $\|\mathbf{u}_1\| = 1$). Each observation vector $\mathbf{x}_n$ is projected onto a scalar $\mathbf{u}_1^T\mathbf{x}_n$. The mean of the projected data is then given by $\mathbf{u}_1^T\overline{\mathbf{x}}$, where

$$\overline{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n, \tag{4.1}$$

the variance of the projected data is given by

$$\frac{1}{N}\sum_{n=1}^{N}\{\mathbf{u}_1^T\mathbf{x}_n - \mathbf{u}_1^T\overline{\mathbf{x}}\}^2 = \mathbf{u}_1^T\mathbf{S}\mathbf{u}_1, \tag{4.2}$$

where $\mathbf{S}$ is the covariance matrix defined by

$$\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \overline{\mathbf{x}})(\mathbf{x}_n - \overline{\mathbf{x}})^T. \tag{4.3}$$

To find the direction for which the projected variance is maximal, we maximize equation (4.2) with respect to the vector $\mathbf{u}_1$. Note that the fact that we have chosen $\|\mathbf{u}_1\| = 1$ makes this a constrained optimization and we have arrived at the following optimization problem:

$$\begin{cases} \max \mathbf{u}_1^T\mathbf{S}\mathbf{u}_1, \\ \text{s.t. } \|\mathbf{u}_1\| = 1. \end{cases} \tag{4.4}$$

To find $\mathbf{u}_1$, introduce a Lagrange multiplier λ_1 and solve the unconstrained maximization problem

$$\max_{\mathbf{u}_1} \mathbf{u}_1^T\mathbf{S}\mathbf{u}_1 + \lambda_1(1 - \|\mathbf{u}_1\|). \tag{4.5}$$

Calculating the derivative with respect to $\mathbf{u}_1$ and setting the result equal to zero, gives

$$\mathbf{S}\mathbf{u}_1 = \lambda_1\mathbf{u}_1. \tag{4.6}$$

This means that $\mathbf{u}_1$ must be an eigenvector of the covariance matrix $\mathbf{S}$. Left-multiplying this equation by $\mathbf{u}_1^T$ and using that $\|\mathbf{u}_1\| = 1$,

$$\mathbf{u}_1^T\mathbf{S}\mathbf{u}_1 = \lambda_1\mathbf{u}_1^T\mathbf{u}_1 = \lambda_1, \tag{4.7}$$

we see that the variance of the projected data is equal to the eigenvalue λ_1. This means that the variance is maximal for the eigenvector $\mathbf{u}_1$ associated with the largest eigenvalue λ_1 of the covariance matrix $\mathbf{S}$. We call this eigenvector $\mathbf{u}_1$ the first principal component.

Since the covariance matrix is a symmetric matrix, all its eigenvectors are orthogonal. This means that we can define the additional principal components $\mathbf{u}_2, \ldots, \mathbf{u}_M$ in an incremental fashion by choosing them to be the eigenvectors associated with the remaining $M-1$ largest eigenvalues $\lambda_2, \ldots, \lambda_M$. The amount of variation explained by component i is given by

$$\frac{\lambda_i}{\sum_{j=1}^{M} \lambda_j}. \tag{4.8}$$

Using the eigenvectors, we can define a transformation matrix $P = [\mathbf{u}_1 \cdots \mathbf{u}_M]$. The original data $X = [\mathbf{x}_1 \cdots \mathbf{x}_N]$ can now be expressed in the new principal component space Y through the following transformation:

$$Y = P^T X. \tag{4.9}$$

4.3.2 Limitations of PCA and application to motion data

PCA can only be meaningfully applied to a data set if a number of assumptions is satisfied [29]. The most important motivation of applying PCA is to decorrelate a data set. For this to work, it must be true that the original data is arranged along orthogonal axes. Further, decorrelation also removes any second or higher order dependencies within the data, so applying PCA to data that do contain higher order dependencies leads to loss of information.

Do these assumptions hold when considering the motion data in our lifting experiment? The recorded data consist of accelerometer and gyroscope data. Naturally, any x-, y- and z-accelerometer and gyroscope data channels are orthogonal. Further, there are no second or higher order dependencies between any of these channels; since we consider freely moving subjects, any value of the acceleration could be combined with any angular velocity.

The third assumption is that we consider a component with low variance to have low information. In other words, if a component has a low variance, we say it must represent noise. This last assumption has an important consequence. Our data consist of different channels. Some of these channels will be of a greater order of magnitude than others. For example, z-accelerometer channels are influenced by gravitation, so on average the measured values will be greater than those of the x- and y-accelerometer channels. Of course this does not necessarily mean that the z-channel contains more information. Simple normalization does not solve this problem, because then the ratio between the channels is not preserved and information might be lost.

Our solution is as follows. Like with normalization we center the data, but instead of dividing by the standard deviation of the channel under consideration,

we divide by the standard deviation per signal type. This means that we calculate the standard deviation of all accelerometer channels combined and use it instead for the normalization procedure of a single accelerometer channel. For example, for any x-accelerometer channel, this works as follows:

$$x_{norm}^{acc} = \frac{x^{acc} - \bar{x}^{acc}}{\sigma_{total}^{acc}}$$

This way, we make sure that the accelerometer channel with the greatest variance indeed corresponds to the channel that contains the most information. The same strategy is applied to the gyroscope channels.

Chapter 5

Improving the training process

5.1 Overview of this chapter

In the previous chapters, we have described the full process of describing activities through hidden Markov models. In this chapter we will look at several aspects of the training process that can be improved. Firstly, we will discuss how to choose the number of Markov states that best describes a certain activity. Secondly, we will present a detailed discussion of how HTK applies the Baum-Welch algorithm to train a HMM, and discuss how to improve the results of the training. In the final section we will go into choosing the best possible initial values of those hidden states.

5.2 Optimizing the number of Markov states

When using HTK to train a HMM, the number of hidden states in the model needs to be specified. This number remains fixed throughout the training process. We say that for a certain activity i the number of states N, $N > 0$, is optimal if it maximizes the accuracy of the recognition results on an independent test data set:

$$N_{\text{optimal}}^{\text{act}_i} := \max_N \{\text{Accuracy}(N)\} \tag{5.1}$$

In practice, what will typically happen is that as the number of states increases from $N = 1$, the accuracy improves. At some point the number of states is sufficient to capture the essence of the activity you are trying to describe. Additional states will be redundant, since there are no more elements of the activity that are not already described by the existing states. The extra states only give meaning to noise, something which is highly undesirable since it will lead to deterioration of the recognition accuracy.

Currently, the software model trains all activities in the dictionary with the same number of states. Naturally, this is not the best way to describe different activities. More complex activities might require more states to best capture their

essence; similar activities require a sufficient number of states to optimally model their differences. Through an artificial data experiment, we will illustrate what exactly can be improved. We also present an iterative approach to find a good estimate of what the optimal number of states for an activity should be.

5.2.1 Artificial data experiment

To illustrate why the choice of the number of states for a HMM is highly important, we conduct the following experiment on an artificial data set. We define two artificial activities in the following way (see also figure 5.1). Activity 1 consists of three states. The time spent in the respective states is normally distributed with $N(\mu_i, \sigma_i^2)$, where $\mu = \{5, 5, 10\}$ and $\sigma^2 = \{0.75, 0.75, 1\}$. The signal distribution in the respective states is normally distributed with $N(\mu_{signal,i}, \sigma^2_{signal,i})$, where $\mu_{signal} = \{4, 6, 10\}$ and $\sigma^2_{signal} = \{1, 1, 1\}$. Activity 2 is defined in the exact same way, only states 1 and 2 are interchanged. We randomly generate a combined total of 100 randomly ordered examples of activities 1 and 2. This artificially generated data set can be used to get a feel of how the classifier handles data. It will also give us an idea of how running the training algorithms results in trained HMMs and how we could improve upon the results.

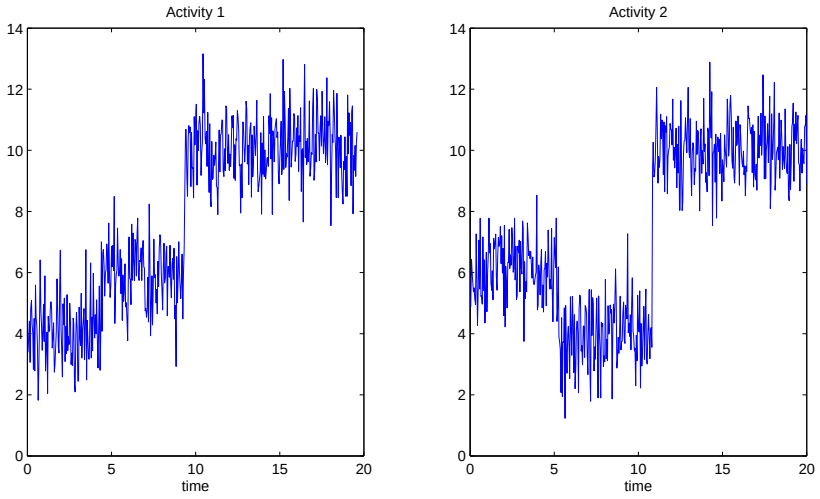


Figure 5.1: Examples of artificial activities 1 and 2.

We divide the data set into a training set consisting of 75 examples of both activities and a test set consisting of the remaining 25. In the first part of the experiment we choose the number of hidden states for both activities to be two. Remember that the 'actual' number of states of both activities is three. Naturally, we expect the results to be poor because of the way activities 1 and 2 are defined; two states cannot be sufficient to model the difference between both artificial activities

so we do not expect the correctness of the results to be much better than 50%. Finally, both HMMs are initialized using a 'flat' start, as described in section 3.3.1, and the Baum-Welch training algorithm is iterated four times.

In the second part of the experiment we choose the number of hidden states for both activities to be three. Three states should be sufficient to model the difference between both artificial activities, so we expect highly accurate recognition results. Again the HMMs are initialized using a 'flat' start and the Baum-Welch training algorithm is iterated four times.

Results

In the two-state experiment all parameter values found by the Baum-Welch algorithm are practically equal for both HMMs (see table 5.1 and figure 5.2). The correctness and accuracy (see also section 3.4.3) of the recognition results are 72.3% and 66.7% respectively. Although these numbers are better than the expected 50%, clearly improvement is needed.

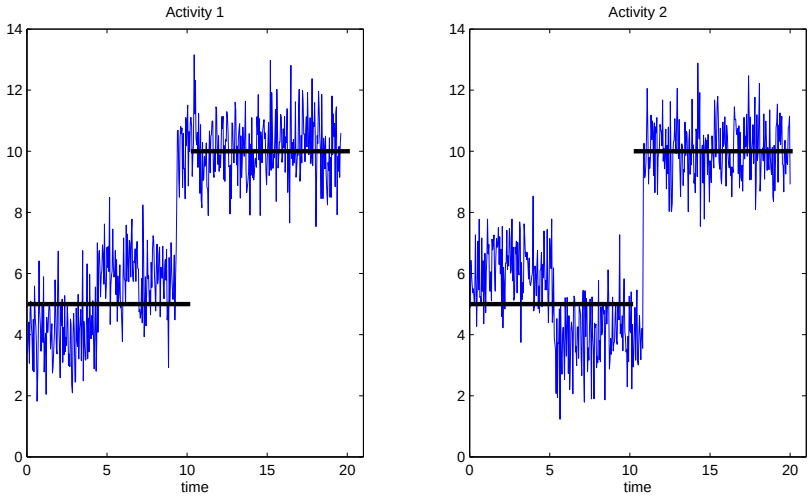


Figure 5.2: Seen here are the means of both states determined by the Baum-Welch algorithm in the two-state experiment, plotted on top of the same examples as seen in figure 5.1

The results of the three-state experiment are much more promising. The correctness and accuracy are now both a perfect 100%. The parameter values found by the Baum-Welch algorithm are presented in table 5.2 and figure 5.3. From the results we notice two things. Firstly, the reconstruction of activity 1 is very accurate. The obtained parameter values lie very close to the original distributions. Secondly, we observe that this not at all the case for activity 2. Since so little time is spent in the second state, the reconstruction basically consists of only two states.

	Activity 1	Activity 2
State 1	$(\mu_1, \sigma_1^2) = (4.99, 1.98)$ $E[T_1] = 10.22$	$(\mu_2, \sigma_2^2) = (5.00, 2.03)$ $E[T_1] = 10.10$
State 2	$(\mu_2, \sigma_2^2) = (10.00, 1.03)$ $E[T_2] = 9.96$	$(\mu_2, \sigma_2^2) = (10.00, 1.01)$ $E[T_2] = 10.15$

Table 5.1: Parameter values determined by the Baum-Welch algorithm for the two-state experiment. $E[T_i]$ is the average time spent in state i.

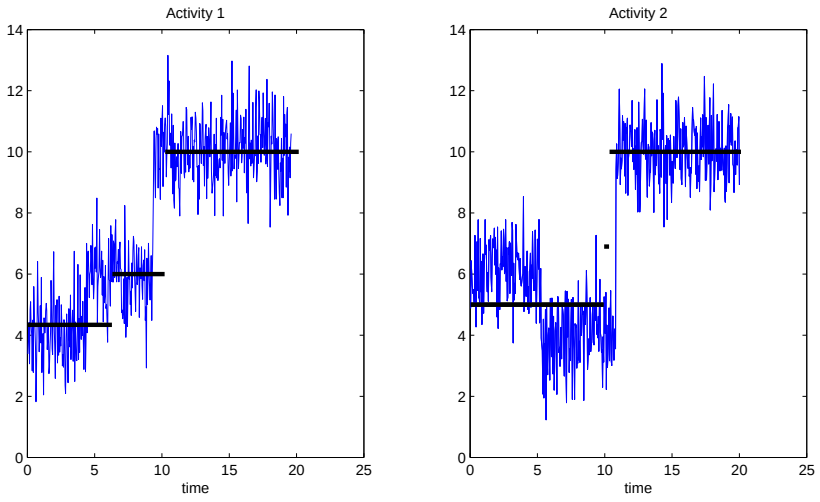


Figure 5.3: Seen here are the means of all three states determined by the Baum-Welch algorithm in the three-state experiment, again plotted on top of the same examples as seen in figure 5.1. Note that the time spent in the second state for activity 2 is very short.

	Activity 1	Activity 2
State 1	$(\mu_1, \sigma_1^2) = (4.34, 1.50)$ $E[T_1] = 6.29$	$(\mu_1, \sigma_1^2) = (5.00, 2.03)$ $E[T_1] = 10.07$
State 2	$(\mu_2, \sigma_2^2) = (6.03, 1.02)$ $E[T_2] = 3.93$	$(\mu_2, \sigma_2^2) = (6.90, 1.09)$ $E[T_2] = 0.063$
State 3	$(\mu_3, \sigma_3^2) = (10.00, 1.03)$ $E[T_3] = 9.95$	$(\mu_3, \sigma_3^2) = (9.99, 1.00)$ $E[T_3] = 10.12$

Table 5.2: Parameter values determined by the Baum-Welch algorithm for the three-state experiment.

We conclude that the main reason for the results being so good, is because the reconstruction of activity 1 is so accurate. The accurate reconstruction not only

makes it easy to recognize whether a data sequence was generated by the HMM describing activity 1, it also makes it easier to recognize whether a data sequence was *not* generated by the HMM describing activity 1 and the reconstruction of activity 2 does not need to be as good. This works out fine for our artificial experiment where we only have two different activities to consider, but it likely will cause problems when the number of activities increases.

From this experiment we learn two things. We have motivated the importance of choosing a sufficient number of states to describe activities. Secondly, as can be seen from the three-state experiment, choosing the right number of states does not necessarily lead to the best possible reconstruction of the activities. We hope to solve this through smarter initialization of the state distributions. We will return to this problem later in this chapter.

5.2.2 Iterative solution

The first question from the previous section is how to find the optimal number of hidden Markov states for each individual activity. We attempt the following strategy to find this number. For each individual activity, set the initial number of states to a not too high value N_0. Remember that each state corresponds to a small basic element of a larger movement (see also section 3.2), so it is reasonable to assume that it is not possible to describe an activity with less than four states. Hence, when classifying the lifting activities described in section 3.2, $N_0 = 4$ should be sufficient. Next, train the HMM with this number of states, test it on independent data and store the accuracy of the recognition results. Increment the number of states by one, and repeat the process. Keep doing this until the accuracy starts decreasing (or does not improve anymore). In pseudocode:

```
for all activities do
    Set initial number of states to N = N_0;
    Train HMM;
    Test model on independent data;
    Store accuracy of results;
    while accuracy improves do
        Set number of states to N = N + 1;
        Train HMM;
        Test model on unknown data;
        Calculate accuracy
    end
    N^{act_i}_{optimal} = N - 1;
end
```

5.3 Optimizing Baum-Welch training

Training of the HMMs through application of the Baum-Welch algorithm takes places on a composite HMM generated by the HTK tool HERest (see also section 3.4.1). This composite HMM consists of a concatenation of all HMMs corresponding to the activities in the training set.

An important downside of this approach is the following. Different activities have different average durations. HERest assumes that these different durations do not vary too much, and hence enough HMMs within the composite HMM will align with their realizations in the training set. If one activity is significantly longer than the other, the misalignment of the HMMs and their realizations might become too severe and lead to deterioration of the recognition results. We will illustrate this effect with an artificial data experiment.

5.3.1 Artificial data experiment

In similar fashion to the experiment conducted in section 5.2.1, define the following two artificial activities. Activity 1 consists of four states. The time spent in the respective states is normally distributed with $N(\mu_i, \sigma_i^2)$, where $\mu = \{10, 10, 10, 10\}$ and $\sigma^2 = \{1, 1, 1, 1\}$. The signal distribution in the respective states is normally distributed with $N(\mu_{signal,i}, \sigma^2_{signal,i})$, where $\mu_{signal} = \{6, 4, 6, 4\}$ and $\sigma^2_{signal} = \{1, 1, 1, 1\}$. Activity 2 also consists of four states. The time spent in the respective states is again normally distributed with $N(\mu_i, \sigma_i^2)$, where $\mu = \{25, 25, 25, 25\}$ and $\sigma^2 = \{1, 1, 1, 1\}$. The signal distribution in the respective states is normally distributed with $N(\mu_{signal,i}, \sigma^2_{signal,i})$, where $\mu_{signal} = \{2, 4, 6, 2\}$ and $\sigma^2_{signal} = \{1, 1, 1, 1\}$. See figure 5.4 for an example of both these activities. The key difference between both these artificial activities is that activity 2 has a much greater average duration than activity 1. We randomly generate a combined total of 100 randomly ordered examples of activities 1 and 2. Activity 1 is four times more likely to occur than activity 2, so on average we expect 80 example of activity 1 and 20 examples of activity 2 in the data set. This is done to try and increase the misalignment between the composite training HMM generated by HERest and the activities in the data set.

For the first part of the experiment, the data set is randomly divided into a training set consisting of 75 activities, and an independent test set consisting of the remaining 25. For both activities, the HMMs are initialized using a 'flat' start, as described in sections 3.3.1, and the Baum-Welch training algorithm is iterated until the recognition results do not improve any further. Naturally, both HMMs consist of four hidden Markov states.

In the second part of the experiment, using the annotations we sort the data in the training set. The first subset of the training set consists only of examples of activity 1, the second subset of the training set consists only of examples of activity 2. The third and final subset of the training set consists of a randomly ordered mix of both activities. The test part of the data set naturally also consists of a randomly

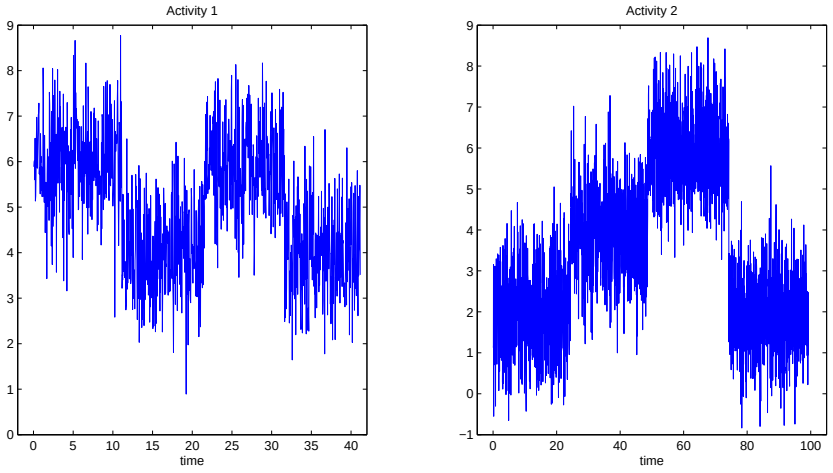


Figure 5.4: Examples of artificial activities 1 and 2. Note that the duration of activity 2 is much greater.

ordered mix of both activities. The size of the training and test sets is still 75 and 25 respectively.

The HMMs for activity 1 and 2 are first trained using their respective specific training sets. After this follows a second training session using the mixed training set. Finally the HMMs are tested on the mixed test set.

Activity 1 is again four times more likely to occur than activity 2. The number of hidden states for both activities is again chosen to be four. The HMMs are initialized using a 'flat' start and the Baum-Welch training algorithm is iterated until the recognition results do not improve any further.

Results

For the first part of the experiment, the recognition results are poor. After three iterations of the Baum-Welch algorithm, the results do not improve any further. The correctness and accuracy are both only 28.0%. See table 5.3 and figure 5.5 for a detailed overview of the results.

The problem seems to be that for both activities, the HMMs are stuck in some representation of the original activities that does not correspond to four states. For activity 1 the third state is degenerate since so little time is spent there. For activity 2 it is even worse: the means of the last three states almost have the same value, so basically the solution corresponds to only two instead of four states.

For the second part of the experiment, the recognition results are almost perfect (see table 5.4 and figure 5.6 for a detailed overview of the results). The correctness and accuracy of the results are 100% and 96% respectively, which corresponds to only a single insertion error (false positive). The results are obtained after five

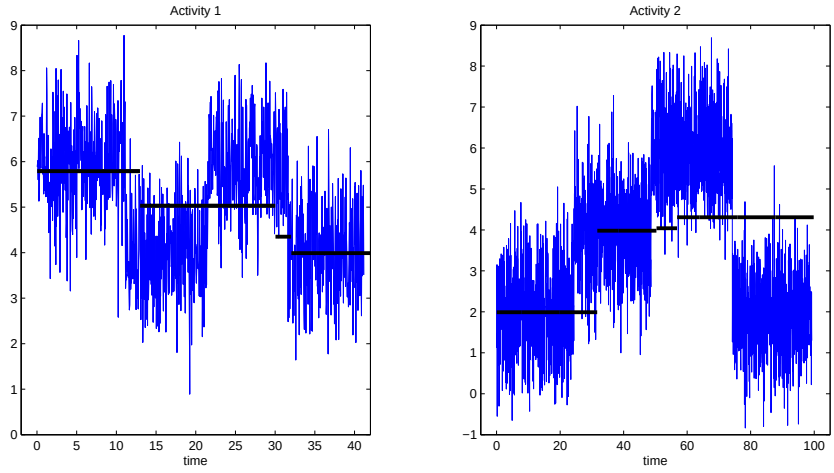


Figure 5.5: Seen here are the means of all four states determined by the Baum-Welch algorithm for the first part of the experiment, plotted on top of the same examples as seen in figure 5.4

	Activity 1	Activity 2
State 1	$(\mu_1, \sigma_1^2) = (5.79, 1.38)$ $E[T_1] = 12.96$	$(\mu_1, \sigma_1^2) = (1.99, 1.00)$ $E[T_1] = 31.68$
State 2	$(\mu_2, \sigma_2^2) = (5.03, 1.95)$ $E[T_2] = 17.04$	$(\mu_2, \sigma_2^2) = (3.98, 0.98)$ $E[T_2] = 18.64$
State 3	$(\mu_3, \sigma_3^2) = (4.35, 1.45)$ $E[T_3] = 0.40$	$(\mu_3, \sigma_3^2) = (4.04, 1.05)$ $E[T_3] = 6.48$
State 4	$(\mu_4, \sigma_4^2) = (3.99, 0.98)$ $E[T_4] = 9.96$	$(\mu_4, \sigma_4^2) = (4.31, 4.86)$ $E[T_4] = 43.04$

Table 5.3: Parameter values determined after three iterations of the Baum-Welch algorithm for the first part of the experiment.

iterations of the Baum-Welch algorithm. The problem with activity 1 is still that the third state is degenerate, since again very little time is spent there. Activity 2 on the other hand is now recovered almost perfectly. Like in the previous section we conclude that the main reason that the results have improved so much is because now at least one of both activities is accurately reconstructed.

From this experiment, we see that we should indeed be careful when training the classifier on a dictionary containing elements that vary too much in duration. The solution to this problem is relatively straightforward, as we have shown in the second part of the experiment. Secondly, as the second part of the experiment also shows, choosing the correct number of states *and* training the HMMs with individ-

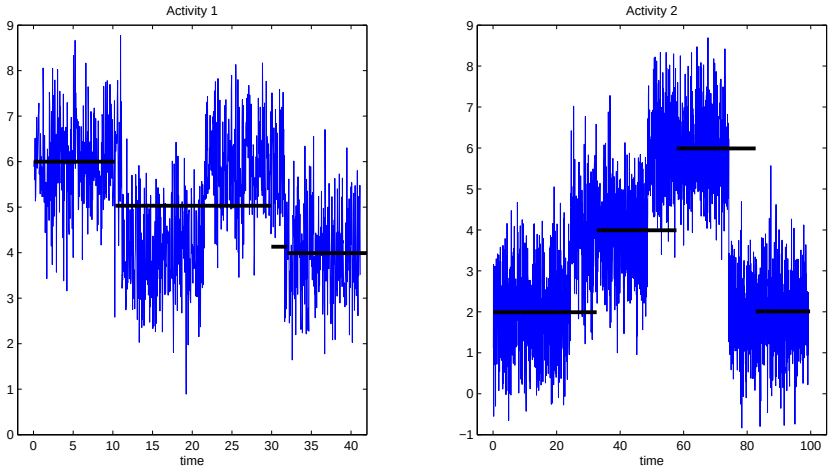


Figure 5.6: Seen here are the means of all four states determined by the Baum-Welch algorithm for the second part of the experiment, plotted again on top of the same examples as seen in figure 5.4

	Activity 1	Activity 2
State 1	$(\mu_1, \sigma_1^2) = (6.00, 0.98)$ $E[T_1] = 10.28$	$(\mu_1, \sigma_1^2) = (1.99, 0.99)$ $E[T_1] = 32.60$
State 2	$(\mu_2, \sigma_2^2) = (5.03, 1.98)$ $E[T_2] = 19.64$	$(\mu_2, \sigma_2^2) = (3.99, 1.00)$ $E[T_2] = 25.16$
State 3	$(\mu_3, \sigma_3^2) = (4.13, 1.12)$ $E[T_3] = 0.20$	$(\mu_3, \sigma_3^2) = (5.99, 0.99)$ $E[T_3] = 24.92$
State 4	$(\mu_4, \sigma_4^2) = (3.99, 0.97)$ $E[T_4] = 10.24$	$(\mu_4, \sigma_4^2) = (2.01, 1.00)$ $E[T_4] = 17.12$

Table 5.4: Parameter values determined after five iterations of the Baum-Welch algorithm for the second part of the experiment.

ual training data might still not be sufficient to provide us with the best possible reconstruction of the original activities. We take a closer look at this problem in the next section.

5.4 Optimizing initialization

For training HMMs, the Baum-Welch algorithm is used. This algorithm finds a local optimizer for the parameters λ of a HMM to best explain an observation sequence (see section 2.4.3). When λ ends up at a suboptimal local optimizer, this might be reflected in recovered solutions that contain degenerate states or inaccurate state

means, as illustrated in sections 5.2.1 and 5.3.1. This problem can be solved by taking a closer look at the initialization of the HMM parameters, most notably the initialization of the signal distributions within each state. In this section we discuss two ways of initializing HMMs, namely the segmental K-means procedure which is a functionality of HTK, and a novel procedure in which we approach the problem through Potts functionals.

5.4.1 Segmental K-means initialization

The first smart initialization method we discuss is the segmental K-means procedure [43, section 2.3.2] [16]. It works as follows. On the first training cycle, using the provided annotations, the training set is uniformly segmented. If for example a training set consists of ten examples of activities and each activity is described by a HMM with n states, the training set is segmented into $10n$ uniform segments. After segmentation the mean and variance of all data points that fall into a segment is determined through a K-means clustering procedure. Since each segment represents one state, each segment is described by a single cluster. The obtained means and variances are used as initial values of the signal distributions of the states. The hope is that in doing this, enough of the initialized states align with 'actual' states and unfavorable local optimizers are avoided. Note that the segmental K-means procedure does not provide us with a way to determine the optimal number of states from the data; this parameter still needs to be specified 'manually' when applying the K-means approach.

We repeat the experiment with one long and one short activity of the previous section, to test how well the segmental K-means procedure performs. The Baum-Welch algorithm is again iterated until the results do not improve any further. The detailed results of this experiment are shown in table 5.5 and figure 5.6.

	Activity 1	Activity 2
State 1	$(\mu_1, \sigma_1^2) = (6.00, 1.00)$ $E[T_1] = 10.12$	$(\mu_1, \sigma_1^2) = (1.99, 1.00)$ $E[T_1] = 25.88$
State 2	$(\mu_2, \sigma_2^2) = (4.01, 1.00)$ $E[T_2] = 10.08$	$(\mu_2, \sigma_2^2) = (3.99, 1.00)$ $E[T_2] = 25.16$
State 3	$(\mu_3, \sigma_3^2) = (5.99, 1.00)$ $E[T_3] = 10.20$	$(\mu_3, \sigma_3^2) = (5.99, 0.99)$ $E[T_3] = 24.92$
State 4	$(\mu_4, \sigma_4^2) = (4.01, 1.00)$ $E[T_4] = 9.92$	$(\mu_4, \sigma_4^2) = (2.00, 1.00)$ $E[T_4] = 23.80$

Table 5.5: Parameter values determined after two iterations of the Baum-Welch algorithm for the repeated second part of the experiment of the previous section. The HMMs are initialized using the segmental K-means procedure.

The results are now highly satisfactory. The correctness and accuracy are both a perfect 100%. Also, we have obtained these results after only two iterations of the

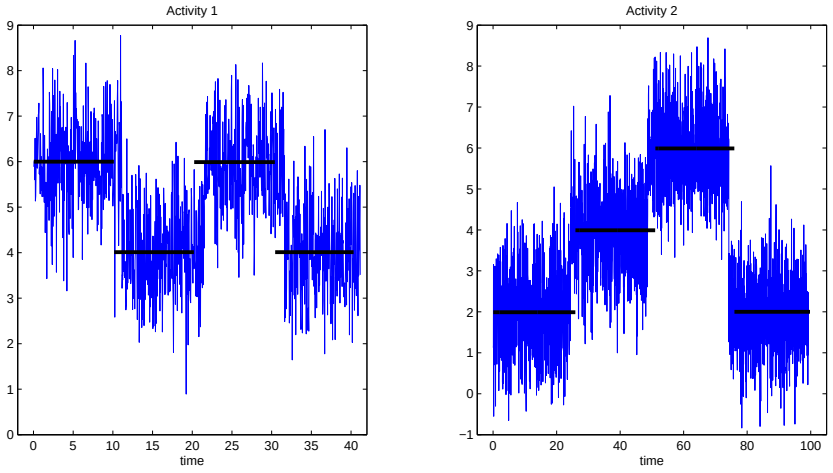


Figure 5.7: Seen here are the means of all four states determined by the Baum-Welch algorithm, for the repeated second part of the experiment of the previous section, but here the HMMs are initialized using the segmental K-means procedure.

Baum-Welch algorithm. On top of this, and in contrast to the previous two sections, we now also have obtained accurate reconstructions of both activities.

5.4.2 Potts problem

Another way to approach the initialization problem is through Potts energy functionals. When describing an activity through a HMM, each datum of the (noisy) measured signal is generated by one of the hidden states of the HMM. The complete signal is described by a sequence of these states, i.e. a piecewise constant signal where each jump discontinuity corresponds to a state transition. In other words, we can interpret determining the most likely state representation of the noisy signal as looking for a piecewise constant function that best describes the original signal. Before continuing, we will make this more precise.

Consider a signal vector $u \in \mathbb{R}^n$. The aim is to reconstruct this original signal u from a noisy measurement vector $f \in \mathbb{R}^n$, which can be described by

$$f = u + \text{noise}.$$

Since we are looking for a solution that is piecewise constant (a different constant for each state in the overall sequence of states), the solution to this problem does not change continuously with the initial conditions. This means that the problem is ill-posed and requires regularization.

Regularization can be achieved by minimizing an energy functional which expresses a tradeoff between a data discrepancy term, which measures the discrepancy between the measured noisy signal f and the original signal u, and a regularization term. We assume the noise to be additive Gaussian noise. This means that

the natural choice for the data discrepancy term is the L^2-norm (see appendix B for an explanation why this is the case). The regularization term contains a-priori information about the solution u, in our case that it should be piecewise constant. Hence, a natural regularization term would be

$$\|\nabla u\|_0 = |\{i : u_i \neq u_{i+1}\}|\,,$$

which counts the number of jumps in the candidate solution u. The full minimization problem, also called a classical Potts problem, is the following:

$$P_\gamma = \gamma \|\nabla u\|_0 + \|u - f\|_2^2 \to \min\,. \tag{5.2}$$

The parameter $\gamma \in \mathbb{R}$ controls how many jump discontinuities the solution will contain, e.g. a larger value of γ will result in fewer jumps.

To solve the Potts functional, we use the Pottslab Matlab toolbox [33–35, 40]. Because of the regularization term, the functional (5.2) is not convex. In this case, often the total variation (TV) penalty $\|\nabla u\|_1 = \sum_i |u_{i+1} - u_i|$ is used instead of the $\|\cdot\|_0$ norm to make the problem convex. The TV problem can be solved using convex optimization and a global minimizer exists and can be calculated. However, as shown in [34], the minimizers of the TV problem are not the same as those of the Potts problem. The minimizers of the Potts problem correspond to genuine piecewise constant solutions, whereas the minimizers to the TV problem do so only approximately.

In the accompanying articles the authors show that for univariate data, as is the case for our problem where we treat each data channel separately, the problem can be solved fast and exactly using dynamic programming. See appendix C for an explanation of the method for calculating this minimizer.

We apply the Potts functional to our problem in the following way. From a training data set, we cut out all examples of a single activity. We resample all these examples to give them equal length to make it possible to calculate a meaningful average. For the resulting calculated 'average' activity, we solve the Potts problem. To determine the best value of the regularization parameter γ, we need to do some manual tweaking to find out which value best suits our type of signal.

The solution of the Potts problem is a piecewise constant, n-dimensional function, where n is the number of data channels under consideration. We can use this result to determine the unknown HMM parameters in the following way. Each constant part of this function represents one of the hidden states. A jump discontinuity corresponds to a state transition, so the solution also provides us with a good estimate of the optimal number of states. The value of each constant part is the mean of the Gaussian signal distribution of the corresponding state. From comparing the 'average' activity to the piecewise constant solution, we can determine an initial guess of the variance of the signal distribution in each state. Finally, the time spent in each state can be used to determine the values of the state transition probabilities.

To test the Potts approach, we solve equation (5.2) for our artificial data experiment of section 5.2.1. The results are shown in figure 5.8. The results are exactly

what we are looking for. Not only are all three states recovered perfectly, the solution is also robust for a large range of the regularization parameter γ. We also try

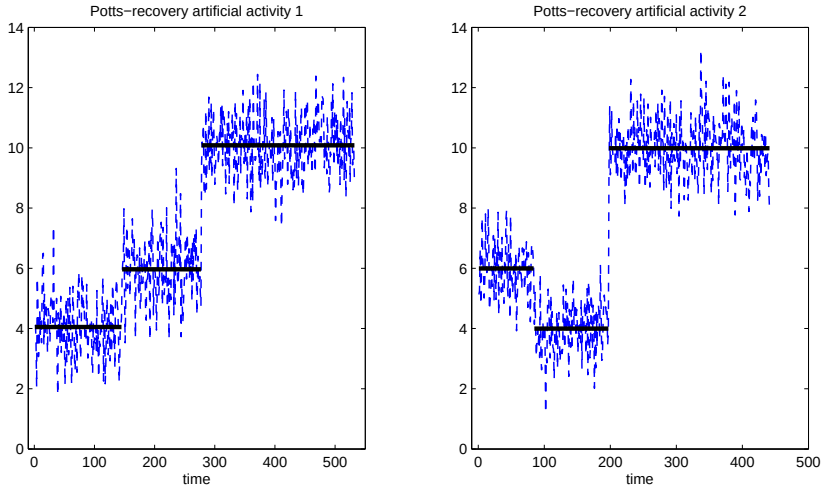


Figure 5.8: Seen here are the recoveries of the three states of both artificial activities through the Potts approach. The three states are recovered perfectly for $\gamma \in [0.07, 2.23]$.

the Potts approach for the artificial data experiment of section 5.3.1. The results are shown in figure 5.9.

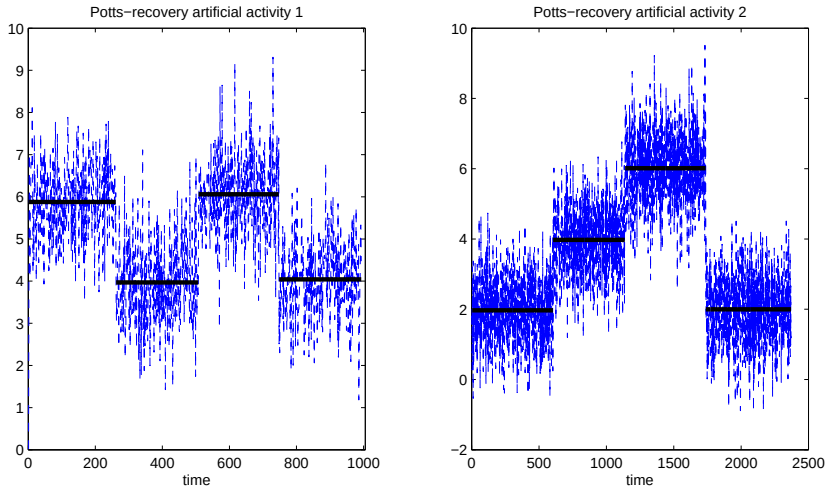


Figure 5.9: Seen here are the recoveries of the four states of both artificial activities of the experiment of section 5.3.1 obtained through the Potts approach. The three states are recovered perfectly for $\gamma \in [0.09, 3.70]$.

We can now also test the Potts approach on a real life example. We consider the lifting activities data set. See figure 5.10 for a plot of the solution of the Potts problem for the z-accelerometer data channel of a 'squat' activity. From this example we see that our original assumption that the hidden Markov states should correspond to stationary parts within the data does not hold. The state representation of an activity is actually more like a piecewise constant representation of the original signal. The specific locations of these states and the order in which they are visited is what differentiates one activity from another.

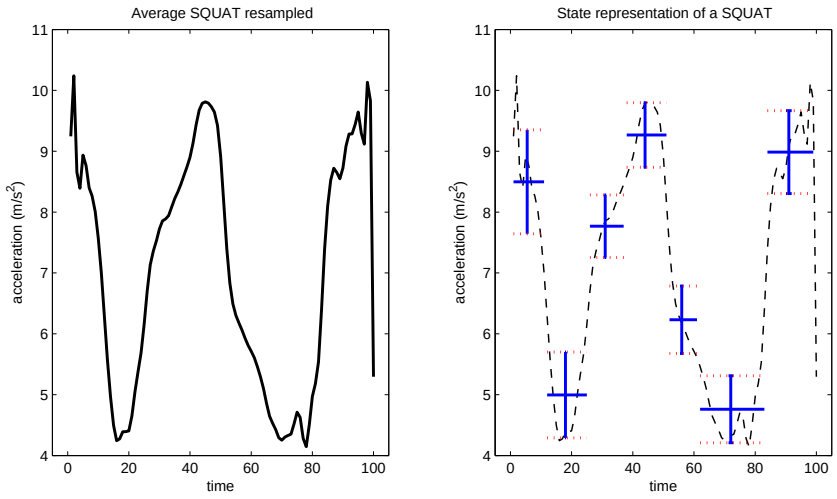


Figure 5.10: Seen on the left is the resampled mean of the z-accelerometer data channel of a 'squat' activity. On the right the same resampled mean is shown, but now the states found by solving the Potts problem ($\gamma = 0.3$) are included. Also shown are the standard deviations around the means of each state.

Summarizing, the solution obtained from the Potts problem approach essentially solves the complete problem of how to initialize a HMM. It provides us with a way to find good initial estimates of all unknown HMM parameters.

Chapter 6

Handling empty data

6.1 Classification doubt

As described in section 3.4.3, FusionAAC can make three types of mistakes: substitution, deletion (false negatives) and insertion errors (false positives). It seems reasonable to assume that when FusionAAC makes a mistake, this is somehow reflected in the computed likelihood associated with the recognized activity.

Being able to recognize classification errors becomes even more relevant when considering data with empty patches where no activities are being performed. Typical data obtained from monitoring subjects will always contain these empty patches. For example, not everything a subject does might be of interest and hence will not be included in the dictionary. FusionAAC is designed to always find the activity in the dictionary that is most likely to have generated a given data sequence and so it will always come up with a classified activity. Since there is nothing to recognize within empty data patches, this will result in a lot of false positives, which we consider to be the type of error that is most important to avoid (see also section 3.4.3).

Ideally, in addition to the total probability for each recognized activity (see section 3.4.2), we would also like to know the probabilities associated with the second best choices. Unfortunately, because of the nature of the problem and the way the software handles the data, the second best answer at any given moment is not so clearly defined. For example, when comparing the two most likely sequences of activities in a data set, start and end times of activities will probably never exactly coincide with each other. Even worse, the number of recognized activities within an observation sequence might not even be equal.

In this chapter we investigate how to approach the problem of handling empty data. We consider the data set consisting of a series of lifting activities, as decribed in section 3.2. The data set is divided into a training set and a testing set. We remove all 'walking', 'standing' and 'sitting' activities from the training set. Since FusionAAC is then no longer trained to recognize these three activities, we have artificially created empty patches of data within the testing set.

Having FusionAAC handle this data will lead to a high number of false posi-

tives. We introduce two strategies to improve the way FusionAAC processes empty data. The first strategy aims to remove false positives through postprocessing of the recognition results. The second strategy aims to avoid false positives by trying to classify the empty patches in the data. These strategies are described in the next sections.

6.2 Removing false positives through postprocessing

The only information FusionAAC provides us with, is the total log probability for each recognized activity. We will try to improve on the results of the above experiment by taking a closer look at these total log probabilities. The HVite recognition tool provides output as shown in table 6.1.

t_{start}	t_{end}	Recognized activity	Total log probability
xxx	xxx	activity i	$\log P(\text{activity } i)$

Table 6.1: Output of the HVite recognition tool. For each recognized activity, a start and end time is provided. The last column shows the total log probability associated with the recognized activity

The first thing to investigate is whether it is possible to observe variation in the total log probability. Naturally, larger values should mean a higher degree of certainty of the classification. The order of magnitude of the calculated log probabilities is $\sim -10^{2\text{-}3}$.

We can now start comparing the results of the classification process with the true list of activities within the test set. We look for a threshold value. Any value below the threshold means that the total probability of a recognized activity is too low and there is reason to doubt the result and the classification is discarded, a value above the threshold means that the classification is reliable. We will test this approach on the data set containing artificial empty patches as described at the beginning of this chapter. The results of this experiment are shown in section 7.5.1.

6.3 Classifying empty data patches

The second strategy consists of introducing a dummy HMM. Consider a dictionary consisting of a number of specific activities. Secondly, consider a data set sparsely populated by the activities from this dictionary, so we again have a data set that contains empty patches. Theoretically, if the activities are different enough from the noise in between them, it might be possible to let the dummy HMM account for the noisy patches. This way, all empty patches will be classified as noise and insertion errors are avoided.

We will test this approach on the data set containing artificial empty patches as described at the beginning of this chapter. However, we do not expect great re-

sults from this experiment. Firstly, the removed activities ('walking', 'standing' and 'sitting') are somewhat similar to the activities we are trying to classify. Secondly, the removed activities are also quite different from each other, which makes it hard to describe them using a single HMM. Unfortunately we do not yet have access to more suitable data to test this strategy, but we expect to be able to do so in the future. The results of this experiment are shown in section 7.5.2.

Chapter 7

Results

7.1 Overview of this chapter

This chapter contains the recognition results of various experiments performed on a real-life data set. This data set consists of a series of lifting activities, which are described in section 3.2. The results are presented mainly in the form of confusion matrices. In a confusion matrix, all activities under consideration are listed in both the rows and the columns of the matrix. The rows correspond to the original classes of activities (denoted as 'ori') in the data set, the columns correspond to the predicted classes of the activities (denoted as 'pred'). Each correct prediction is located on the diagonal of the matrix (denoted in gray), each off-diagonal element is a Substitution error S. This presentation of the results makes it easy to see whether a certain activity is systematically confused with another. In the confusion matrices, D and I mean Deletion error and Insertion error respectively (see section 3.4.3)

For each of the experiments in this chapter, the $n \times (n+1)$ initial transition probability matrix $\mathbf{A}$, where n is the number of states of the HMM, is of the form $a_{i,i} = 0.8$ and $a_{i,i+1} = 0.2$:

$$\mathbf{A} = \begin{pmatrix} 0.8 & 0.2 & & \\ & \ddots & \ddots & \\ & & 0.8 & 0.2 \end{pmatrix}.$$

Some preliminary testing showed that it is not necessary to be more specific when initializing the transition probability matrix. The only thing that is important is to specify which state transitions are allowed and which are not. Secondly, as also described in section 3.3.1, for each experiment in this section the initial state distribution is chosen to be $\boldsymbol{\pi} = \{1, 0, \ldots, 0\}$. Any further relevant parameter value initializations are specified in the coming subsections.

Section 7.2 contains the results of testing the classifier on a subset of the lifting activities data set. To make it easier on the classifier, only the 'pure' lifting activities are considered, namely lifting an object by stooping (4) or squatting (5), lifting an

object positioned to the right (6) or to the left (7), and putting the object down again (8). The omitted activities, namely standing still (1), walking (2) and sitting (3) vary more in duration and hence are more difficult to classify. Section 7.3 contains the results of testing the classifier on the complete lifting activities data set, and hence presents a greater challenge to the classifier.

In all the experiments of this chapter eight data channels are used, namely x- and z-accelerometer and y- and z-gyroscope data (see also section 3.2.1), recorded by sensors positioned on the sternum and the spine. These eight channels should be sufficient for FusionAAC to be able to differentiate between the above activities.

Section 7.4 contains the results of applying PCA (see chapter 4) to the data set before running FusionAAC. Here we are mainly interested to see whether it is possible to (a) reduce the number of input data channels and (b) improve on the results of sections 7.2 and 7.3.

Section 7.5 contains the results of applying the postprocessing techniques (which are described in chapter 6) to a data set containing empty patches.

7.2 Recognition of 'pure' lifting activities

This section contains the results of testing the classifier on a subset of the lifting activities data set which is described in section 3.2. The main goal of this section is comparing three different methods of initialization of the HMMs, namely flat start initialization, segmental K-means initialization and initialization through the Potts approach, all of which are described in section 5.4.

For the flat start initialization, the number of hidden Markov states for all HMMs are chosen to be equal. The same holds for the segmental K-means initialization. In both these experiments, the number of states is increased until the recognition results do not improve any further. For each value of the number of states, the Baum-Welch algorithm is iterated until the recognition results do not improve any further.

For the Potts initialization, not only the number of states varies per activity, but also the initial signal distribution varies per state. For each individual activity, the means for the Gaussian distributions within each state are determined in the following way. We solve the Potts problem for each separate input data channel. Typically, the number of states obtained from the Potts solution varies per data channel. We choose the data channel that gives us the maximum number of states to be leading. To define the means per state for the parallel data channels, for each location of the leading states the average of the Potts solution in the other data channels is calculated . This way each data channel is divided into an equal number of states. The value of the regularization parameter γ is chosen to be equal to 0.1. For this value, the number of states found for each activity roughly corresponds to the optimal number of states found for the flat start initialization. The number of states for each activity with this approach is roughly stable for $\gamma \in [0.05, 0.15]$.

Preliminary testing showed that the value of the variance for the signal distri-

bution does not seem to have a significant influence on the results, so we opt to initialize the variance in each state with the variance of the complete data set. This gives the HMMs some additional freedom to converge to the optimal parameter configuration. The Baum-Welch algorithm is again iterated until the recognition results do not improve any further.

The training set consists of 17 'lifting from the left', 18 'lifting from the right', 18 'stoops', 18 'squats' and 71 'putting object down' activities. The (independent) testing set consists of 8 'lifting from the left', 8 'lifting from the right', 8 'stoops', 8 'squats' and 34 'putting object down' activities.

Flat start initialization

In table 7.1, the results are shown for the optimal initial parameter configuration for a flat start initialization.

ori\pred	Down	Left	Right	Squat	Stoop	D	# originals
Down	34	0	0	0	1	1	34
Left	0	8	0	0	0	0	8
Right	0	0	8	0	0	0	8
Squat	2	0	0	6	0	0	8
Stoop	0	0	0	0	8	0	8
I	0	0	0	0	0	Total:	0

Table 7.1: Recognition results after a flat start initialization on the 'pure' lifting data set, obtained for $n = 8$ states and four iterations of the Baum-Welch algorithm.

Segmental K-means initialization

In table 7.2, the results are shown for the optimal initial parameter configuration for a segmental K-means initialization.

ori\pred	Down	Left	Right	Squat	Stoop	D	# originals
Down	31	0	0	0	1	2	34
Left	0	8	0	0	0	0	8
Right	0	0	8	0	0	0	8
Squat	1	0	0	7	0	0	8
Stoop	0	0	0	0	8	0	8
I	0	0	0	0	0	Total:	0

Table 7.2: Recognition results after segmental K-means initialization on the 'pure' lifting data set, obtained for $n = 13$ states and two iterations of the Baum-Welch algorithm.

Potts initialization

In table 7.3, the results are shown for the optimal initial parameter configuration for a Potts approach initialization.

ori\pred	Down	Left	Right	Squat	Stoop	D	# originals
Down	28	1	0	1	2	2	34
Left	0	8	0	0	0	0	8
Right	0	0	8	0	0	0	8
Squat	1	0	0	6	0	1	8
Stoop	0	0	0	0	8	0	8
I	0	0	0	0	0	Total:	0

Table 7.3: Recognition results after Potts initialization on the 'pure' lifting data set, obtained after five iterations of the Baum-Welch algorithm.

Summary and discussion of the results

Table 7.4 gives an overview of the results of this section. The recognition results

Experiment	**# states**	**Correctness**	**Accuracy**	S	D	I	**# B-W**
Flat start	8	93.9%	93.9%	3	1	0	4
K-means	13	93.9%	93.9%	2	2	0	2
Potts	9-11	87.9%	87.9%	5	3	0	5

Table 7.4: Comparison between the results of the different initialization methods.

are accurate. On a total of 66 activities only a few errors are made by FusionAAC. Surprisingly, smart initialization of the HMMs does not seem to improve the results, but we remark that there is not much room to improve upon the flat start results. The Potts approach performs slightly worse than the flat start and K-means initializations. The only improvement we gain from applying the segmental K-means procedure is that it reduces the number of Baum-Welch iterations.

For all three experiments, the most difficult activity to classify is 'putting an object down'. The errors made are mainly substitution errors where either an activity is confused with the 'down' activity or vice versa. This error could easily be avoided by introducing a language model. A language model can be compared to grammar; where certain types of words can only be (or mostly) followed by certain other types of words, the same holds for activities. For example, for the lifting activities data set, it is not possible to pick up two objects consecutively without putting the former object down first. Providing this kind of structure to the dictionary of the classifier will likely correct most of the errors. Language models are beyond the scope of this research, but they would not be difficult to implement.

For the above experiments, we only considered data recorded by sensors positioned on the spine and on the sternum. Considering the accuracy of the results and the type of errors made, we do not expect significant improvements of the results by considering different or extended sensor configurations.

7.3 Recognition of lifting activities: complete data set

This section, together with appendix D.1, contains the results of testing the classifier on the complete lifting activities data set which is described in section 3.2. The main goal of this section is again comparing three different methods of initialization of the HMMs, namely flat start initialization, segmental K-means initialization and initialization through the Potts approach, all of which are described in section 5.4. The approach for each initialization method is the same as in the previous section.

The training set consists of 25 'lifting from the left', 26 'lifting from the right', 26 'stoops', 26 'squats', 105 'putting object down', 13 'sitting', 94 'standing still' and 124 'walking' activities. The (independent) testing set consists of 9 'lifting from the left', 7 'lifting from the right', 9 'stoops', 8 'squats', 33 'putting object down', 4 'sitting', 15 'standing still' and 37 'walking' activities.

In tables D.1, D.2 and D.3 in appendix D.1, the results are shown for the optimal initial parameter configurations for a flat start initialization, the segmental K-means procedure and the Potts approach respectively.

Summary and discussion of the results

Table 7.5 gives an overview of the results of this section.

Experiment	**# states**	**Correctness**	**Accuracy**	S	D	I	**# B-W**
Flat start	11	95.9%	91.8%	3	2	5	7
K-means	10	98.4%	95.9%	2	0	3	0
Potts	5-11	91.8%	−%	10	0	514	7

Table 7.5: Comparison between the results of the different initialization methods.

For the flat start initialization, the results are decent. On a total of 122 activities, 10 errors are made. In contrast to the classification experiment on the 'pure' lifting activities data set, the flat start initialization is now clearly outperformed by the segmental K-means procedure. Not only is the amount of classification errors halved by this smart initialization, FusionAAC also requires less iterations of the Baum-Welch algorithm. There do not seem any activities that FusionAAC has specific problems with. We mention again that the introduction of a language model might improve the recognition results even further.

The Potts initialization clearly underperforms. The main issue is the enormous amount of insertion errors, most notably for the 'sitting', 'standing still' and 'walk-

ing' activities. Closer inspection of the results reveals that the problem is with the transition probability matrices. For the three problematic activities the following happens. After iterating the Baum-Welch algorithm a few times, the probabilities of transitioning to the next hidden Markov state become too large. Because of this, the average time spent in the hidden Markov chain decreases. This results in a situation where it becomes more likely for FusionAAC to classify for example the activity 'standing still' as a number of repeated 'standing still' activities. This explains why the number of insertion errors is so large. On top of this, this large number of insertions also makes it more difficult to classify the other activities. Changing the initial transition probability matrix does not remedy the problem.

As of yet we do not fully understand why this problem occurs. There is one notable difference between the segmental K-means approach and the Potts approach. For the segmental K-means initialization the HMMs all have the same number of states ($n = 10$). For the Potts approach, this parameter varies. The HMMs describing the lifting activities are roughly similar, with $n = 9 - 11$ states. The HMMs for the three problematic activities 'sitting', 'standing still' and 'walking' on the other hand are described by only $n = 5, 7, 6$ states respectively. Perhaps the problem is caused by this fact, and the software somehow struggles with the combination of HMMs with a varying number of states.

For the above experiment we again only considered input data recorded by sensors positioned on the spine and sternum. The difference between the activities in the dictionary of this experiment is now greater compared to the differences in the 'pure' lifting activities dictionary. The recognition results however are still highly accurate, and we do not expect significant improvements of the recognition results by considering different or extended sensor configurations.

7.4 Recognition of lifting activities: PCA

We now apply PCA preprocessing to the eight data channels of the extended lifting activities data set. We only combine PCA with the segmental K-means initialization since that approach has lead to the best results in the previous section. We conduct this experiment on the same data as was used in the previous section.

In table 7.6 the results obtained after application of PCA are compared with the results of the previous section. All PCA inputs to FusionAAC in these experiments account for more than 90% of the information in the data set. Using the first four principal components as input to FusionAAC, combined with a segmental K-means initialization procedure and 11 states for all HMMs, already results in accurate recognition results that are comparable to the results obtained after using a flat start initialization in the previous section. Using the first five principal components is enough to improve on the results of the flat start initialization from the previous section. Using the first six or seven principal components results in a total of six classification errors, only one more error than the best results obtained in the previous section after application of the segmental K-means procedure.

Experiment	# channels	# states	Corr	Acc	S	D	I	# B-W
Flat start	8	11	95.9%	91.8%	3	2	5	7
K-means	8	10	98.4%	95.9%	2	0	3	0
PCA	4	11	94.3%	90.2%	6	1	5	0
PCA	5	12	96.7%	91.8%	3	1	6	0
PCA	6	12	95.9%	95.1%	4	1	1	0
PCA	7	10	96.7%	95.1%	4	0	2	0

Table 7.6: Comparison between the recognition results with and without application of PCA.

7.5 Avoiding and removing false positives

In this section we give an overview of the results of testing our strategies for removing and avoiding false positives as described in chapter 6. Section 7.5.1 contains the results after applying postprocessing, where we look for variation in the calculated total log probabilities. Section 7.5.2 contains the results after applying the second method, where we introduced a dummy HMM to account for all empty parts in the data set. We compare the results for both a flat start and a segmental K-means initialization. We omit the Potts approach, because the previous section showed that as of yet it cannot compete with the K-means smart initialization.

The number of states is again chosen to be equal for all HMMs, and increased until the recognition results do not improve any further. For each value of the number of states, the Baum-Welch algorithm is iterated until the results do not improve any further.

For testing the first method, the training set consists of 25 'lifting from the left', 26 'lifting from the right', 26 'stoops', 26 'squats' and 105 'putting object down' activities. The (independent) testing set consists of 9 'lifting from the left', 7 'lifting from the right', 9 'stoops', 8 'squats' and 33 'putting object down' activities. The testing set further contains 56 unlabeled empty patches of data. For testing the second method, we randomly add 231 'junk' activities to the training set and 56 'junk' activities to the testing set.

7.5.1 Removing false positives through postprocessing

In table 7.7 the results are shown of testing the classifier on a data set containing empty patches. In table 7.8 the results are shown after postprocessing. We have used a threshold value of -1000: if the total log probability is below this threshold, the classification result is discarded.

In table 7.9 the results before and after postprocessing are compared. Postprocessing results in a significant decrease of the false positives, but it is not possible to remove all of these errors. Also, postprocessing results in a significant amount of false negatives: all but one of the 'lifting from the right' activities are now deleted.

ori\pred	Down	Left	Right	Squat	Stoop	D	# originals
Down	32	0	0	1	0	0	33
Left	0	9	0	0	0	0	9
Right	0	0	7	0	0	0	7
Squat	0	0	0	8	0	1	8
Stoop	0	0	0	0	9	0	9
I	8	6	31	12	0	Total:	57

Table 7.7: Results of the experiment after segmental K-means initialization on data containing empty patches, obtained for $n = 6$ states after six iterations of the Baum-Welch algorithm. We notice that the actual activities of interest are classified almost perfectly, but the results are worthless because the amount of false positives is so high.

ori\pred	Down	Left	Right	Squat	Stoop	D	# originals
Down	30	0	0	1	0	2	33
Left	0	9	0	0	0	0	9
Right	0	0	1	0	0	6	7
Squat	0	0	0	8	0	0	8
Stoop	0	0	0	0	9	0	9
I	6	2	3	3	0	Total:	14

Table 7.8: Results of the experiment after postprocessing of the results.

Experiment	**# states**	**Corr**	**Acc**	S	D	I
Before	6	98.5%	12.1%	1	0	57
After	6	87.9%	66.7%	1	8	14

Table 7.9: Comparison between the results before and after postprocessing.

7.5.2 Classifying empty data patches

In table 7.10, the results are shown for the optimal initial parameter configuration when we introduce a dummy HMM (named 'Junk') to classify all empty patches within the data set. The results are obtained after a flat start initialization. In table 7.11, the results are shown for the optimal initial parameter configuration when we introduce a dummy HMM (named 'Junk') to classify all empty patches within the data set. The results are obtained after a segmental K-means initialization.

Summary and discussion of the results

Table 7.12 gives an overview of the results of this section. The recognition results

ori\pred	Down	Left	Right	Squat	Stoop	Junk	D	# originals
Down	28	0	0	1	0	0	4	33
Left	0	6	0	1	1	0	1	9
Right	1	1	2	0	3	0	0	7
Squat	1	0	0	3	4	0	0	8
Stoop	0	0	0	0	9	0	0	9
Junk	0	3	1	1	6	40	5	56
I	5	5	0	1	11	3	Total:	25

Table 7.10: Recognition results after introducing a 'junk' HMM, obtained after a flat start initialization for $n = 15$ states after 7 Baum-Welch iterations.

ori\pred	Down	Left	Right	Squat	Stoop	Junk	D	# originals
Down	32	1	0	0	0	0	0	33
Left	0	9	0	0	0	0	0	9
Right	0	0	7	0	0	0	0	7
Squat	0	0	0	8	0	0	0	8
Stoop	0	0	0	0	9	0	0	9
Junk	0	0	0	0	0	52	4	56
I	2	0	0	0	0	0	Total:	2

Table 7.11: Recognition results after introducing a 'junk' HMM, obtained after segmental K-means initialization for $n = 12$ states after 1 Baum-Welch iteration.

Experiment	**# states**	**Correctness**	**Accuracy**	S	D	I	**# B-W**
Flat start	15	72.1%	51.6%	24	10	25	7
K-means	10	95.9%	94.3%	1	4	2	1

Table 7.12: Comparison between the results of the different initialization methods when introducing a dummy HMM.

obtained from applying a flat start initialization are poor, as was expected. The recognition results obtained from applying the segmental K-means initialization on the other hand, are surprisingly good. While the original activities 'walking', 'standing still' and 'sitting' are quite different from each other, trying to describe them all through one 'junk' HMM does lead to highly accurate recognition results. FusionAAC not only succeeds in classifying the activities of interest, but also in classifying the 'empty' patches in the data.

Closer inspection of the segmental K-means initialization results reveals that it is possible to improve them even further. The results contain four deletion errors, but only 'junk' activities have been deleted. Since we are not interested in the empty patches anyway, it does not matter that FusionAAC deletes them.

Chapter 8

Conclusion

8.1 Overview of this research

In this research, the aim has been to improve the FusionAAC software, which uses HMMs for the classification of human activities. We have focused on three main areas of improvement.

Firstly, in chapter 4 we have have looked at preprocessing of the data through application of PCA, to maximize the amount of information and minimize the amount of noise in the input data set. Secondly, in chapter 5 we have looked in depth at the training process of the HMMs and how to improve various aspects of it. Lastly, in chapter 6 we have looked at ways to both remove and avoid false positives in the recognition results.

8.2 Improvements to training process

The first aspect of the training process we considered is choosing a suitable number of hidden Markov states for each HMM. We have motivated the importance of this parameter through an experiment on an artificial data set in section 5.2. We have also tested two ways of varying the number of states for HMMs describing activities in a real life data set. Firstly, we initialized all HMMs with the same number of states and iteratively increased this number until the recognition results did not increase any further. This method is described in section 5.2.2. Secondly, we applied a novel procedure which uses Potts functionals to find a good initial estimate of the number of states to describe each activity individually. For a description of this method, see section 5.4.2.

The results of the first method are promising. The recognition results in both section 7.2 and section 7.3 show that initializing all HMMs with the same number of states can lead to highly accurate recognition results. The results of the second method are mixed. Although our theoretical results suggest that individualizing the number of states per activity should improve the recognition results, we have not yet succeeded in generating equally accurate recognition results from varying

the number of states as we obtained for a fixed number of states. The main reason for this is that our implementation of the Potts approach does not yet seem to be functioning as it should. While our theoretical results in section 5.4.2 show highly accurate recoveries of the underlying artificial activities, trying to recover real life activities using the Potts approach is still problematic, as the results in section 7.3 show.

The second aspect of the training process we considered is the way the training data should be organized before presenting it to FusionAAC to train with. Our theoretical results from 5.3 suggest that sorting part of the training set to individualize the training per HMM could help achieve more accurate recoveries of activities. We have not fully implemented this improvement and hence as of yet have been unable to test it on real life data.

The third and final aspect of the training process we considered is the initialization of the HMM parameters, most notably the initial signal distributions within the hidden Markov states. In section 5.4 we have introduced two smart initialization methods, namely the segmental K-means procedure and the Potts approach, and we have motivated the importance of smart initialization through testing these two methods on an artificial data set.

We have compared the recognition results achieved by smart initialization on a real life data set with the old flat start initialization method employed by FusionAAC. The results of these tests (shown in sections 7.2 and 7.3) further support our claims of the importance of smart initialization. When testing FusionAAC on a pure lifting activities data set, the flat start initialization and segmental K-means procedure perform equally well, but when testing FusionAAC on the more difficult complete lifting activities data set the segmental K-means procedure clearly outperforms the original flat-start. Not only have the recognition results improved, but smart initialization also reduces the number of necessary iterations of the Baum-Welch algorithm, which makes FusionAAC more computationally efficient.

The second smart initialization through application of the Potts approach does not lead to an improvement of the recognition results. As also discussed in section 7.3 and at the beginning of this section, the implementation of this method still needs some work.

Combining all the results and conclusions of this section, we can now make a good suggestion of what the optimal procedure is to train a set of HMMs. Firstly, a good estimate of the optimal number of states for the HMMs can be achieved by employing the incremental strategy as described in section 5.2.2. Secondly, part of the training data should be individualized per HMM. Lastly, application of the segmental K-means initialization method provides FusionAAC with an improved way to initialize the signal distributions for all hidden Markov states.

8.3 Improvements through preprocessing

The goal of applying PCA to the original input data, is to try and reduce the number of data channels needed as input to FusionAAC, while at the same time the recognition results do not deteriorate. The results in section 7.4 show that it is possible to reduce the number of input data channels. For example, using only the first six principal components only results in one additional classification error when compared to the recognition results obtained using all eight original data channels as input. We conclude that PCA can be applied as a way to reduce the dimensionality of the input data and could hence help make FusionAAC more computationally efficient.

PCA did not result in an improvement of the recognition results. We conclude that the original eight data channels hence do not contain noise that is harmful to FusionAAC.

8.4 Improvements through postprocessing

As we have motivated in section 3.4.3, avoiding false positives in the recognition results is highly important. We have suggested two ways to avoid false positives and hence improve the recognition results. The first method is through postprocessing (see chapter 6) of the results. The results in section 7.5.1 from testing this approach show that is it possible to remove false positives from the recognition results by considering the output classification probabilities that FusionAAC provides us with. Although it does not seem feasible to expect to be able to remove all classification errors from the results, the number of false positives can be greatly reduced.

We have also looked to improve our other results through this postprocessing method, but for the other experiments in this report it was not possible to remove any classification errors. Perhaps the reason for this lies in the fact that most of the other results already are quite accurate and there is not much room for improvement in the first place.

Contrary to our expectations, the second method to avoid false positives shows great promise (see section 7.5.2). With a flat start initialization the method performs poorly, but the results obtained after a segmental K-means initialization are highly accurate. With this method to avoid false positives we have shown that it is possible to model empty patches within a data set by adding an extra HMM to the dictionary.

The reason we think that this method works so well, is that FusionAAC is highly capable of classifying the activities we are interested in (see the accurate results of sections 7.2 and 7.3). This makes it easier for FusionAAC to simultaneously classify everything we are not interested in.

8.5 Computational efficiency

Two of the improvements described in this chapter could help improve the computational efficiency of FusionAAC. Firstly we have applied PCA to the input data channels. In our experiments we only considered eight channels, but for other classification problems we might have to work with a data set where it is not so clear how to select the parts of the data that best describe the underlying structure. PCA can help to significantly reduce the dimensionality of the input data and hence possibly also reduce the computation time for FusionAAC.

Secondly, the results from section 7.3 show that applying the smart segmental K-means initialization method can greatly reduce the number of Baum-Welch iterations. Again, for our experiments in this report, we only considered eight data channels, but when this number increases applying the segmental K-means procedure might help reduce the computation time for FusionAAC.

8.6 Future recommendations

As mentioned most notably in section 7.2, the introduction of a language model might help improve the recognition results. Language models are beyond the scope of this research, but for example for the lifting activities dictionary, the concept is not hard to imagine. We suggest that it should be one of the first areas of future research.

In the above section on improving the training process, we have mentioned individualizing the training process by training each HMM separately. We have not yet implemented this improvement, but we have motivated how this might improve the recognition results and hence think it is also an interesting direction for future work.

The last suggestion for future research is on how to choose the data that serves as input to FusionAAC. In the experiments in this report we only considered eight data channels that we thought would be sufficient to capture all important elements of the activities we are trying to recognize. Another option would be to consider a significantly larger sensor configuration. We can let PCA determine the key features of this data set. Choosing the first n principal components might serve as highly informative input to FusionAAC while at the same time the computational demand on the software would not become too great.

Bibliography

[1] HTK 3.4. `http://htk.eng.cam.ac.uk/`. [Online; accessed 29 June 2015].

[2] Fusion 3D. `http://www.ambulab.com/index.php/fusion`. [Online; accessed 29 June 2015].

[3] Faisal Bashir, Wei Qu, Ashfaq Khokhar, and Dan Schonfeld. Hmm-based motion recognition system using segmented pca. In *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, volume 3, pages III–1288. IEEE, 2005.

[4] M. Bicego. *Hidden Markov Models for Pattern Recognition and Computer Vision: methodological issues and applications*. PhD thesis, University of Verona, March 2003.

[5] Elaine A. Biddiss and Tom T. Chau. Upper limb prosthesis use and abandonment: a survey of the last 25 years. *Prosthetics and orthotics international*, 31(3):236–257, 2007.

[6] Christopher M. Bishop et al. *Pattern recognition and machine learning*, volume 4. Springer Verlag, New York, 2006.

[7] Horst Bunke, Markus Roth, and Ernst Günter Schukat-Talamazzini. Off-line cursive handwriting recognition using hidden markov models. *Pattern recognition*, 28(9):1399–1413, 1995.

[8] Xsens Technologies B.V. `https://www.xsens.com/`. [Online; accessed 29 June 2015].

[9] Sylvain Calinon and Aude Billard. Recognition and reproduction of gestures using a probabilistic framework combining pca, ica and hmm. In *Proceedings of the 22nd international conference on Machine learning*, pages 105–112. ACM, 2005.

[10] O. Cappé, E. Moulines, and T. Rydén. *Inference in Hidden Markov Models*. Springer Verlag, New York, 2005.

[11] Antonin Chambolle. Image segmentation by variational methods: Mumford and shah functional and the discrete approximations. *SIAM Journal on Applied Mathematics*, 55(3):827–863, 1995.

[12] Parent Project Muscular Dystrophy. http://www.parentprojectmd.org/site/PageServer?pagename=Understand_about. [Online; accessed 29 June 2015].

[13] Felix Friedrich, Angela Kempe, Volkmar Liebscher, and Gerhard Winkler. Complexity penalized m-estimation: Fast computation. *Journal of Computational and Graphical Statistics*, 17(1):201–224, 2008.

[14] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[15] Frederick Jelinek, Lalit Bahl, and Robert Mercer. Design of a linguistic statistical decoder for the recognition of continuous speech. *Information Theory, IEEE Transactions on*, 21(3):250–256, 1975.

[16] Biing-Hwang Juang and Lawrence R Rabiner. The segmental k-means algorithm for estimating parameters of hidden markov models. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 38(9):1639–1641, 1990.

[17] Xsens MTw Development Kit. https://www.xsens.com/products/mtw-development-kit/. [Online; accessed 29 June 2015].

[18] Anders Krogh, Michael Brown, I. Saira Mian, Kimmen Sjölander, and David Haussler. Hidden markov models in computational biology: Applications to protein modeling. *Journal of molecular biology*, 235(5):1501–1531, 1994.

[19] D. Mumford and J. Shah. Boundary detection by minimizing functionals. *Image understanding*, pages 19–43, 1988.

[20] Bryan Pardo and William Birmingham. Modeling form for on-line following of musical performances. *Proceedings of the National Conference on Artificial Intelligence*, 20(2):1018–1023, 2005.

[21] K. Person. On lines and planes of closest fit to system of points in space. *Philosophical Magazine*, 2:559–572, 1901.

[22] Symbionics Program. www.flextension.nl/projecten/symbionics/. [Online; accessed 29 June 2015].

[23] Adapt Project. http://www.imdi-sprint.nl/adapt/?lang=en. [Online; accessed 29 June 2015].

[24] Lawrence Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[25] Lawrence Rabiner and Biing-Hwang Juang. An introduction to hidden markov models. *ASSP Magazine, IEEE*, 3(1):4–16, 1986.

[26] Roessingh Research and Development. http://rrd.nl/. [Online; accessed 29 June 2015].

[27] Zahid Riaz, Anf Gilgiti, and Sikander M. Mirza. Face recognition: a review and comparison of hmm, pca, ica and neural networks. In *E-Tech 2004*, pages 41–46. IEEE, 2004.

[28] Sheldon M. Ross. *Introduction to probability models.* Academic press, ninth edition, 2007.

[29] Jonathon Shlens. A tutorial on principal component analysis. *arXiv preprint arXiv:1404.1100*, 2014.

[30] Smason79. https://en.wikipedia.org/wiki/Mixture_distribution#/media/File:Gaussian-mixture-example.svg. [Online; accessed 30 October 2015].

[31] A.C. Stange. Data optimization techniques and their influence on activity classification. Master's thesis, University of Twente, June 2010.

[32] Thad Starner, Joshua Weaver, and Alex Pentland. Real-time american sign language recognition using desk and wearable computer based video. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(12):1371–1375, 1998.

[33] Martin Storath and Andreas Weinmann. Fast partitioning of vector-valued images. *SIAM Journal on Imaging Sciences*, 7(3):1826–1852, 2014.

[34] Martin Storath, Andreas Weinmann, and Laurent Demaret. Jump-sparse and sparse recovery using potts functionals. *Signal Processing, IEEE Transactions on*, 62(14):3654–3666, 2014.

[35] Martin Storath, Andreas Weinmann, Laurent Demaret, Vasileios Angelopoulos, Jürgen Frikel, and Kilian Hohm. http://pottslab.de/. [Online; accessed 18 January 2016].

[36] Universiteit Twente. https://www.utwente.nl/nieuws/!/2016/1/468156/ut-ontwikkelt-robotarm-voor-duchenne-patienten. [Online; accessed 28 January 2016].

[37] Alessandro Vinciarelli and Samy Bengio. Offline cursive word recognition using continuous density hidden markov models trained with pca or ica features. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 3, pages 81–84. IEEE, 2002.

[38] R.G.V. Wassink. Automated activity monitoring of human activities. Master's thesis, University of Twente, August 2003.

[39] R.G.V. Wassink, C.T.M. Baten, and P.H. Veltink. Classifying human lifting activities automatically by applying hidden markov modeling technology. *Journal of Biomechanics*, 40:S428, 2007.

[40] Andreas Weinmann, Martin Storath, and Laurent Demaret. The l^1-potts functional for robust jump-sparse reconstruction. *SIAM Journal on Numerical Analysis*, 53(1):644–673, 2015.

[41] Wing Wong and Mark Stamp. Hunting for metamorphic engines. *Journal in Computer Virology*, 2(3):211–229, 2006.

[42] Stephen John Young, N.H. Russell, and J.H.S. Thornton. *Token passing: a simple conceptual model for connected speech recognition systems.* Cambridge University Engineering Department Cambridge, UK, 1989.

[43] Steve Young, Gunnar Evermann, Mark Gales, Thomas Hain, Dan Kershaw, Xunying Liu, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, Valtcho Valtchev, and Phil Woodland. *The HTK book (for HTK Version 3.4).* Cambridge University Engineering Department, UK, 2009.

[44] M.J.E. de Zeeuw. Ambulatory arm activity classification in subjects with duchenne muscular dystrophy. Master's thesis, University of Twente, February 2013.

Appendix A

Equivalence mixture distribution versus single Gaussian

In this section we show that for a HMM λ with J hidden states, for which the observable signals in each hidden state have a (multivariate) Gaussian mixture distribution

$$b_j(\mathbf{x}) = \sum_{m=1}^{M_j} c_{jm} \mathcal{N}[\mathbf{x}|\boldsymbol{\mu}_{jm}, \mathbf{U}_{jm}],$$

there exists another equivalent HMM λ' with $J' = \sum_{j=1}^{J} M_j$ hidden states (one for each mixture component), for which the observable signal distribution in each hidden state consists of only a single Gaussian

$$b'_{j'}(\mathbf{x}) = \mathcal{N}[\mathbf{x}|\boldsymbol{\mu}_{j'}, \mathbf{U}_{j'}].$$

Equivalence is meant here in a probabilistic sense: if a HMM λ is equivalent to another HMM λ', this means that for a given observation sequence $\mathcal{O}$ it holds that $P(\mathcal{O}|\lambda) = P(\mathcal{O}|\lambda')$. First we show how λ' is related to λ. Secondly, we will prove equivalence. For this proof, extensive use was made of [4, §3.3]

A.1 Constructing the equivalent HMM

Each state j of λ is split into M_j states, one for each of the M_j mixtures. This results in a total of $J' = \sum_{j=1}^{J} M_j$ states for λ'. The state space $\mathcal{S}'$ of λ' can be represented in the following way:

$$\mathcal{S}' = \{q'_1, q'_2, \ldots, q'_{J'}\} = \{q'_{11}, \ldots, q'_{1M_1}, q'_{21}, \ldots, q'_{2M_2}, \ldots, q'_{J1}, \ldots, q'_{JM_J}\}$$

where q'_{jm} is a new state of λ' corresponding with the m-th Gaussian of the mixture in the original state j of λ. The signal distribution in this state is given by

$$b'_{jm}(\mathbf{x}) = \mathcal{N}[\mathbf{x}|\boldsymbol{\mu}_{jm}, \mathbf{U}_{jm}]. \tag{A.1}$$

The new transition probability matrix is given by

$$A'_{ik,jm} = P\left(X_{t+1} = q'_{jm} | X_t = q'_{ik}\right). \tag{A.2}$$

This matrix can be simplified to

$$A'_{ik,jm} = A_{ij} c_{jm}, \tag{A.3}$$

where c_{jm} is the weight coefficient of mixture m in the original state q_j. Note that, as required,

$$\sum_{jm} A'_{ik,jm} = \sum_{j=1}^{J} \sum_{m=1}^{M_j} A'_{ik,jm} = \sum_{j=1}^{J} A_{ij} \sum_{m=1}^{M_j} c_{jm} = \sum_{j=1}^{J} A_{ij} = 1.$$

The new initial state distribution is given by

$$\pi'_{jm} = c_{jm} \pi_j. \tag{A.4}$$

A.2 Proof of equivalence

To compute the quantities $P(\mathcal{O}|\lambda)$ and $P(\mathcal{O}|\lambda')$ we use the forward variable, as defined in 2.4.1,

$$\alpha_t(j) = P\left(\mathcal{O}^{1:t}, X_t = q_j | \lambda\right), \tag{A.5}$$

and the recursive relations (see also 2.4.1)

$$\begin{cases} \alpha_1(j) & = \pi_j b_j(O_1), \\ \alpha_{t+1}(j) & = \left[\sum_{i=1}^{J} \alpha_t(i) A_{ij}\right] b_j(O_{t+1}), \end{cases} \tag{A.6}$$

which give us

$$P\left(\mathcal{O}|\lambda\right) = \sum_{j=1}^{J} \alpha_T(j). \tag{A.7}$$

Similarly,

$$P\left(\mathcal{O}|\lambda'\right) = \sum_{jm} \alpha_T(jm) = \sum_{j=1}^{J} \sum_{m=1}^{M_j} \alpha_T(jm) \tag{A.8}$$

$$= \sum_{j=1}^{J} \alpha'_T(j) \tag{A.9}$$

where

$$\alpha'_T(j) = \sum_{m=1}^{M_j} \alpha_T(jm). \tag{A.10}$$

So, if we can show that $\alpha_T(j) = \alpha'_T(j)$ the proof that $P(\mathcal{O}|\lambda) = P(\mathcal{O}|\lambda')$ is complete. We will attempt to do this through mathematical induction on the length T of the observation sequence $\mathcal{O}$.

- **Base step:** $T = 1$

From (A.6) we know that

$$\alpha_1(j) = \pi_j b_j(O_1) = \pi_j \sum_{m=1}^{M_j} c_{jm} \mathcal{N}[O_1|\boldsymbol{\mu}_{jm}, \mathbf{U}_{jm}].$$

Secondly, using the recursive relations (A.6) and the definition of the new initial state distribution (A.4)

$$\begin{aligned}
\alpha'_1(j) &= \sum_{m=1}^{M_j} \alpha_1(jm) = \sum_{m=1}^{M_j} \pi'_{jm} \mathcal{N}[O_1|\boldsymbol{\mu}_{jm}, \mathbf{U}_{jm}] \\
&= \sum_{m=1}^{M_j} c_{jm} \pi_j \mathcal{N}[O_1|\boldsymbol{\mu}_{jm}, \mathbf{U}_{jm}] = \pi_j \sum_{m=1}^{M_j} c_{jm} \mathcal{N}[O_1|\boldsymbol{\mu}_{jm}, \mathbf{U}_{jm}] \\
&= \alpha_1(j)
\end{aligned}$$

and we are done.

- **Induction step:** $\alpha_T(j) = \alpha'_T(j) \Rightarrow \alpha_{T+1}(j) = \alpha'_{T+1}(j)$

First, again using recursive relations (A.6)

$$\begin{aligned}
\alpha_{T+1}(j) &= \left[\sum_{i=1}^{J} \alpha_T(i) A_{ij} \right] b_j(O_{T+1}) \\
&= \left[\sum_{i=1}^{J} \alpha_T(i) A_{ij} \right] \sum_{m=1}^{M_j} c_{jm} \mathcal{N}[O_{T+1}|\boldsymbol{\mu}_{jm}, \mathbf{U}_{jm}].
\end{aligned} \tag{A.11}$$

Secondly,

$$\begin{aligned}
\alpha'_{T+1}(j) &= \sum_{m=1}^{M_j} \alpha_{T+1}(jm) \\
&= \sum_{m=1}^{M_j} \left[\sum_{il} \alpha_T(il) A'_{il,jm} \right] b_{jm}(O_{T+1}) \\
&= \sum_{m=1}^{M_j} \left[\sum_{i=1}^{J} \sum_{l=1}^{M_l} \alpha_T(il) A'_{il,jm} \right] b_{jm}(O_{T+1}) \\
&= \sum_{m=1}^{M_j} \left[\sum_{i=1}^{J} \sum_{l=1}^{M_l} \alpha_T(il) A_{ij} c_{jm} \right] \mathcal{N}[O_{T+1}|\boldsymbol{\mu}_{jm}, \mathbf{U}_{jm}] \\
&= \sum_{m=1}^{M_j} c_{jm} \mathcal{N}[O_{T+1}|\boldsymbol{\mu}_{jm}, \mathbf{U}_{jm}] \sum_{i=1}^{J} A_{ij} \sum_{l=1}^{M_l} \alpha_T(il) \\
&= \left[\sum_{m=1}^{M_j} c_{jm} \mathcal{N}[O_{T+1}|\boldsymbol{\mu}_{jm}, \mathbf{U}_{jm}] \right] \sum_{i=1}^{J} A_{ij} \alpha'_T(i). \qquad \text{(A.12)}
\end{aligned}$$

Here, the fourth equality uses the definition of the new transition probability matrix (A.3) and the last equality makes use of relation (A.10). Comparing equations (A.11) and (A.12) and applying the induction hypothesis $\alpha_T(j) = \alpha'_T(j)$, it holds that $\alpha_{T+1}(j) = \alpha'_{T+1}(j)$ which concludes the proof that $P(\mathcal{O}|\lambda) = P(\mathcal{O}|\lambda')$.

Appendix B

Data discrepancy for additive noise

B.1 From noise model to variational formulation

Our strategy to find the proper data discrepancy term $D(f, u)$ in the Potts functional (see equation 5.2) is to use a-priori information about the noise in the recorded data. We assume that the recorded data f^δ contains additive noise δ:

$$f^\delta = u + \delta.$$

Consider both the signal u we are trying to recover and the recorded data f^δ as random variables. We can then make use of the a-posteriori probability given by Bayes' formula as

$$P(u|f^\delta) = \frac{P(f^\delta|u)P(u)}{P(f^\delta)}, \tag{B.1}$$

where $P(u|f^\delta)$ is the probability that some given recorded data f^δ are based on an underlying true signal u, $P(f^\delta|u)$ is the probability that from the true underlying signal u the data f^δ were recorded, and $P(u)$ and $P(f^\delta)$ are the a-priori probabilities of u and f^δ respectively.

Through maximization of this a-posteriori probability, we can find the Maximum A-posteriori Probability (MAP) estimator:

$$\hat{u} = \arg\max_u P(u|f^\delta) = \arg\max_u P(f^\delta|u)P(u), \tag{B.2}$$

where the equality holds because of Bayes' formula in equation (B.1) and $P(f^\delta)$ can be neglected since it does not depend on u. Instead of maximizing $P(u|f^\delta)$, we can also minimize the negative logarithm of this probability:

$$\hat{u} = \arg\min_u \left[-\log\left(P(u|f^\delta) \right) \right] \tag{B.3}$$

$$= \arg\min_u \left[-\log\left(P(f^\delta|u) \right) - \log\left(P(u) \right) \right] \tag{B.4}$$

We have now arrived at a variational model of the form

$$P(u) = D(f, u) + \alpha R(u), \tag{B.5}$$

where the data discrepancy term $D(f, u) = -\log\left(P(f^\delta|u)\right)$ and the regularization term $\alpha R(u) = -\log\left(P(u)\right)$

B.2 Data discrepency for additive Gaussian noise

From now on, we focus on the data discrepancy term. We assume the additive noise δ to have a Gaussian distribution, with $E[\delta] = 0$ and $var(\delta) = \sigma^2_{noise}$. Further, let f^δ_{ij} and u_{ij} be the i-th data point and j-th data channel of the recorded data and underlying signal respectively, and let δ_{ij} be a realization of the noise δ. Then,

$$\begin{aligned} f^\delta_{ij} &= u_{ij} + \delta_{ij} \\ &\Leftrightarrow \delta_{ij} = f^\delta_{ij} - u_{ij}. \end{aligned}$$

Using this, we find the probability that for data channel j the recorded data f^δ_j were generated from the underlying signal u_j:

$$\begin{aligned} P\left(f^\delta_j|u\right) &= \prod_i P\left(\delta_{ij} = f^\delta_{ij} - u_{ij}\right) \\ &= \prod_i \frac{1}{\sigma_{noise}\sqrt{2\pi}} \exp\left(-\frac{\left(f^\delta_{ij} - u_{ij}\right)^2}{2\sigma^2_{noise}}\right) \end{aligned} \tag{B.6}$$

Subsituting equation (B.6) into the variational formulation of equation (B.5), we find

$$\hat{u}_j = \arg\min_{u_j}\left[\sum_i \frac{1}{2}\left(f^\delta_{ij} - u_{ij}\right)^2 + \beta R(u)\right], \tag{B.7}$$

where all relevant constant terms are accounted for in the constant β. Now scaling equation (B.7) with the total number of data points n_j and letting $n_j \to \infty$, we arrive at a continuous limit for the data discrepancy term:

$$\lim_{n_j\to\infty} \frac{1}{n_j}\sum_i \frac{1}{2}\left(f^\delta_{ij} - u_{ij}\right)^2 = \frac{1}{2}\int \left(f^\delta_j - u_j\right)^2 \mathrm{d}t, \tag{B.8}$$

where the integral is over the time duration of the recorded data. Summarizing all the above, we have arrived at an asymptotical variational model:

$$\hat{u}_j = \arg\min_{u_j}\left[\frac{1}{2}\int \left(f^\delta_j - u_j\right)^2 \mathrm{d}t + \beta R(u)\right]. \tag{B.9}$$

We conclude that in case of additive Gaussian noise, a natural choice for the data discrepancy term is the L^2-norm.

Appendix C

Solving the classical Potts problem

The problem we are trying to solve in section 5.4.2 is a classical L^2-Potts problem of the form

$$P_\gamma = \gamma \|\nabla u\|_0 + \|u - f\|_2^2 \to \min . \tag{C.1}$$

For univariate data, this problem can solved fast and exactly using dynamic programming [11, 13, 19, 40].

The basic idea is as follows. Let $f = (f_1, \ldots, f_n)$ be a data vector. A minimizer for the functional C.1 can be computed if the minimizers of the partial data $(f_1), (f_1, f_2), \ldots, (f_1, \ldots, f_{n-1})$ are known. The partial minimizers can be calculated in the following way. Let $u^1, u^2, \ldots, u^{n-1}$ be the respective minimizers for for the partial data. To compute a minimizer for the complete data $(f_1, \ldots, f_{n-1})$, we create a set of n candidates $v^1, \ldots, v^n$, where each candidate has length n and is of the form

$$v^i = (u^{i-1}, \underbrace{\mu_{i:n}, \ldots, \mu_{i:n}}_{\text{length } n-i+1}). \tag{C.2}$$

Here u^0 is an empty vector, and $\mu_{i:n}$ is the mean of the partial data $f_{[i:n]} = (f_i, \ldots, f_n)$. Of these minimizer-candidates v^i, the one with the lowest Potts functional value is a minimizer for data $f_{[1:n]}$. This solution can be calculated in polynomial $\mathcal{O}(n^2)$ time and $\mathcal{O}(n)$ space complexity [13].

Appendix D

Test results

D.1 Recognition of lifting activities: complete data set

ori\pred	Down	Left	Right	Sit	Squat	Stand	Stoop	Walk	D	# originals
Down	30	0	0	0	2	0	0	1	0	33
Left	0	9	0	0	0	0	0	0	0	9
Right	0	0	7	0	0	0	0	0	0	7
Sit	0	0	0	4	0	0	0	0	0	4
Squat	0	0	0	0	8	0	0	0	0	8
Stand	0	0	0	0	0	14	0	0	1	15
Stoop	0	0	0	0	0	0	9	0	0	9
Walk	0	0	0	0	0	0	0	36	1	37
I	0	0	0	0	0	2	2	1	Total:	5

Table D.1: Recognition results after a flat start initialization on the complete lifting data set, obtained for $n = 11$ states after seven Baum-Welch iterations.

ori\pred	Down	Left	Right	Sit	Squat	Stand	Stoop	Walk	D	# originals
Down	32	0	1	0	0	0	0	0	0	33
Left	0	9	0	0	0	0	0	0	0	9
Right	0	0	7	0	0	0	0	0	0	7
Sit	0	0	0	4	0	0	0	0	0	4
Squat	0	0	0	0	8	0	0	0	0	8
Stand	0	0	0	0	0	14	0	1	0	15
Stoop	0	0	0	0	0	0	9	0	0	9
Walk	0	0	0	0	0	0	0	37	0	37
I	0	0	0	0	0	3	0	0	Total:	3

Table D.2: Recognition results after segmental K-means initialization on the complete lifting data set, obtained for $n = 10$ states after 0 Baum-Welch iterations.

ori\pred	Down	Left	Right	Sit	Squat	Stand	Stoop	Walk	D	# originals
Down	30	0	1	0	1	0	0	1	0	33
Left	0	7	0	0	0	2	0	0	0	9
Right	0	0	7	0	0	0	0	0	0	7
Sit	0	1	0	1	0	0	0	2	0	4
Squat	0	0	0	0	6	1	0	1	0	8
Stand	0	0	0	5	0	10	0	0	0	15
Stoop	0	0	0	0	0	2	7	0	0	9
Walk	0	0	0	0	0	0	0	37	0	37
I	9	5	0	24	3	59	2	304	Total:	406

Table D.3: Recognition results after Potts initialization on the complete lifting data set, obtained after seven Baum-Welch iterations. Because of the enormous amount of insertion errors the results are worthless.