Sleep Stage Classification — Learning Bayesian Classification Networks

by

Michael Höhle and
Brian Kristiansen

Master's Thesis June 1999

Aalborg University
Department of Computer Science
Fredrik Bajers Vej 7E • DK 9220 Aalborg Ø • Denmark

AALBORG UNIVERSITY
Department of Computer Science

Fredrik Bajers Vej 7, DK-9220 Aalborg Øst

Title:
Sleep Stage Classification — Learning Bayesian Classification Networks

Project Period:
February 1st – June 16th, 1999

Semester:
Dat6 (Master's Thesis)

Project Group:
C4-214c

Members:
Michael Höhle
Brian Kristiansen

Supervisor:
Professor Finn V. Jensen

Number of Copies: 9

Number of Pages: 149

ABSTRACT:

This thesis covers the construction of a Bayesian network for automatic classification of sleep stages based on measurements of the brain and the eye activity. Two networks, a simple-Bayes and a constrained model are formulated, both classifying each epoch without considering information from previous epochs. The networks are extended by incorporating history information such as the Markov chain property and amount of time asleep. Special focus is given on learning the quantitative structure of the proposed networks. In order to learn time-sliced Bayesian classification networks, a powerful tool which combines database and expert information is developed. The learning methods implemented are gradient descent with likelihood metric, the EM-algorithm, and neural network type training. Evaluating the learned networks for sleep classification reveals the simple-Bayes network extended with time history and trained by neural network type training as the network giving the best results. A noticeably higher agreement with the experts was obtained with our system than by the Nightingale sleep analysis system.

AALBORG UNIVERSITET
Institut for Datalogi

Fredrik Bajers Vej 7, DK-9220 Aalborg Øst

Titel:
Søvnklassifikation — Learning af Bayesianske klassifikationsnetværk.

Projektperiode:
1. februar – 16. juni, 1999

Semester:
Dat6 (Speciale)

Projektgruppe:
C4-214c

Personer:
Michael Höhle
Brian Kristiansen

Vejleder:
Professor Finn V. Jensen

Antal kopier: 9

Antal sider: 149

SYNOPSIS:

Dette speciale dækker konstruktion af et Bayesiansk netværk til automatisk klassifikation af søvnstadier baseret på målinger af hjerne- og øjenaktivitet. To netværk formuleres, et simple-Bayes og en constrained model, begge netværk klassificerer en epoke uden at tage information fra tidligere epoker i betragtning. Netværkene udvides siden ved at inkorporere information om fortiden som f.eks. Markov kæde egenskaben og sovetid. Speciel opmærksomhed gives læring af den kvantitative struktur for de foreslåede netværk. For at kunne lære time-sliced Bayesianske klassifikationsnetværk, udvikles et værktøj, som kombinerer database- og ekspertinformation. De implementerede metoder er gradient descent med likelihood som metrik, EM-algoritmen og neural network type training. Evaluering af de trænede søvnklassifikationsnetværk afslører, at simple-Bayes modellen udvidet med sovetid og oplært med metoden neural network type training giver de bedste resultater. En mærkbar højere overenstemmelse med eksperterne blev opnået med vores system end med Nightingale søvnanalysesystemet.

Preface

This report describes the project work performed during the spring of 1999 by project group C4-214c, 10th semester (DAT6) at the Department of Computer Science at Aalborg University. The topic of the project is to develop and apply methods — based on the general framework of decision support systems — enabling the construction of a Bayesian network for classification of sleep stages.

The report is mainly addressing the supervisor, the censor, and others with interest in Bayesian network learning or sleep stage classification.

Citations are on the form [⟨author⟩, ⟨year⟩, ⟨page reference⟩], which refers to the bibliography on page 143. Note, the ⟨page reference⟩ field is optional and can instead contain a reference to, e.g., a theorem. Figures, tables, equations, and definitions are numbered as ⟨chapter⟩.⟨number⟩, where ⟨chapter⟩ corresponds to chapter in which it appear, and ⟨number⟩ corresponds the number of appearance within the chapter, and each category having its own counter. Inline definitions are indicated by bold face, a list of them can be found in the index on page 147. Notation with respect to Bayesian networks is taken from [Jensen, 1996].

We would like to thank our sleep expert, associate professor Kim Dremstrup Nielsen, Department of Medical Informatics and Image Analysis, Aalborg University for providing us with sleep data and helping us to acquire knowledge about automatic sleep analysis. Furthermore, we would like to thank Hugin Expert A/S for providing us with a full version of Hugin during the project period.

The orignial version from June 1999 has been edited by the authors to correct missspellings and other minor errors.

Michael Höhle and Brian Kristiansen
Aalborg, December 1999

Contents

Chapter 1 Introduction **1**

Chapter 2 Sleep Analysis **3**

2.1 The Procedure of (Automatic) Sleep Analysis 3

2.2 Nocturnal Polysomnographic Recordings 5

2.2.1 Electroencephalogram (EEG) 5

2.2.2 The Electrooculogram (EOG) and the Actioculogram (AOG) 6

2.2.3 The Electromyogram (EMG) 7

2.3 Sleep Classification (Scoring) 7

2.4 Summary of the Requirements 12

Chapter 3 Feature Extraction and Data Organization **13**

3.1 Extracting Information from the Various Channels 13

3.1.1 EEG 13

3.1.2 AOG 15

3.2 The Data 16

3.2.1 Description of the Data 16

3.2.2 Finding K-Complexes and Sleep Spindels 17

3.2.3 Reducing 2 Second Intervals into 30 Second Intervals . 17

3.2.4 Normalizing the Data 18

3.2.5 Discretizing the Data 18

3.2.6 Organizing the Database 19

3.3 Summary 21

Chapter 4 Modeling the Qualitative Structure **23**

4.1 Single-Epoch Classifier — The Simple-Bayes Model 24

4.2 The Constrained Model 25

4.3 History 29

4.3.1 Templates and Time Slice Bayesian Networks 30
4.3.2 Markov Chain Model 33
4.3.3 Changes of the Probabilities During the Sleep 34
4.4 Type of Person 41
4.5 Summary 43

Chapter 5 Learning and Sampling Algorithms 45

5.1 Learning Bayesian Networks 45
5.2 Learning Bayesian Networks with the EM-Algorithm 46
5.2.1 Maksimum Likelihood Estimation 48
5.2.2 Maximum a Posteriori Estimation 50
5.2.3 Deriving the Parameters of the Dirichlet distribution . 52
5.3 Gradient-Descent Learning of Bayesian Networks 54
5.3.1 The Distance Measure 54
5.3.2 The Likelihood of the Data Cases Measure 59
5.3.3 Damping of the Gradient Step 60
5.3.4 Extending the Gradient-Descent Methods with Generalized Parameters 62
5.4 Learning Neural Network Like Bayesian Networks 63
5.4.1 Calculating the Gradient in a Small Example 64
5.5 Theoretical Considerations 65
5.6 Sampling 67
5.6.1 Logic Sampling 67
5.6.2 Experiments with Logic Sampling 68
5.7 Implementational Dimension 69
5.8 Summary 69

Chapter 6 Experimental Results - The Chest Clinic 71

6.1 The Chest Clinic Classification Problem 71
6.2 Test Setups 72
6.3 Analysis of the Running Behavior 76
6.4 Evaluating the Quality of the End-Results 79

6.4.1 Influence of the Database Size 80
6.4.2 Influence of the Damping Function 82
6.4.3 Influence of Missing Data 83
6.4.4 Influence of the Number of Parameters 84
6.5 Consequences with respect to tsl 84
6.6 Summary 85

Chapter 7 Training and Evaluation of the Sleep Stage Classifier 87
7.1 The Simple-Bayes Model 88
7.1.1 The Setups of the Evaluation for the Simple-Bayes Model 88
7.1.2 The Learning of the Simple-Bayes Model 89
7.1.3 Evaluation of the Simple-Bayes Model 90
7.2 The Constrained Model 95
7.2.1 The Setups of the Evaluation of the Constrained Model 95
7.2.2 The Training and Evaluation of the Constrained Model 97
7.3 The Models Expanded with History Information 98
7.3.1 The Preparations and Setups 99
7.3.2 Training and Evaluation of the History Expansions . . 101
7.4 Summary 103

Chapter 8 Conclusions 107

Appendix A Design and Implementation of the Time Sliced Learner 109
A.1 Flow and Structure of the tsl Program 110
A.1.1 The Flow of tsl 110
A.1.2 The Construction Phase 110
A.1.3 The Execution Phase 111
A.2 The tsl Specification Language 112
A.2.1 The Settings Object 112
A.2.2 The Potentials Object 116
A.3 Components 117
A.3.1 Domain 118

A.3.2 Table Manager 119
A.3.3 Database Connectivity 121
A.3.4 The `Method` component 123
A.3.5 Sampler 125
A.3.6 Evaluator 125
A.4 Running tsl on a Tiny Example 125
A.5 Summary 126

Appendix B Complete Grammar for the tsl Specification Language 127
B.1 The Settings Object 127
B.2 The Potentials Object 128

Appendix C Reference Card for Factor Levels 129

Appendix D Evaluation of the Simple-Bayes Model Using Priors 131

Appendix E Linear Modeling of Signals 139

Bibliography 143

Index 147

Chapter 1

Introduction

We, the authors, tend to argue philosophically about the existence of a Bayesian network construction cycle consisting of construction, analyzing, and adjusting phases. But, we must admit that the networks we have constructed so far are best characterized as toy-examples. Knowledge engineer is what we would like to call us, but our current status can best be described as theorist. The art of crafting Bayesian networks requires a great deal of experience but also theoretic insight in order to choose the best methods to solve an existing task. Our aim with this Master's thesis is to gain experience in crafting networks for a real world problem, the sleep classification task. We managed to establish cooperation with a sleep expert who was interested in examining the applicability of Bayesian networks for classification of sleep stages. This classification, based on measurements of brain, eye, and muscle activities, forms the basis for a graphical representation of a person's sleep behavior for one night. Using artificial intelligence methods to solve the task is not new; neural networks, rule based systems [Nielsen, 1992], and even Bayesian networks [Nielsen *et al.*, 1997] have been used. Currently, our expert performs sleep classification by using a rule-based system called Nightingale. Our goal is to create a system improving the rather simple Bayesian models from [Nielsen *et al.*, 1997] which performs better than Nightingale.

Surveys show that creating the qualitative structure usually is pretty straightforward, [Coupé and van der Gaag, 1998b]. Our focus thus becomes to elicit the quantitative structure of the proposed networks. Sleep analysis is difficult in this regard because the experts, due to the complexity of the brain, only have a limited understanding of the underlying physiological processes resulting in different sleep stages. It is therefore necessary to elicit a great deal of the necessary probabilities by learning from a database containing patient sleep recordings. Especially, we are interesting in learning methods which take into account that the network has to perform classifications, e.g. the method of neural network type training proposed by [Jensen, 1999]. Therefore, a tool (tsl) enabling learning of time sliced Bayesian classification networks, combining database and expert knowledge, has to be implemented. Dictated by the nature of the data we only consider batch learning methods.

To our knowledge, no investigations of neural network type training have been made, therefore as a byproduct we want to evaluate the applicability of this method for learning classification problems. Applicability is investigated in the well-known chest clinic example using artificially generated data and by comparing scorings obtained using the learned sleep classification networks to scorings obtained by Nightingale.

The present thesis is organized as follows. In Chapter 2 and Chapter 3 we present the sleep classification problem at hand and show how the raw measurement data are converted to data that fits into the framework of discrete Bayesian networks. By analyzing the data, an extension to the simple-Bayes network, taking some of the correlation between input features into account, is proposed in Chapter 4. Also extensions with history information, e.g. Markov chain property, are considered in this chapter.

We then in Chapter 5 turn our attention towards the learning methods, including neural network type training, these are described on a theoretical level. Chapter 6 contains an preliminary investigation of the methods for the chest clinic example. This investigation also serves as a test of our implemented general purpose tool for learning time sliced classification networks described in Appendix A.

Finally, in Chapter 7 we use the implemented tool to learn the proposed sleep classification networks and compare their results with the ones obtained by Nightingale.

Chapter 2

Sleep Analysis

Since human beings spends almost one third of their lifetime with sleeping, we may ask ourselves, what is actually happening during sleep? Several physiological features of the body changes during sleep, e.g. blood pressure, heart rhythm, body temperature, respiration, muscle activity, eye activity, and brain activity, [Mortensen *et al.*, 1997]. Of these features we will concentrate on brain activity (EEG), eye activity (AOG), and muscle activity (EMG).

For example, it is known that the periods with deep sleep are important for the production of human growth hormone (HGH), which is needed in the process of growth for children and in the process of renewing the cells in adults, [Nielsen, 1992, p. 1.7]. During sleep about 80% of the total daily amount of HGH is released. In the REM (rapid eye movement) phase the patient is dreaming, which is important for the psychological restoration of the human mechanism. Therefore, both deep sleep and REM sleep are of importance for human beings to function normally both physiologically and psychologically, [Parkes, 1985, p. 29].

The interest in sleep and sleep analysis does not only have a background in the desire for obtaining a better understanding of the sleep process. In todays industrialized world a large number of individuals suffer from sleep disorders, e.g. insomnia, [Nielsen, 1992, p. 2.1]. The diagnosis of sleep disorders and their type and severity are therefore of high importance to the health sector.

2.1 The Procedure of (Automatic) Sleep Analysis

When a patient complains about suffering from a sleep disorder, the doctor can decide to perform a thorough investigation by conducting a sleep analysis. This procedure consists of the following five steps, [Nielsen, 1992]:

Nocturnal polysomnographic recordings: Recording of signals of the brain activity (EEG), the eye activity (EOG), and the muscle activity (EMG). These signals are stored in an appropriate format on an appropriate medium, which today is e.g. the EDF format stored on a cd-rom, [Kemp *et al.*, 1992]. Nocturnal polysomnographic recordings are described in Section 2.2.

Feature extraction: Here, the signals obtained in the previous step are analyzed by extracting usable features. The EEG signal is filtered and then divided into five frequency bands (delta, theta, alpha, sigma, and beta). The EOG and EMG signal is filtered and then discretized binary with respect to given thresholds. Upon these extracted features other features can be derived, e.g. K-complexes and sleep spindles. Feature extraction is described further in Chapter 3.

Classification: In this step the extracted features are used to classify the sleep stage upon the combination of the features; this procedure is also called scoring. Various ways to score the features for each epoch have been proposed. The first systematically but manual method proposed was described in [Rechtschaffen and Kales, 1968] and summarized by us in Section 2.3. Despite the manual's age it is still used worldwide today.

During the last 10 to 15 years automatical methods have come into existence, one of these is the Nightingale system, [Nielsen, 1992], based upon the rules given by [Rechtschaffen and Kales, 1968].

The end-result of the classification step is a hypnogram in which the scorings are plotted against the time of sleep measured in epochs; an epoch is an interval of 20 or 30 seconds.

Diagnosis: Based upon the hypnogram and other outer observation, e.g. snoring and blood pressure, the quality of the sleep can be determined by a doctor who is an expert in sleep disorders. At the moment research for automation of this step, especially with respect to sleep apnea, is being carried out at the Department of Medical Informatics and Image Analysis at Aalborg University. An important element in the diagnosis is the patient's history, the anamnesis, and data from other sources, e.g. day- and night-time function and of course nocturnal polysomnographic recordings, e.g. EEG, EMG, and EOG.

Treatment: When the diagnosis has been made, the treatment, if any, of the patient can begin.

In the classification step described above the following sleep stages can be scored, [Nielsen, 1992]. *Sleep stage W* is the stage in which the subject is awake, and *sleep stage REM* is the stage where most of the dreaming takes place, here the person exhibits *rapid eye movement*. *Sleep stage 1* is the transition stage between wakefulness and light sleep with a normal duration of a few minutes. The sleep in this sleep stage is light where the consciousness is going from drowsiness to real sleep. Real sleep is found in *sleep stage 2* which represents normal sleep, half of the time asleep is spent in this stage. *Sleep stage 3* and *sleep stage 4* defines the deep sleep where stage 4 is the deepest.

Before the sleep can be classified upon extracted features, sleep data must be recorded.

2.2 Nocturnal Polysomnographic Recordings

Nocturnal polysomnographic recordings are normally recordings of the brain activity (EEG — electroencephalogram), eye activity (EOG — electrooculogram), and muscle activity (EMG — electromyogram), [Nielsen, 1992]. Other activities can also be considered, e.g. sound recordings of snoring. Based upon these three activities a sleep classification is possible.

2.2.1 Electroencephalogram (EEG)

The **electroencephalogram (EEG)** is the electrical activity emitted from the brain. The EEG observed on the scalp is the net effect of millions of neurons firing more or less synchronized. The rhythms (e.g. sleep spindles and delta activity — described later) this firing of neurons produces seems to be a pacemaker for other parts of the brain, [Nielsen, 1992, Section 3.1].

The recording of EEG occurs through two surface electrodes mounted according to Figure 2.1, at C_4-A_1 or C_3-A_2. The use of two electrodes pairs is for backup, since only one is needed to obtain valid measurements.

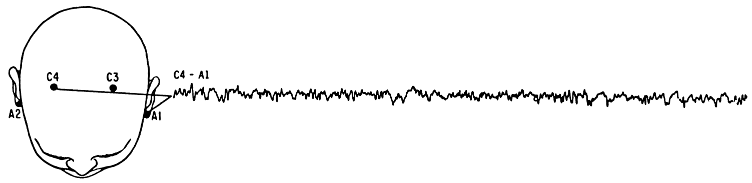

Figure 2.1: Placement of the two electrodes for EEG activity recordings, here C_4-A_1 is used. Note, C_3-A_2 could have been used as well. [Rechtschaffen and Kales, 1968]

The frequency range of interest of EEG for sleep analysis is between 0.5–25 Hz. During the long history of sleep classification, it has become tradition that this frequency range is divided into five frequency bands: delta, theta, alpha, sigma, and beta, [Rechtschaffen and Kales, 1968]. The limits of the five bands used, [Nielsen, 1992, p. 3.5], are depicted in Table 2.1.

Frequency band	Amplitude [μV]	Frequency [Hz]
Delta (δ)	75 – 500	0.5 – 3.5
Theta (θ)	100	3.5 – 8.0
Alpha (α)	75	8.0 – 12.0
Sigma (σ)	50	12.0 – 14.5
Beta (β)	30	14.5 – 25.0

Table 2.1: The partition of the EEG signal in the five different frequency bands, [Nielsen, 1992, p. 3.5].

Besides the EEG patterns, four transient patterns relevant for sleep analysis exist: K-complexes, sleep-spindles, vertex sharp waves, and sawtooth waves. Other types (e.g. paroxysmal patterns, spikes and waves) can also occur, but these patterns are mostly seen in abnormal EEG, e.g. as epilepsy, [Nielsen, 1992]. Only K-complexes and sleep spindles will be considered here.

The characteristics of the five EEG patterns are:

Delta activity: Delta activity is the dominating activity in sleep stage 3 and 4 (deep sleep). Delta activity can also be seen in pathological EEG caused by e.g. brain tumours, [Nielsen, 1992, Chapter 3]. The transient EEG pattern, **K-complexes**, found in sleep stage 2, is defined as EEG wave forms each having a well-delineated negative sharp wave immediately followed by positive, see Figure 2.2. The wave has a duration above 0.5 seconds, thus, it is present in the delta frequency band. Therefore, it is difficult to discriminate between a K-complex and a single period of delta activity in the frequency domain. See [Nielsen, 1992] and [Mortensen *et al.*, 1997] for further descriptions.

Theta activity: Theta activity is present during drowsiness, and therefore, it is used to define sleep stage 1, [Nielsen, 1992, Chapter 3].

Alpha activity: Alpha activity appears when the person is relaxed and has the eyes closed. It is the most investigated activity, as it is the only rhythm clearly present in wakefulness. The rhythm is blocked or attenuated by visual or mental efforts, and the mean rhythm frequency is 10.2 ± 0.9 Hz, [Nielsen, 1992, Chapter 3].

Sigma activity: The sigma activity is determined by the presence of sleep spindles. A **sleep spindle** is defined as a burst-like activity (i.e. an increasing activity followed by a decreasing) in the sigma band having a duration longer than 0.5 seconds. Sleep spindles can be detected using frequency analysis methods. See [Nielsen, 1992] for further descriptions.

Beta activity: The beta activity is not a rhythmic activity like alpha and sigma, it is a noise-like component of the EEG, and it is present in alert wakefulness, light sleep stages (stage 1 and 2), and in REM sleep.

2.2.2 The Electrooculogram (EOG) and the Actioculogram (AOG)

The **electrooculogram (EOG)** and the **actioculogram (AOG)** are used to provide information about the eye movements, i.e. slow eye movements (sleep stage 1) and rapid eye movements (sleep stage REM).

The EOG is an electrical signal recorded with standard electrodes placed like in Figure 2.3. Whereas, AOG is a signal obtained by mechanical movements of a small transducer placed on one of the eyelids, see [Nielsen, 1992, section 3.2] for further information about EOG and AOG. The advantages of using AOG are that AOG only requires one channel, but EOG

|SS|KK|

Figure 2.2: This epoch of an EEG recording illustrates a transition between stage 2 and stage REM. Just before the midpoint of the epoch, there is a clear sleep spindle followed by a K-complex. After the midpoint clear saw-tooth waves (underlined) are present. Taken from [Rechtschaffen and Kales, 1968, Figure 19].

requires two (one for each eye), and furthermore measurements are not disturbed by the signals emitted form the brain since AOG is measured using a transducer.

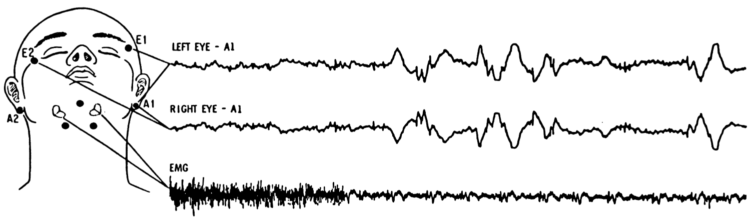

Figure 2.3: Montage of electrodes for 2 channel sleep EOG and for EMG recordings, [Nielsen, 1992, Figures 3.3 and 3.6] and [Rechtschaffen and Kales, 1968, Figure 1]

2.2.3 The Electromyogram (EMG)

The electromyogram (EMG) is the action potentials from at least one motor unit deep below the skin, it can be derived with needle or surface electrodes. EMG is recorded bipolarly, the electrodes are mounted as in Figure 2.3. The EMG varies a lot during the night, if the subject is moving, snoring, or chewing the amplitude is high, whereas if the subject is in restful sleep or awake the amplitude is low; in REM sleep the amplitude is even lower.

2.3 Sleep Classification (Scoring)

A standard for manual sleep description and classification (scoring system for sleep stages of humans), which is used world wide, is given in [Rechtschaffen and Kales, 1968]. In this section we will shortly describe this standard.

Scoring is done epoch-by-epoch, which is chosen to be 20 or 30 second intervals (at that time depending on the paper speed of the polygraph). The epoch-by-epoch approach does not imply that each epoch is considered in isolation, actually, the score assigned to a particular epoch can depend on the preceding or succeeding epoch scorings. If more than one stage is present in an epoch, the score of that epoch should be the stage which is present the most of the epoch. The manual is open for interpretations, meaning that two different sleep experts can score differently, or the same expert can even score the same epoch differently when examining the recordings again.

In the sleep classification manual, [Rechtschaffen and Kales, 1968], human sleep is divided into seven stages: W (wakefulness), MT (movement time), 1, 2, 3, 4, and REM (rapid eye movement), where sleep stages 3 and 4 are deep sleep and stages 1 and 2 are light sleep; these four stages are also called non-REM (NREM) stages. In the following scoring of the sleep stages according to [Rechtschaffen and Kales, 1968] is described.

Sleep stage W (wakefulness): The subject is awake. It is characterized by alpha activity and/or a low amplitude, mixed frequency EEG. However, in 10–20% of the subjects alpha activity is not seen, [Nielsen, 1992, p. 4.3]. In this stage REMs and eye blinks are often present in the EOG (AOG) tracing, and it is often accompanied by a relatively high tonic EMG.

Sleep stage MT (Movement Time): Stage MT is scored if more than half of an epoch is obscured by muscle activity in at least one channel. MT should not be defined from body movements nor movement arousals. **Body movement** is simply when the body moves, but it often has short durations. A **Movement arousal** is defined as any increase in EMG. Both body movements and movement arousals can be detected during all stages. See [Rechtschaffen and Kales, 1968] for further information on this sleep stage.

Sleep stage 1: The subject is in the transition from wakefulness to the other sleeps stages or in the stage following body movements during sleep. It is defined by relatively low amplitude, mixed frequency EEG with a dominance of activity in the 2-7 Hz band. Alpha activity is decreased (in both amplitude and frequency) to less than 50% compared to the activity in stage W. Theta and beta activity is present, and sleep spindles may not appear. Slow eye movements are present, specially in beginning of the stage, while, rapid eye movements are absent. Sleep stage 1 tends to be relatively short, about 1-7 minutes.

Sleep stage 2: This sleep stage is defined by the presence of one or both of the transient EEG patterns, sleep spindles and K-complexes, and the absence of delta activity with high amplitude. This stage contains mixed background activity mainly dominated by sigma and delta activity.

Since K-complexes and sleep spindles are transient phenomena, long periods may pass between their occurrences without the stage being changed. Therefore, if a period between two of these phenomena lasts less than three minutes and an epoch in between would otherwise meet the requirements for stage 1, then it will be score as stage 2 unless indication of movement arousal (stage MT) or considerable increase in EMG exist. But, if the period lasts longer than three minutes, it is scored as stage 1, even if it contains no movement arousal. See [Rechtschaffen and Kales, 1968] for further details.

Sleep stage 3: This stage is defined by an EEG record where at least 20-50% of the delta activity is dominating the epoch. The amplitude must be greater than 75μV. AOG should not indicate eye movements. Sleep spindles could be present in this stage.

Sleep stage 4: This stage is defined by an EEG record where more than 50% of the delta activity with amplitude greater than 75μV is dominating the epoch. Again, sleep spindles could occur during this stage. Note, when the delta activity is considered in the frequency domain, it is difficult to discriminate between a high amplitude of delta activity which results in sleep stage 3 or 4 and a K-complex which results sleep stage 2.

Sleep stage REM (Rapid Eye Movement): This sleep stage is defined by mixed frequency EEG activity with relatively low amplitude, and also episodic REMs appear in the AOG. The EEG pattern is almost the same as in sleep stage 1, but, alpha activity is usually more prominent and the frequency is 1-2 Hz slower, [Rechtschaffen and Kales, 1968].

Both sleep spindles and K-complexes are absolutely absent in this stage. Though, mostly during the first REM period sleep spindles could be present, but EMG remains at the same level. If the span between two spindles is less than half of an epoch, REM is scored, otherwise, stage 2 is scored.

Detecting sleep stage REM is quite difficult due to problems with determining the start and the end of a REM period since the three indicators, EEG, EOG, and EMG, may or may not change simultaneously, [Rechtschaffen and Kales, 1968]. The start of a REM period can for instance be corrected in the following way:

- When the EEG has changed to low amplitude, mixed frequency for at least one epoch before REM is detected by the EOG and furthermore sleep spindles and K-complexes have stopped; then, the epochs from the last spindle or K-complex to the current epoch are scored as REM if the EMG does not change considerably, see Figure 2.4.

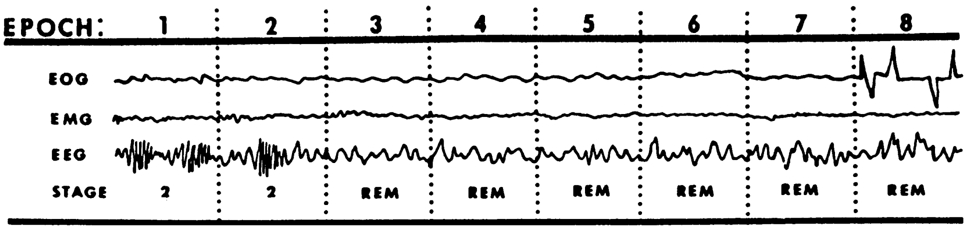

Figure 2.4: Schematic illustration of a rule for scoring the start of stage REM. Taken from [Rechtschaffen and Kales, 1968, Figure 7A].

The end of a REM period can for instance be corrected in the following way:

- When a period of low amplitude, mixed frequency but without eye movements follows after a period with REM; then, if the EMG remains at the same level and if there are no intervening sleep spindles or K-complexes, score as stage REM regardless of the duration, see Figure 2.5.

Other special cases of when to start or end a REM period are discussed further in [Rechtschaffen and Kales, 1968].

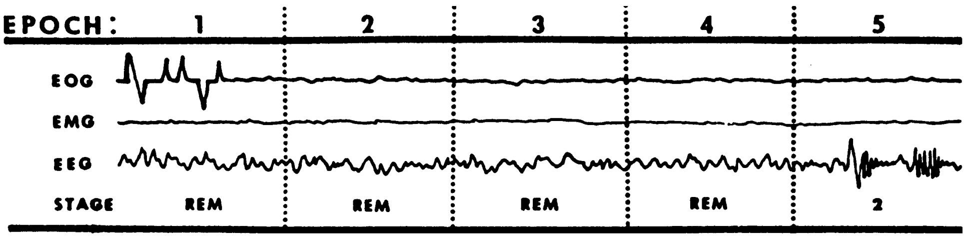

Figure 2.5: Schematic illustration of a rule for scoring the end of stage REM. Taken from [Rechtschaffen and Kales, 1968, Figure 8A].

Table 2.2 gives an schematic overview of this standard classification scheme, [Nielsen, 1992].

Param	Stage W	Stage REM	Stage 1	Stage 2	Stage 3	Stage 4
delta				$\% < 20$	$20 < \% < 50$	$> 50\%$
theta		$\surd$	$\surd$			
alpha	$\surd$		$\% < 50$			
sigma				$\surd$		
beta	$\surd$					
k-complex				$\surd$		
sleep spindle				$\surd$		
EMG	high	low				
REM (AOG)		$\surd$				

Table 2.2: Schematic interpretation of the visual sleep scoring standard. $\surd$ indicates that the parameter is an indicator for the scoring of the stage. The percentages indicate the amount of an epoch that must be dominated by that parameter. Scoring of MT (movement time) is based on EMG only, [Nielsen, 1992, p. 7.1]

Note, since most of the literature available scores sleep stage MT as sleep stage W, we have chosen to do the same in the rest of this thesis. We do this even though [Brink *et al.*, 1994] concludes that a solution to one of the problems in their classifier could probably be to split stage W into wakefulness in start and end and wakefulness during sleep (sleep stage MT).

After defining these six sleep stages, Figure 2.6 shows the distribution of the time spent in each sleep stage.

The end-product of the sleep classification procedure is a hypnogram, an example of a hypnogram is depicted in Figure 2.7 of a normal sleeping young adult subject. The **sleep latency** is defined as the time spent in bed before sleep stage 2 occurs, and a **REM period**

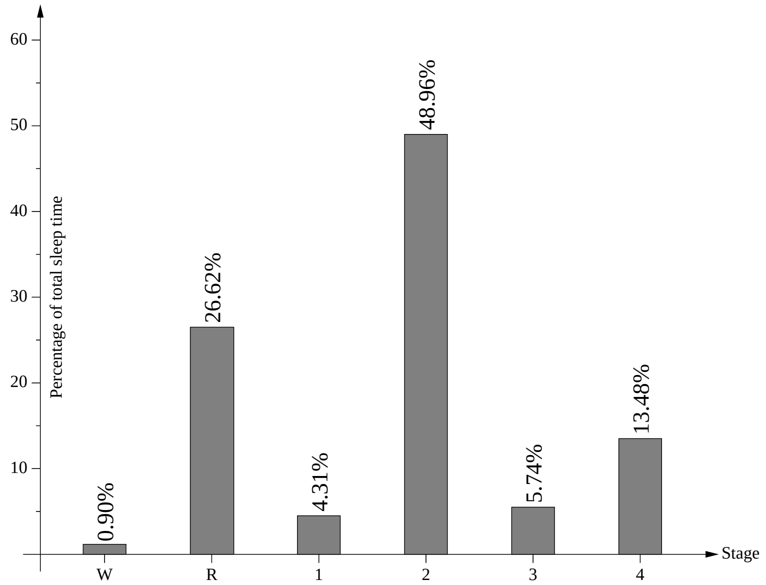

Figure 2.6: The distribution of the sleep stages for subject between 20–29 years, [Nielsen, 1992, p. 1.5]. Originally, the distribution is divided into males and females in [Nielsen, 1992], which we have averaged assuming a population of 50% males and 50% females.

is defined as a non-extensible interval of consecutive REM scorings where small peaks of disturbances are allowed. A **REM sleep latency** is defined as the time between the end of the sleep latency and the start of the first REM period, and a **sleep cycle** is defined from the start of a REM period to the start of the next, [Nielsen, 1992, p. 1.3].

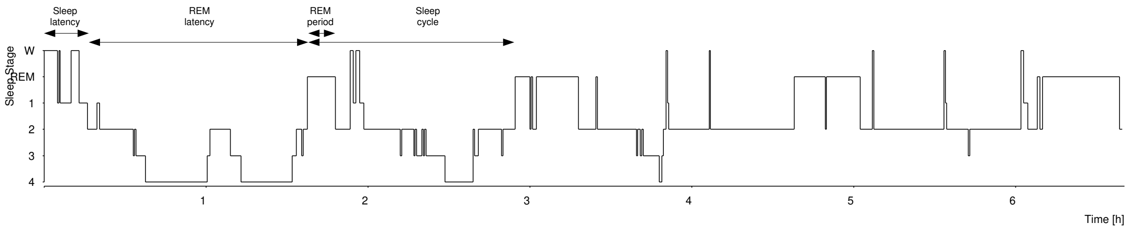

Figure 2.7: An example of a hypnogram — the end-product of the sleep classification task — of a normal young adult (cont03).

A normal sleep pattern consists of a transition from wakefulness into the transition stage, sleep stage 1, which has a normal duration of a few minutes, [Nielsen, 1992, p. 1.3]. Sleep stage 1 is then followed by sleep stage 2, 3, and 4, this REM latency period has a duration of about 70 to 90 minutes, and it is followed by the firsts REM period which can be short or even missing. After this first REM period, the sleep stages 1, 2, 3, and 4 follow again and yet another REM period occurs. This rhythm is repeated throughout the night and it normally has a period length between 70 and 120 minutes with an average of about 90 minutes, [Nielsen, 1992, p. 1.3]. Throughout the night the length of REM periods is increased whereas the length of the periods with deep sleep (sleep stage 4) is decreased.

2.4 Summary of the Requirements

In this thesis we are going to develop the qualitative and the quantitative structure of a Bayesian network sleep classifier. Our goal is to develop a system which can provide the data necessary for plotting hypnograms. This corresponds to the classification step of the sleep analysis procedure described in Section 2.1.

The Bayesian network sleep classifiers proposed in this thesis will all be based on the discretized features extracted from the EEG signal divided into the five frequency bands: delta, theta, alpha, sigma, and beta. The models will also be based on the AOG signal and the K-complexes and sleep spindles derived from the EEG signal. The EMG signal will not be used because we were unable to obtain filtered data, and according to our expert EMG does not contain significant information not also contained in AOG.

The aim of this thesis is to construct a better classifier than the one in the Nightingale system by training the Bayesian network models from expert scorings by using different learning methods. We call this task the **sleep classification problem**. Special focus is given to neural network type training, a method especially suited for classification networks.

Chapter 3

Feature Extraction and Data Organization

In this chapter we shortly describe how to extract the features from the raw nocturnal sleep recordings of the patients. Hereafter, we describe how these extracted data are presented, and how new features can be derived from that. This description is finished with a specification of how the database for automatic sleep classification is built up.

3.1 Extracting Information from the Various Channels

The previous chapter described the features used in the sleep stage classification task. This section deals with how these features can be extracted from the raw 100Hz data signals recorded from each patient. Essentially, this is a signal processing task already implemented in Nightingale from which it is also possible to extract the feature values to a data file. In our opinion a knowledge engineer should not accept such a "black box" generation mechanism of data, but work up a detailed understanding of the mechanisms applied. This section reflects this philosophy by describing how information extraction is performed in Nightingale for the EEG and AOG channels. These extracted values will constitute the foundation on which we later build a Bayesian network for the automatic sleep classification task. Our hope is that knowledge of the extraction methods might enable us to construct better models. This section is mainly based on [Nielsen, 1992].

3.1.1 EEG

EEG is the channel containing the highest information value about the current sleep stage, therefore a comprehensive feature extraction process is performed. Section 2.3 described how classification based on EEG consisted of determining which of the $\delta, \theta, \alpha, \sigma$, and β bands dominated the power spectrum. Finding the dominating band is done within the scope of fixed segmentation using 2 second segments; this has two reasons. First, AR modeling is only valid under the assumption of **locally stationary signals** (i.e. non-time varying, [Diggle, 1992, p. 13]) which is approximately ensured by using not too long segments, see [Nielsen, 1992, p. 6.13] and [Oppenheim and Schafer, 1989]. Secondly, by use of segmentation, each epoch consists of a number of segments which allow easier detection of K-complexes and sleep spindles. Using 2 second segments thus becomes a compromise between using stationary processes and obtaining good spectral resolutions.

Analysis of the various bands is performed in the power spectrum of the signal. A direct calculation of the power spectrum by a Fast Fourier Transformation (FFT) of the raw signal introduces problems such a leakage and a large variance of the spectral estimates, [Oppenheim and Schafer, 1989, Chapter 9] and [Diggle, 1992]. Even application of various filters can not solve these problems. An alternative approach overcoming the shortcomings of FFT is to use parametric analysis in the shape of *linear modeling*.

With linear modeling a locally stationary EEG segment can be described using only a few coefficients, this is computationally cheap and results in more stable spectral estimates than direct FFT calculations. An autoregressive process with order p, denoted AR(p), is a typical linear process used for modeling biomedical signals. With an AR(p) process the EEG signal of a segment is described as a linear combination of past EEG values and a white noise input. Using standard signal processing theory, the coefficients of the linear combination can be extracted to obtain a best fit to the data. Figure 3.1 shows the difference of a power spectrum when using FFT and AR(p), respectively.

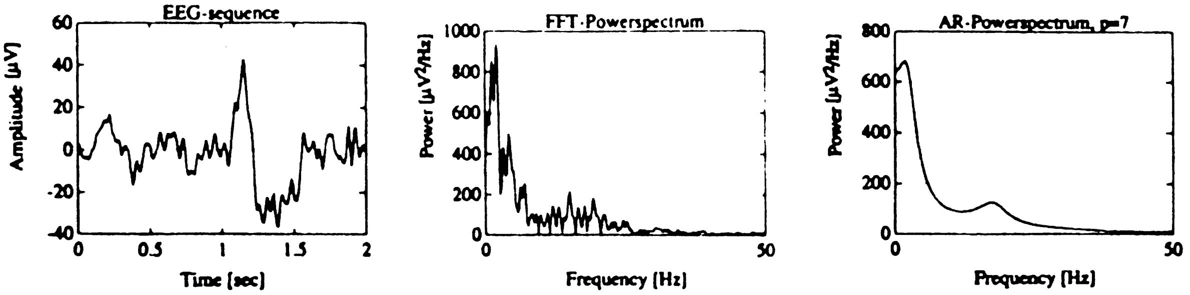

Figure 3.1: Illustration of the power spectrum of a 2 second raw EEG signal transformed to the power spectrum by respectively FFT and AR(7) modeling [Nielsen, 1992, Fig. 6.5]. Note how AR(7) modeling produces a more smooth power spectrum.

Choosing the right order p of the autoregressive process is a subject of constant discussion since it requires a lot of subjective judgment. Too low p oversimplifies the spectrum whereas too high order is algorithmical overkill. We abstain from further discussion and only note that typical orders range from five to twenty and Nightingale uses $p = 7$.

A more detailed description of parametric analysis can be found in Appendix E, it is a result of our quest to understand how the signals really are processed. Unfortunately, the theory proved to be quite complicated without being able to provide us with knowledge on improving the quality of our models.

Pre-emphasizing

Our aim was to establish which band in the power spectrum is dominating the segment by integration of the power spectrum. Investigations of the EEG signal show that the power is not equally distributed over the frequency band. Here, an inverse proportionality between power and band has been observed, with a decade of about 20 dB, see Figure 3.2.

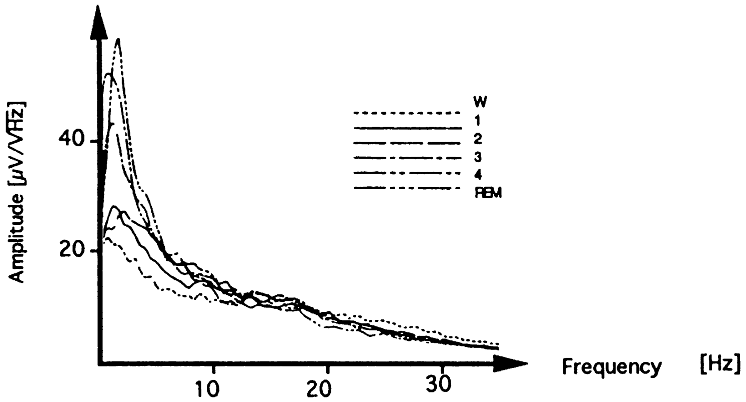

Figure 3.2: Illustration of how the average amplitude is inverse proportional with the frequency [Nielsen, 1992, Fig. 6.6].

This inverse proportionality makes it hard for high frequency as e.g. β to be dominating. By performing an **inverse filtering** of the form

$$f_{pre}(\omega) \quad = \quad f_Y(\omega) \cdot \omega \cdot k, \tag{3.1}$$

the power becomes equally distributed over the frequency bands. Here, k is the **pre-emphasizing slope constant**, in our case $k = 20\,\mathrm{Hz}^{-1}$, and $f_Y(\omega)$ is the expression for the power spectrum obtained by AR(7) modeling, see (E.5).

Band powers

After performing pre-emphasizing of the spectrum, determining the dominating band is simply a question of determining band powers. These can be computed by simple integration of the power spectrum between the lowest and highest frequency of each band, e.g. $\alpha_{lo} = 8\,\mathrm{Hz}$ and $\alpha_{hi} = 12\,\mathrm{Hz}$.

$$P_{\text{EEG-class}} \quad = \quad \frac{\Delta\omega}{hi - lo} \sum_{\omega=lo}^{hi} f(\omega),$$

where $\Delta\omega$ is the frequency resolution — in our case 100 Hz — and $f(\omega)$ is obtained by (3.1). The dominating class is the signal with highest band power.

3.1.2 AOG

Information about the eye movement is only used as an aid to detect REM sleep. For the AOG signal a simple thresholding is enough to detect whether there is eye movement or not. The AOG is segmented into 2 second segments which are bandpass filtered at 2-5 Hz to increase the signal-to-noise ratio and then averaged within the 30 second epoch. The threshold is usually chosen as the average value of the entire night, i.e. the average of all epochs, see Figure 3.3. From the figure the strong correlation between the REM phase and eye activity becomes obvious.

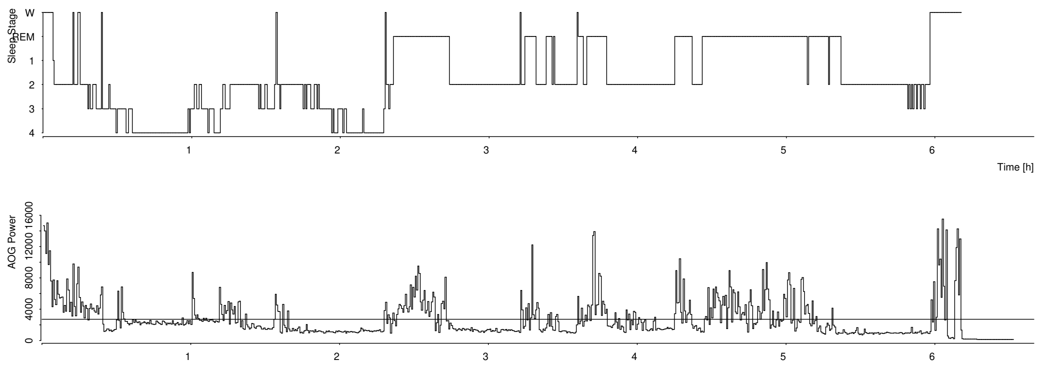

Figure 3.3: Upper plot shows the hypnogram for cont01 based on the scorings from one expert. Lower plot shows the corresponding AOG signal with the average value of the signal plotted as threshold line.

Unfortunately, it was not possible to extract the filtered AOG signal from Nightingale. Thus, we have to extract the raw 100 Hz data for the AOG channel ourselves. Having consulted our expert, no bandpass filtering was performed and the average threshold was computed from the raw signal.

3.2 The Data

In this section we will describe the structure of the data used for sleep staging. First, we will describe the kind of data which was extracted from Nightingale. Next, we will describe how we detect the K-complexes and the sleep-spindles. Afterwards, we describe how to reduce the data intervals from 2 second segments to a 30 second epoch. Finally, we describe how the data is normalized and discretized.

3.2.1 Description of the Data

The data consists of six normal persons with age ranging from 28–38 years, a normal person (also call normal control) does not suffer from any sleep disorder, [Nielsen, 1992]. From each set of raw data the features for EEG have been extracted from Nightingale, therefore, the data for each person contains a list of the following elements.

Epoch number: The sleep is partitioned into 30 second epochs, which are numbered starting with 0. Actually, the sleep data is partitioned into 2 second segments, therefore, it is reduced to 30 second epochs, see Subsection 3.2.3

EEG signals: The power of each of the five frequency bands of the EEG signal: Delta, theta, alpha, sigma, and beta. Also the total power of the EEG signal is included, i.e. the sum of the power the five frequency bands, see the partitioning of the EEG signal in Table 2.1. The values for the power of thee EEG features are absolute continuous values; but since [Nielsen *et al.*, 1997] concludes that relative values yield better results, we normalize the absolute values by total power value. By this procedure the normalized power values will be in the interval $[0, 1]$, see Subsection 3.2.4. To make the normalized data suitable for discrete Bayesian networks, the data is discretized into four or five states as described in Subsection 3.2.5.

AOG values: Values for AOG measurement, has been added to the data sets by manual extraction as dicussed in Section 3.1.2.

KK and SS values: Values for whether a K-complexes (KK) and/or a sleep spindle is present is added to the data sets, see Subsection 3.2.2 for how to find the K-complexes and the sleep spindles.

Scorings: A total of three scorings are provided: manual scorings from two experts working at independent Danish sleep laboratories, and one from the Nightingale program, [Nielsen, 1992].

3.2.2 Finding K-Complexes and Sleep Spindels

K-complexes and sleep sleep spindles are detected in the same way, except that K-complexes are found in the delta frequency band and sleep spindles are found the sigma frequency band. The detection is performed on the continuous unnormalized values of the band power.

To detect a K-complex for a segment, k, the power of the delta frequency band is considered for k and its two neighbours. A complex is detected for the segment if the following boolean expression evaluates to true

$$(T \cdot p_{k-1} < p_k) \wedge (T \cdot p_{k+1} < p_k),$$

where p_k is the δ band power and T is a threshold. The detection of sleep spindles is similar, but happens on the σ band. We choose $T = 3$ based on investigations in the dataset, where we try to find a threshold which maximizes the detection of K-complexes and sleep spindles for stage 2 scorings and minimizes detection of the phenomena for the other stages. Figure 3.4 shows our thresholding detection mechanism for a few epochs.

3.2.3 Reducing 2 Second Intervals into 30 Second Intervals

The data consists of 2 second segments, whereas the expert scorings cover a 30 second epoch. Therefore, the data are reduced to 30 second intervals simply by taking the average of the fifteen 2 second segments. For K-complexes and sleep spindles the 15 boolean values

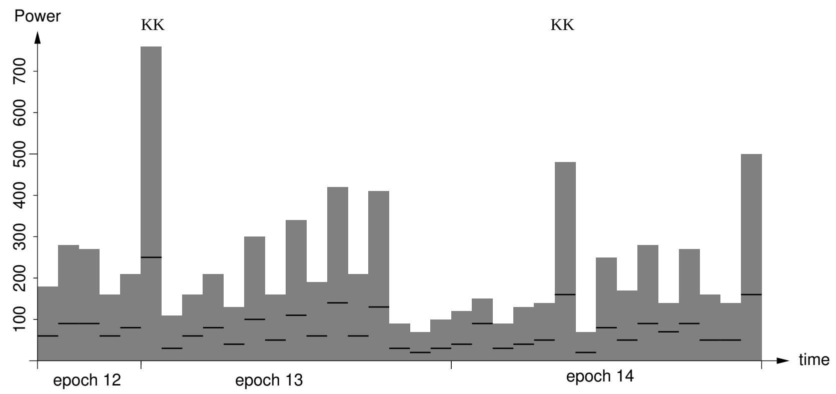

Figure 3.4: Illustration of K-complex detection. The 2 second delta band power values are an extract from the normal control cont01, covering epochs 12 to 14 where only the last third of epoch 12 is depicted. The horizontal black line on each segment shows the value determining existence of a K-complex; if the values of the two neighbors do not exceed this value a K-complex is detected.

representing existence in a segment are combined into one epoch value by using a boolean OR.

3.2.4 Normalizing the Data

Since the five activities (delta, theta, alpha, sigma, and beta) are partitions of the EEG signal, we have chosen to normalize the data such that each value is normalized with respect to the total power (the sum of the five activities). Therefore, the domain of each activity value is $[0, 1]$.

3.2.5 Discretizing the Data

Various ways to discretize the data exists. For instance one could choose to discretize into 5 intervals of equal size, i.e. $\{[0.0, 0.2[, [0.2, 0.4[, [0.4, 0.6[, [0.6, 0.8[, [0.8, 1.0]\}$. This method is rather primitive and naive. Therefore, we will take another approach that takes the distribution of the values within $[0, 1]$ into account, preferable in a way giving the best classification results.

One method to take the distribution of the values into account is to compute the intervals based on the sample mean, $\hat{\mu}$. Here, we determine an interval around the average (estimated mean) by calculating the sample deviation, $\hat{\sigma} = \sqrt{\mathrm{Var}}$, and then letting the interval be $[\hat{\mu} - \hat{\sigma}, \hat{\mu} + \hat{\sigma}]$. But, this interval does not cover the observations good enough (only about 70-75%), therefore, the interval $[\hat{\mu} - 2 \cdot \hat{\sigma}, \hat{\mu} + 2 \cdot \hat{\sigma}]$ seems to be a better choice. These intervals cover about 95% of the observations. The percentage of cover is only valid in the case where the observations conditioned on sleep stage follow a normal distribution. Investigations performed by us show that the data are not really normally distributed. On the other hand the performed investigations are only meant as a guideline for the discretizing, thus we assume that the data are approximately normal distributed. Table 3.1 shows the partition intervals for the delta activity based on scorings from one of the experts (AMD).

Scoring	Mean	Variance	Deviation	Interval	% fit	#obs
W	0.0722	0.0043	0.0659	[0.0063, 0.1381]	87.6%	347
			0.1318	[-.0596, 0.2040]	95.1%	
REM	0.0729	0.0012	0.0352	[0.0377, 0.1082]	85.5%	993
			0.0705	[0.0024, 0.1434]	95.6%	
NREM-1	0.0694	0.0007	0.0265	[0.0429, 0.0958]	72.5%	9
			0.0530	[0.0164, 0.1223]	95.6%	
NREM-2	0.1147	0.0027	0.0516	[0.0631, 0.1663]	71.9%	1795
			0.1032	[0.0115, 0.2179]	95.1%	
NREM-3	0.2010	0.0048	0.0693	[0.1317, 0.2703]	67.5%	381
			0.1387	[0.0623, 0.3397]	96.1%	
NREM-4	0.2597	0.0085	0.0919	[0.1678, 0.3517]	62.0%	631
			0.1838	[0.0759, 0.4436]	98.1%	

Table 3.1: The partition intervals for delta activity based on AMD scorings. For each row the first subrow corresponds to values for $\hat{\sigma}$ and the second subrow for $2\hat{\sigma}$.

This interval partitioning does not provide the discretization directly, therefore, one need to use his or her own common sense, in which the interval partitions can be taken into account. The way we did it is as follows, see Figure 3.5. First, plot all the intervals into a diagram like the one in the figure. Next, if any parts of intervals dominate a sleep stage, make the borders of that subinterval candidates for the separation, consider e.g. sleep stage 3 and 4 in the figure, here stage 4 dominates the interval $[0.3397, 0.4436]$ and both stages dominate together the interval $[0.2179, 0.03397]$. If it is difficult to place a separation, the smaller interval partitionings including fewer of the observations (the thin horizontal lines) can be considered, see e.g. the dotted separation line at 0.1663 in connection with sleep stage 2 in the figure. Finally, choose the most relevant separations; here, we chose a maximum of five intervals for the separation.

Table 3.2 depicts the distribution of the observations into the discrete intervals based on AMD scorings. As expected, sleep stage 3 and 4 dominates when high values of delta are present, and the other stages dominates when low values of delta are present.

Using this approach, all the other frequency bands can be discretized into at most five intervals. The intervals for delta, theta, alpha, sigma, and beta activities that are used in the automatic classifiers, described in Chapter 4, are depicted in Table 3.3.

3.2.6 Organizing the Database

The data presented in this section is collected into a remote Oracle SQL database located on the local area network, each time observed cases are to be fetched we connect to this

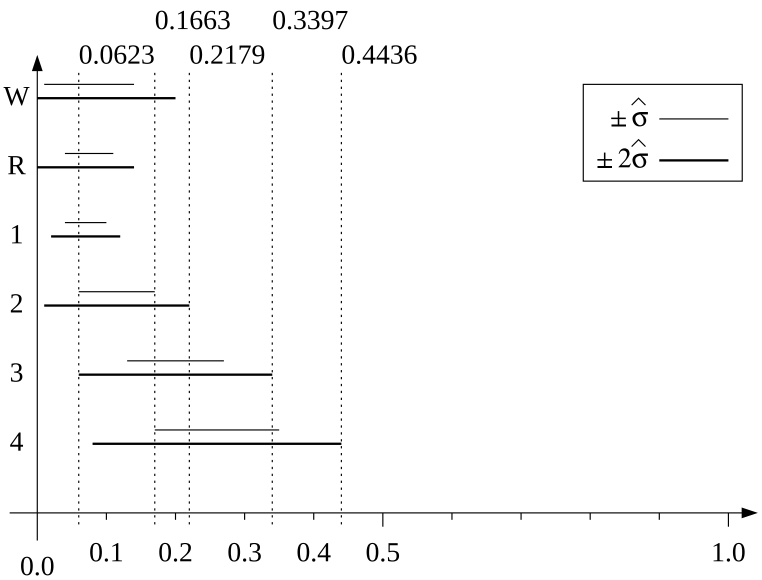

Figure 3.5: Discretizing the EEG signal for partitioning the intervals for delta activity based on AMD scorings.

Interval	Total	W		REM		NREM-1		NREM-2		NREM-3		NREM-4	
	#	#	%	#	%	#	%	#	%	#	%	#	%
[0.00,0.06]	891	200	22.4	445	49.9	33	3.7	213	23.9	0	0.0	0	0.0
[0.06,0.17]	2237	114	5.1	518	23.2	57	2.5	1304	58.3	140	6.3	104	4.6
[0.17,0.22]	500	21	4.2	22	4.4	1	0.2	190	38.0	99	19.8	167	33.4
[0.22,0.34]	423	8	1.9	7	1.7	0	0.0	87	20.6	127	30.8	194	45.9
[0,34,1.00]	187	4	2.1	1	0.5	0	0.0	1	0.5	15	8.0	166	88.8

Table 3.2: The distribution of the δ-band observations into the discrete intervals based on AMD scorings.

Delta	Theta	Alpha	Sigma	Beta
[0.0000,0.0623]	[0.0000,0.1668]	[0.0000,0.1198]	[0.0000,0.0800]	[0.0000,0.1500]
[0.0623,0.1663]	[0.1668,0.2325]	[0.1198,0.2048]	[0.0800,0.1300]	[0.1500,0.2322]
[0.1663,0.2179]	[0.2325,0.2982]	[0.2048,0.2780]	[0.1300,0.1764]	[0.2322,0.3300]
[0.2179,0.3397]	[0.2982,1.0000]	[0.2780,1.0000]	[0.1764,0.2071]	[0.3300,0.5200]
[0,3397,1.0000]			[0.2071,0.1000]	[0.5200,1.0000]

Table 3.3: The five intervals for each of the five frequency bands (activities) used in the automatic classifier.

database. Organization of the data within the database is done such that each subject has his/her own table of data. Before the data sets are loaded into the database all the actions described previously in this section have been performed, e.g. normalization, and discretizing.

Since each set of data contains scorings from two independent experts and one from Nightingale, there exists a column for each of them. For different purposes further columns can be added to each data set, e.g. a column indicating the previous stage for each of the experts. These extra columns will be discussed where needed. From scratch the database consists of the following columns:

`CONT`	The control (subject) number.
`EPOCH`	The epoch number (number of 30 seconds periods in bed).
`DELTA`	The discretized and normalized power of the delta frequency band.
`THETA`	The discretized and normalized power of the theta frequency band.
`ALPHA`	The discretized and normalized power of the alpha frequency band.
`SIGMA`	The discretized and normalized power of the sigma frequency band.
`BETA`	The discretized and normalized power of the beta frequency band.
`POWER`	The total amount of power in the epoch.
`KK`	The K-complex indicator (present/not present).
`SS`	The sleep spindle indicator (present/not present).
`AOG`	The AOG indicator (present/not present).
`AMD`	The scorings produced by the expert *AMD*.
`GOR`	The scorings produced by the expert *GOR*.
`AUT`	The scorings produced by Nightingale.

3.3 Summary

In this section we described how the raw data signals for the EEG and AOG channels are converted such that they can be used as input features for a discrete Bayesian network. Especially the quality of the discretizing intervals is important since all classification will happen on the basis of the discretized values. We chose the intervals by trying to maximize their statement about which sleep stage the band power will result in. The results of the signal processing part we extracted from Nightingale while we implemented our own routines to perform the necessary data manipulation before the sleep data could be imported into the Oracle database as specified. We can now turn our attention to construct the qualitative structure of the sleep classification network.

Chapter 4

Modeling the Qualitative Structure

This chapter covers the construction of the Bayesian networks to be used as sleep stage classifiers. To develop a Bayesian network solving the sleep stage classification task we note that our task is to construct a discrete network which derives the correct sleep stage from the following input: discretized band powers $(\delta, \ldots, \beta)$, presence of K-complexes and sleep spindles, and finally existence of extraordinary AOG activity. Research in [Brink *et al.*, 1994] and [Nielsen *et al.*, 1997] solves the classifying task using different approaches. One possible way is to try to model the rules defined in [Rechtschaffen and Kales, 1968] by a Bayesian network and thus constructing some sort of probabilistic rule based system. In [Brink *et al.*, 1994] the analysis of the EEG signal is done by letting each 2 second segment be represented by a variable containing the five bands as states. Each band variable is instantiated to the dominating band for that segment. To establish the probabilities in the network, manual investigations of the connections between dominating band and sleep stage are performed.

Our work will be based on the ideas described in [Nielsen *et al.*, 1997]. Here, each input feature is represented by a stochastic variable, covering the discretization intervals. Note, we perform the detection of the K-complexes and sleep spindles in a different way than in [Nielsen *et al.*, 1997] as well as we use a different discretization of the band powers for the 30 second epochs.

The desired output, i.e. the current sleep stage, is represented as a stochastic variable as well. By instantiating all the input variables such that they correspond to the measurements of the epoch and then performing a max propagation, the current sleep stage is obtained. This simple template thus has a direct connection between input and output — the question is which way the influence goes, from input to output or the other way around. The conceptual difference between these two modeling approaches is that if all arrows would go into the sleep stage variable, the interpretation would be that sleep stage is some superimposed phenomena, which is caused by the input. Making all input variables parents of sleep stage makes the number of probabilities to be specified intractable. In the reverse situation, sleep stage is considered an underlying physiological phenomena which is only observable to us by its influence on the input features. Our expert prefers this model because he is interested in finding a deeper understanding of the sleep process, thus searching for underlying physiological phenomena.

The development in this chapter is divided into four parts. First, we construct a simple-Bayes network modeling the classification task of one epoch based on measurements from the epoch only. Secondly, this simple-Bayes model is extended such that correlations between some of the variables are taken into account. Thirdly, these single-epoch classifiers will serve as cornerstones for extensions that take information from the past into account by developing a notion of history wrapped around the single-epoch classifier. Fourthly, we finalize the chapter with discussion on how to implement a person type into the models. By applying this divide and conquer strategy our hope is that by optimizing work in each part that the whole is optimized.

4.1 Single-Epoch Classifier — The Simple-Bayes Model

The simple-Bayes[1] model for our classification task is illustrated in Figure 4.1. The model is quite simple since it assumes direct influence between input and output and reflects virtually no knowledge of the underlying physiological processes. Both the expert and we, the knowledge engineers, would like to extend this model by including the regulating role of neurophysiology, neurotransmitters, circadian, ultradian rhythms, etc. The problem is that no agreed upon theory exists of the effect of these physiological activities on sleep, [Mendelson, 1989] and [Peter *et al.*, 1991].

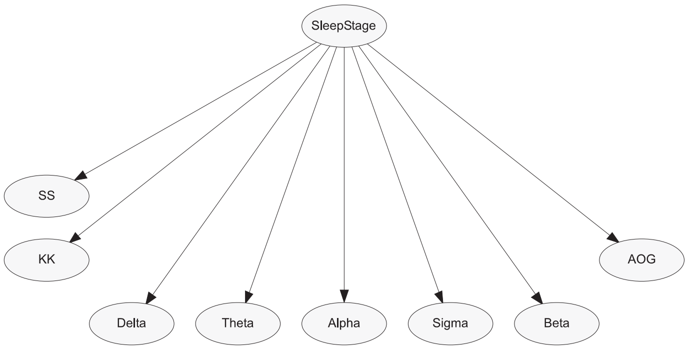

Figure 4.1: Simple-Bayes network reflecting a minimal version of our classification task.

Assessing the probabilities in the simple-Bayes model in Figure 4.1 is done by training the network with the expert scorings in possession. For the four patients in our training set we have scorings from two different experts with an interrater agreement of only 74%. Several schemes now exists — we could just train the network with the scorings from each expert separately, or we can train them with both scoring weighting the experts with some kind of credibility as in Figure 4.2, or we can simply use only scorings of epochs in which the two experts agree. The last approach is not very good because the low agreement between the experts reduces the number of cases, and it will also make history and Markov chain criteria, discussed later, difficult since then we might not have scorings for consecutive epochs. As no difference in the two experts credibility could be established the expert are weighted equally.

[1]simple-Bayes is also known as naïve-Bayes, see [Friedman *et al.*, 1997]

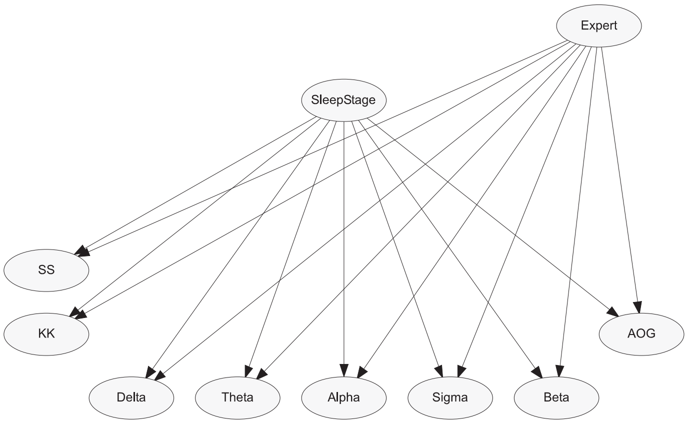

Figure 4.2: Simple-Bayes network extended with an expert node with two states, *gor* and *amd*. The *Expert* node is implemented such that scorings from the two experts are weightend equally, i.e. each probability is averaged over the two experts, e.g. $P(SS|Expert, SleepStage) = \frac{1}{2}\left(P(SS|amd, SleepStage) + P(SS|gor, SleepStage)\right).$

Prudent acquiring of the numbers is the theme of Chapter 5. For now we only consider the qualitative structure.

After construction of the simple-Bayes model, an evaluation of its structural abilities is needed. In Chapter 7 the models overall ability to classify is evaluated. The problem with the simple-Bayes model is that it does not take the correlations between e.g the frequency bands into account, therefore, the correlated (if any) frequency bands carry greater weight than other features. Hence, we will in the next section discuss a constrained model utilizing some of the correlation.

4.2 The Constrained Model

The main weakness of the simple-Bayes model is that once reasoning opposite to the arrows is performed to find the sleep stage, the contributions of the input variables are weighted equally towards the conclusion. This assumption is only reasonable when there is no or only little correlation between the input variables. Assume e.g. that there was some deterministic relationship between the value of δ and θ. Now, say that the values of δ and θ indicate that we are in sleep stage 4, whereas α favors a sleep stage Wake. In the simple-Bayes model the two input variables favoring stage 4 dominate the one variable favoring stage Wake, thus sleep stage 4 is giving a higher probability than deserved since δ and θ are not independent.

An investigation of a hypothesis claiming that little or no correlation between the input variables in our classifier falls naturally into two topics. First, an investigation of the correlation between the individual band powers has to be performed. Secondly, the correlation between the transient phenomena (KK and SS) and the frequency bands from which they are derived has to be investigated.

To gain insight into the correlation between the different frequency band inputs, Figure 4.3 shows pairwise plots of all the non discretized band powers. The plots clearly show high correlation between several bands, e.g. the relationship (δ — α). Appearance of these correlations devalues the applicability of the simple-Bayes model.

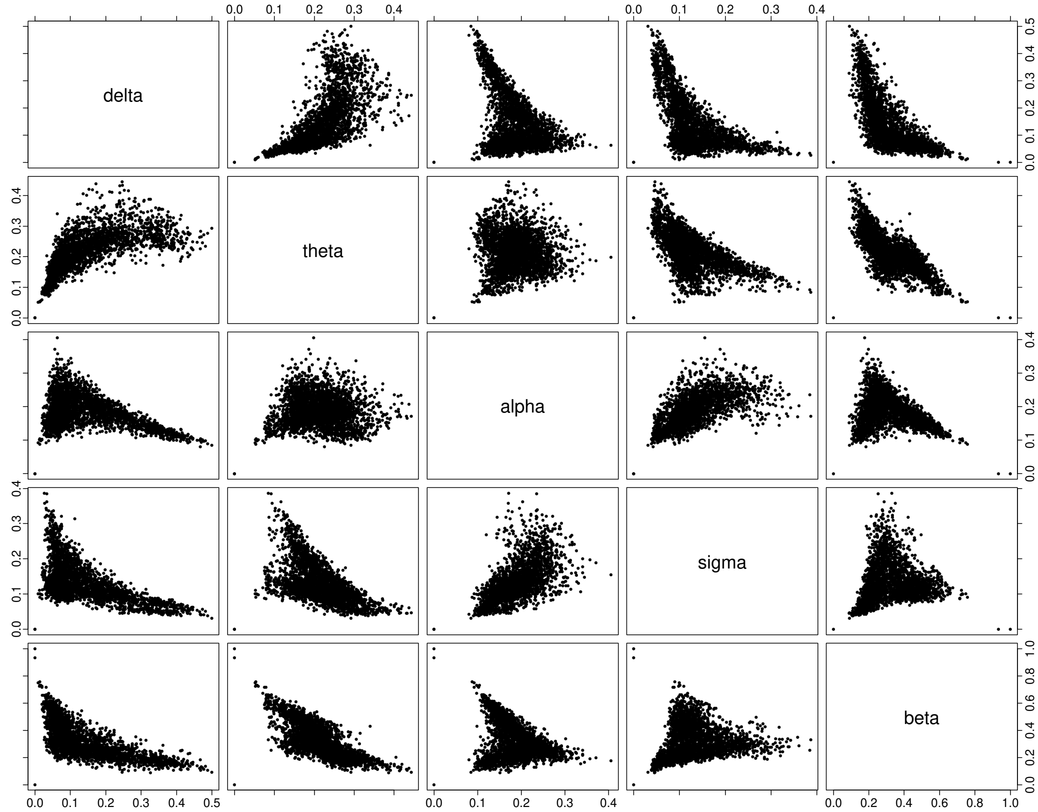

Figure 4.3: Non-discretized values of all pairs of the frequency bands plotted against each other. Data cases for which the expert *AMD* has scored sleep stage 4 in the training data set are used for the plots. The pattern is repeated for the other sleep stages.

The correlation could stem from the existence of hidden variables not included in the model, but since there is no knowledge of the physiological process these hidden variables can not be identified. Instead we are forced to apply a technical procedure and simply try to extend the simple-Bayes model to cope for obvious correlation not reflected by the model.

Based on the plots in Figure 4.3 we have chosen to model correlation between only three frequency bands. Here, the relationships (δ — α), (δ — β), and (α — σ) will be considered. The plots of these indicate an almost linear relationship: When the delta activity is high in (δ — α) and (δ — β), the α and β activities are low, and vice versa. In (α — σ) the situations is the opposite, when α activity is high then the σ activity is also high, and vice versa. Another approach could have been to say that the neighbouring frequency bands are correlated because of the neighbourhood in the power spectrum, but the plots do not entirely support this claim.

Looking at the feature extraction of K-complexes and sleep spindles from the delta and sigma band, respectively, it becomes obvious that there is high connection between these two variables. A sleep spindle is found if the sigma band dominates at least one of the 2 second segments of the epoch. Naturally, this means that a high sigma percentage for the epoch results in a higher probabilities for sleep spindles, and similarly for K-complexes and the delta frequency band. In other words the cause-effect relationships ($\delta \rightarrow KK$) and ($\sigma \rightarrow SS$) exist.

There are several ways to add nodes and links to cope for the correlation of both frequency band powers and transient phenomena. As the correlations of the band powers do not appear to be the cause-effect type, the most natural choice of model would be a chain graph as depicted in Figure 4.4.

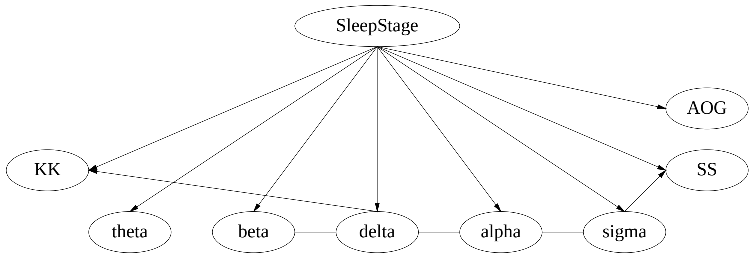

Figure 4.4: A chain graph model containing the three desired correlation between the specified frequency bands and the cause-effect correlation between *delta* and *KK* and between *sigma* and *SS*.

A method to transform the undirected relationships of the chain graph into a Bayesian network is to replace each undirected link with a constraint node with two states, it receives the two nodes connected by the edge as its parents, [Jensen, 1996, p. 48]. This method only applies when there is no evidence on the nodes formerly connected, this is not the case for our classifier since there is always evidence on the input features.

To overcome this problem and to make the idea of a functional relationship more realistic, we distinct between observed and true band power values. Such a notion is somewhat similar to the concept of general dynamic linear models of observation equation and system equation, [West and Harrison, 1989]. The true values could be regarded as the values of some real physiological process occurring in the brain, whereas the observed values are the observable effect of it. The true value is distorted by noise from other activities, errors in the measure equipment, skull, tissue, etc.

Note, the transform of the undirected relationships in the chain graph depicted in Figure 4.4 can also be performed by simply exchanging the undirected edge with a directed. This does not yield any restrictions that the affected nodes may not be observed. Here we need not to make any explicit statements about relationships and let the learner acquire the numbers. We decided to use a constrained model because it contains hidden nodes, making the learning more challenging. Furthermore, our constrained model contains a proposal for describing underlying processes, whereas the conversion of the chain graph does not give any additional understanding of the processes.

A data analysis can fix some of the parameters in the constrained model such that there are fewer parameters. We now claim that among the true values, deterministic functional relationships exist. As the true values never are observed, the transform of [Jensen, 1996] is applicable between the true values. The question is now what kind of functional relationship to use. We base this discussion on the pairwise plots from Figure 4.3, where we can exploit the strength of the non-discretized values. For simplicity we perform the coarse assumption that only a simple linear relationship between the true band powers for the three chosen correlations exists. Figure 4.5 shows how the parameters of the three linear relationships established by fitting lines using least square approximation, [DeGroot, 1985].

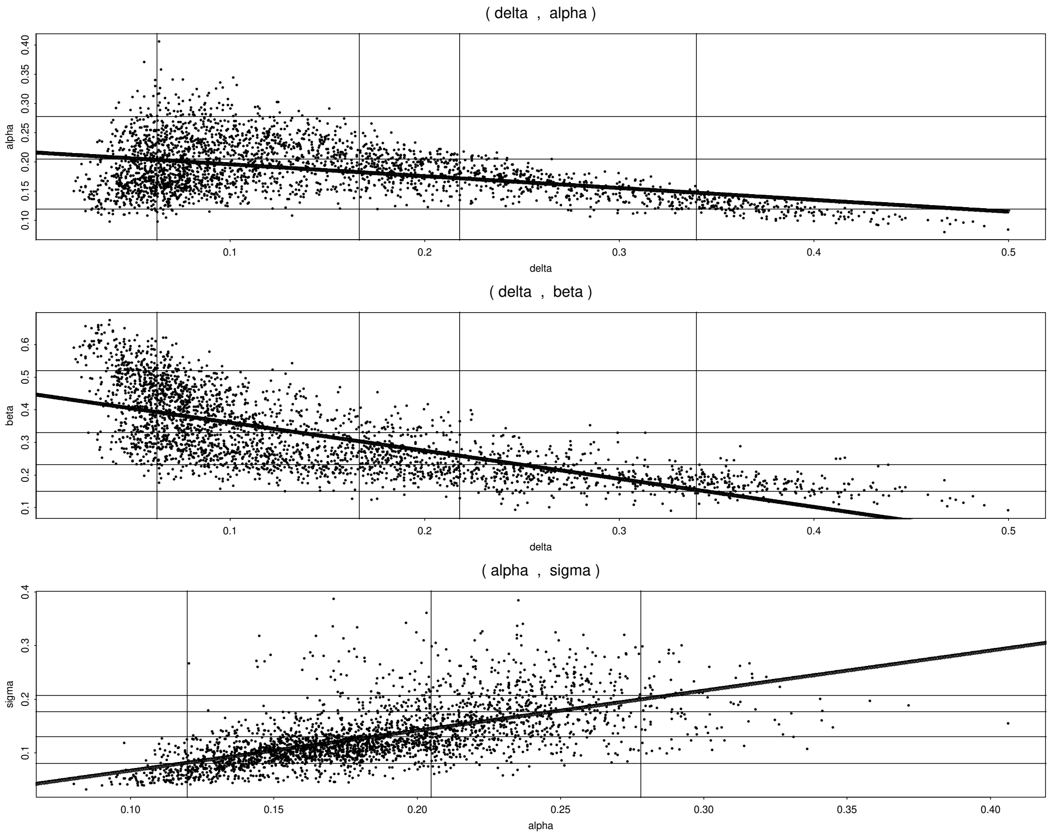

Figure 4.5: Least square fitted line into the plot of the pairs for the three selected pairs with especially high correlation. The vertical and the horizontal lines in the plots show the discretization intervals used for the individual bands.

For each of the constraints the discretized functional relationship between the two variables is modeled as a simple *possible/not possible* relationship. Since noise is modeled between the true and the observed values of the frequency bands, this should be a fair modeling task. Determining whether a state of a constraint node is *possible* or *not possible* is simply a question of whether the fitted line goes through the corresponding rectangle of each discretizing area. Using the modeling trick the conditional probability tables for the three constraint nodes will be as depicted in Table 4.1. All the extensions to the simple-Bayes model yield a new classifier depicted in Figure 4.6.

δ	0				1				2				3				4			
α	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
$C_{\delta\alpha} = \mathrm{y}$	0	0	1	0	0	1	0	0	0	1	0	0	0	1	0	0	1	1	0	0
$C_{\delta\alpha} = \mathrm{n}$	1	1	0	1	1	0	1	1	1	0	1	1	1	0	1	1	0	0	1	1

δ	0					1					2					3					4				
β	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	0	1	2	34	
$C_{\delta\beta} = \mathrm{y}$	0	0	0	1	0	0	0	1	1	0	0	0	1	0	0	0	1	1	0	0	1	1	0	0	0
$C_{\delta\beta} = \mathrm{n}$	1	1	1	0	1	1	1	0	0	1	1	1	0	1	1	1	0	0	1	1	0	0	1	1	1

α	0					1					2					3				
σ	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4
$C_{\alpha\sigma} = \mathrm{y}$	1	1	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	1	1
$C_{\alpha\sigma} = \mathrm{n}$	0	0	1	1	1	1	0	0	1	1	1	1	0	0	1	1	1	1	0	0

Table 4.1: The conditional probability tables of the three constraint nodes, $C_{\delta\alpha}$, $C_{\delta\beta}$, and $C_{\alpha\sigma}$.

A challenging task is to find out in what way the true values are distorted to the observed ones. This can only be assessed by studying the data material in the training set, in our situation we let the learner do the work.

So far we have dealt with the qualitative structure below the *SleepStage* node, in the next section we will concentrate on the part concerning linking of the single epochs, i.e. what we denote as history information.

4.3 History

In this section we discuss some ways of constructing a history component to keep track of information obtained in previous time slices. A reason for looking at history information is that we expect the single-epoch models to oscillate too much between the sleep stages. To reduce these expected oscillations, histories can be applied. History information also gives a more precise picture of the sleep process, according to e.g. [Kemp, 1987, chapter 2] there is some dependency between the sleep stage of the current epoch and the stage of the previous. This dependency is modeled with a Markov chain, see Subsection 4.3.2. Other dependencies such as the amount of time the subject has been asleep so far during the night and the number of REM periods experienced will also be examined in this section. Before discussing different ways of handling history, we will introduce a way of representing time sliced networks.

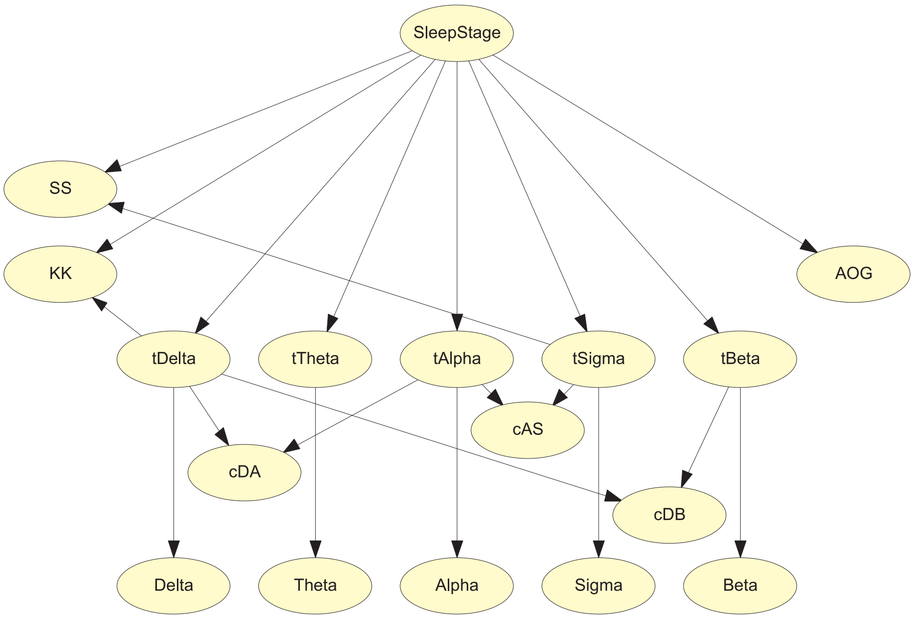

Figure 4.6: The constrained version of the simple-Bayes network where existence of "true" values of the bands are assumed.

4.3.1 Templates and Time Slice Bayesian Networks

In this subsection we will describe a terminology for representing Bayesian networks called a template. The notion of templates translates concepts of the object-oriented paradigm to construction of Bayesian networks. This subsection and the notation used are mainly based upon [Bangsø and Wuillemin, 1999].

Templates are especially suited for representing large networks with lots of replications in a compact way, formally a template is defined as follows.

Definition 4.1 (Template)

A **template** is a way to specify a Bayesian network. It is a DAG over a set of **simple nodes**, which consists of a set of inport nodes, a set of hidden nodes, and a set of export nodes. An **inport node** is a node that has parents outside the template. An **export node** is a node that has children outside the template. A **hidden node** is an internal node in the template, and it cannot be seen outside the template, i.e. it has neither children nor parents outside the template, whereas the import and export nodes can be seen from the outside. The three sets of nodes are pairwise disjoint.

An instantiation of a template is called an **abstract node** that can be used as a node in another template specification; though, no recursion is allowed.

The **underlying Bayesian network** of a template is the Bayesian network corresponding to the DAG of the expanded template, i.e. the traditional network which has to be specified in order to achieve the conditional independence structure.

A simple node is either a real node or a reference node, where a **real node** specifies a chance node, and a **reference node** specifies an import node or an export node only. A reference node only has one parent, which is called a **referenced node**, and the reference node is therefore a placeholder of it, i.e. a reference node is simply a representation of the referenced node, which is placed outside the template. A prior distribution of a reference node is required when no referenced node is specified.

Also two different types of links exist. A **directed link** is a link between two simple nodes as known from regular Bayesian networks, denoted *Simple Node* $\rightarrow$ *Simple Node*. A **reference link** is a link between a simple node and a reference node, denoted *Simple Node* $\Rightarrow$ *Reference Node*; where $A \Rightarrow B$ means that A is referenced by B. In the underlying network A and B will be merged into one node, see Figure 4.7.

Note, in [Bangsø and Wuillemin, 1999] a construction node that has no impact on the underlying network is also introduced, therefore, we have chosen not to use construction links in this thesis. It is also possible to have hierarchies of templates corresponding to inheritance in object-oriented programming, see [Bangsø and Wuillemin, 1999]. Using templates to model our classifier is done only to obtain a compact representation of the time sliced structure, therefore, we have no need for hierarchies in this thesis.

A time sliced Bayesian network is used when the same structure of a part of the network is repeated several times over time. To specify time slices, the basic structure of one slice must be specified. The **basic structure** is the part of the network to be repeated; this can be represented as a template where some of the restrictions described above are relaxed. It is legal to specify **self-reference links**, which is a reference link to an import node within the template.

As an example of a template Figure 4.7 illustrates the time slice template representation for our classifier with expert information from Figure 4.2 extended with a previous stage.

A reference link from the referenced node *Expert* to the reference node *Expert* inside the template is present. Also a reference link from *SleepStage* to *SleepStage_old* is present. In the latter case the link is a self-reference link representing the Markov chain model which will be described in subsection 4.3.2. The meaning of such a link is that *SleepStage_old* is not present in the first time slice, whereas in the remaining it is a parent of *SleepStage*, this is illustrated in Figure 4.8.

When instantiating a time slice template the total number of time slices, N, is required. The instantiation process makes N copies of the template into a single abstract node (also denoted the expanded network), such that each of the self-reference import nodes is represented as one import node in this abstract node, and one import node for each of the other import nodes for each copy. The abstract node also contains export nodes for every copy. The expanded network of the template given in Figure 4.7 discussed above is depicted in Figure 4.8.

Often this abstract node can be intractable large, e.g. in our case we can have up to 799 copies of the template; therefore, some adjustments are preferred. One way to reduce the size of the network is simply to only consider one template at a time, meaning that the time slice window will be of size one (see [Kjærulf, 1995] for a discussion of time slice window sizes).

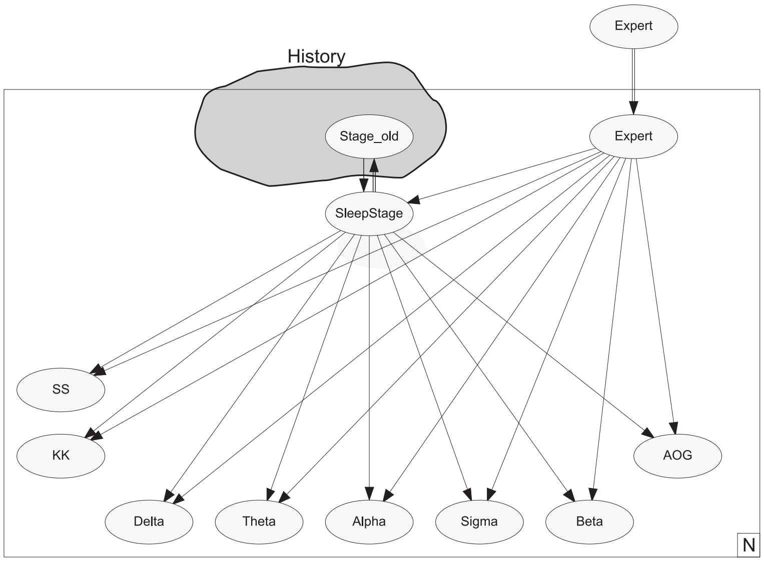

Figure 4.7: The template version of the simple-Bayes classifier where the expert is included and the network is extended with self references through the *Stage_ old* node (Markov chain).

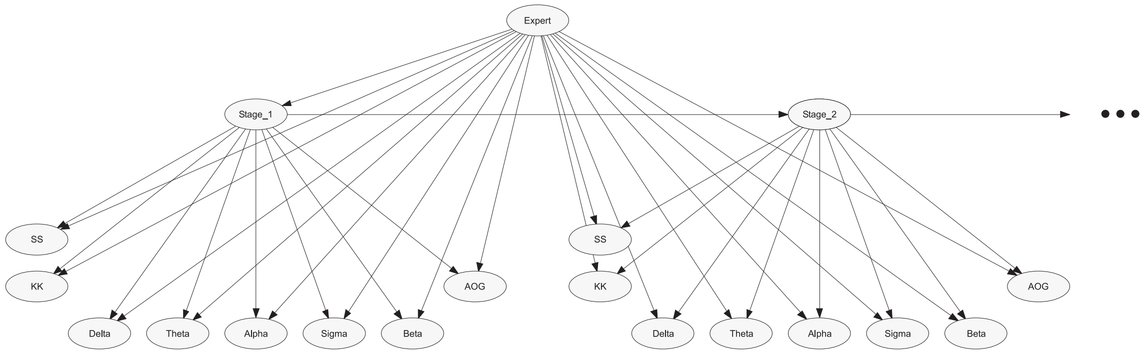

Figure 4.8: Expansion of the time slice template given in Figure 4.7. In each time slice i, *Stage_ i* corresponds to *SleepStage* and *Stage_* $i-1$ corresponds to *Stage_ old*.

We will now discuss how to reduce to total expanded model given in the previous into a single time slice without losing too much information. The loss of information is only present when having a global node which has a child in each of the time slices, e.g. the *Expert* node in Figure 4.8. Since we in Section 4.1 incorporated the expert information directly into the model without using a specific node, there will not be any loss of information. For the classification task we are solving here, the expert node is therefore not necessary.

The idea is now to have a virtual component, which we call a history component, containing all information obtained previously, e.g. information of the previous scored sleep stage (Markov chain), the time the subject has been asleep, the number of REM periods visited. In the next subsections we will discuss how these history components are implemented in time sliced models where only one slice is present at a time.

4.3.2 Markov Chain Model

A sleep scoring is performed for each 30 second epochs, in this approach the sleep stage of each epoch is considered as a stochastic variable (similar to the Bayesian network model in Section 4.1). In [Kemp, 1987] it is shown that the stochastic variables of the current epoch is dependent on the previous. We will now take a Markov chain approach.

Definition 4.2 (Markov chain)
Let $X_0, X_1, \ldots$ be a sequence of stochastic variables having the state space Ω. For each $m > 1$, assign a non-negative number $p_m(x_i, x_j)$ to each pair $(x_i, x_j) \in \Omega \times \Omega$ such that $\forall x_j \in \Omega : \sum_{x_i \in \Omega} p_m(x_i, x_j) = 1$. Then, the sequence is called a **Markov chain** on Ω if it fulfils the Markov property

$$\begin{aligned} P(X_m = x_m | X_{m-1} = x_{m-1}, \ldots, X_0 = x_0) &= P(X_m = x_m | X_{m-1} = x_{m-1}) \\ &= p_m(x_{m-1}, x_m) \end{aligned}$$

for all $m > 1$ and each sequence $x_0, \ldots, x_{m-2}, x, y \in \Omega$.

The probabilities $p_m(x_{m-1}, x_m)$ are called **transition probabilities** and the state X_0 is called the **initial state** of the Markov chain. The initial state is defined from an initial distribution, $P(X_0)$, on Ω.

Taking a Markov chain approach we assume that the current epoch is only dependent of the previous. This means that when determining the score for the current epoch, we must take the previous scoring into account. Thus, a finite state machine can be constructed such that each transition arrow is attached a probability indicating the probability to follow that arrow. A finite state machine is depicted in Figure 4.9.

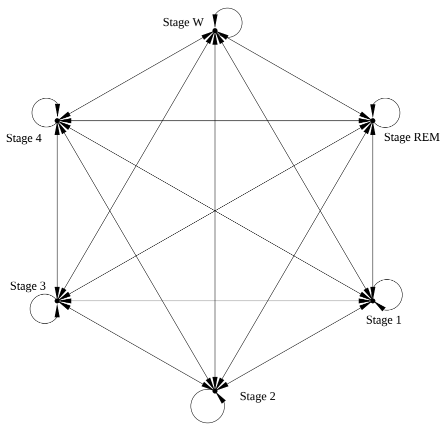

Figure 4.9: Stage transition diagram depicted as a finite state machine illustrating the changing of the sleep stages possible during the sleep, [Nielsen, 1992]. To each arrow a probability indicating the probability to follow that arrow is attached.

An alternative illustration of the process is given in Figure 4.10, in which the transition probability including the probability of being in each of the sleep states are depicted. The larger the area of a stage circle is the more likely is it to be in that sleep stage, see also Figure 2.6 on page 11 showing corresponding probabilities. Likewise, for each arrow its size indicates the probability of the transition.

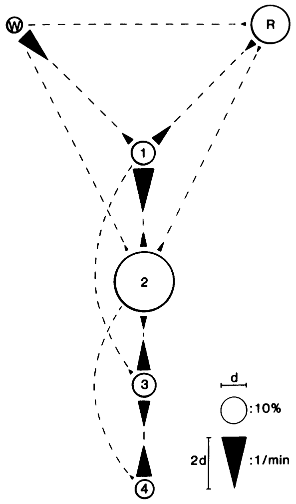

Figure 4.10: Average whole-night sleep structure as estimated from 46 hypnograms of 23 subjects. Circle areas are proportional to the percentage of the time spent in the corresponding stage. Arrow areas are proportional to the corresponding transition probabilities, see Table 4.2, [Kemp, 1987], the arrow in the legend (1/min) corresponds to approximately 15%.

The probabilities attached to each arrow of both Figure 4.9 and Figure 4.10 are given in Table 4.2. The probabilities of this table are to be used to improve the single sliced classifiers described in Sections 4.1 and 4.2. The *SleepStage* node will now, depending on the previous stage, have the vertical row in Table 4.2 corresponding the previous stage as its prior probabilities. The first node of the Markov chain will have the prior probabilities corresponding to the simple-Bayes model described in Section 4.1, the probabilities are depicted in Figure 2.6 on page 11. The Markov chain history model for the simple-Bayes model is depicted in Figure 4.11.

Note, the probabilities in Table 4.2 and in Figure 2.6 are based on a data set much larger than the data set we are exploring. Even though, the probabilities calculated from our training data set are distributed almost similarly.

4.3.3 Changes of the Probabilities During the Sleep

We will now examine how the transition probabilities of the Markov chain model are affected throughout the night. It is well known that the sleep pattern changes during the night, e.g. a lot of deep sleep (stages 3 and 4) in the beginning and the REM periods are longer in the end of the sleep than in the beginning, see the hypnogram in Figure 2.7 on page 11.

First, we look at how the prior probabilities of each sleep stage change during the sleep by

New stage / Present stage	Stage W	Stage REM	Stage 1	Stage 2	Stage 3	Stage 4
Stage W	87.7	0.6	10.7	1.0	0.0	0.0
Stage REM	3.9	92.0	3.4	0.7	0.0	0.0
Stage 1	6.5	5.2	74.6	13.6	0.1	0.0
Stage 2	2.7	0.9	1.4	90.5	4.3	0.2
Stage 3	1.2	0.1	0.3	25.1	67.6	5.7
Stage 4	1.0	0.4	0.0	6.8	17.8	74.0

Table 4.2: Transition probability matrix on stage transitions taken from 500 visually scored sleep recordings. Horizontally the table shows the probability of changing to a (new) sleep stage from a present (shown vertically), [Nielsen, 1992].

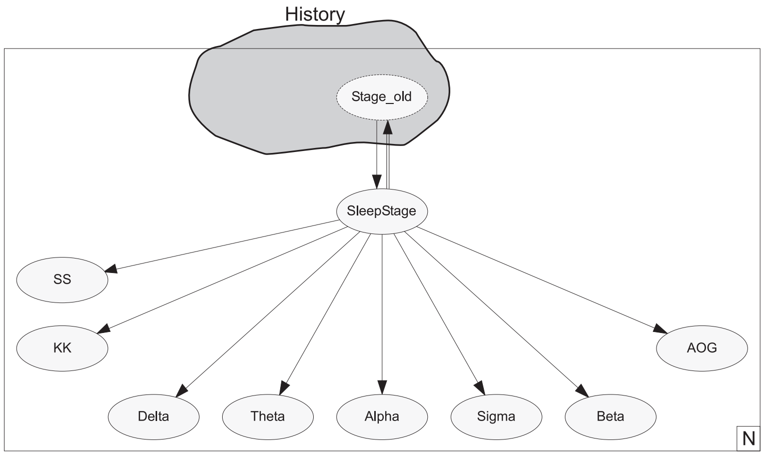

Figure 4.11: The simple-Bayes model extended with Markov chain history which takes the previous scoring into consideration.

considering the amount of time asleep. Second, we look at how the transition probabilities of the Markov chain model, described in Subsection 4.3.2, changes over time. Finally, we will investigate how the prior probabilities of *SleepStage* changes from sleep cycle to sleep cycle, i.e. from REM period to REM period.

Changes of the Prior Probabilities During the Sleep

To examine the changes of the prior probabilities of *SleepStage* during the sleep, we have divided the sleep time into segments of 60 minutes, i.e. 120 epochs each. The model is depicted in Figure 4.12, where the time history component is represented by the *Time* node. For each of the epochs we will calculate (by counting the frequencies) the prior probabilities. The results of this investigation are depicted in Table 4.3.

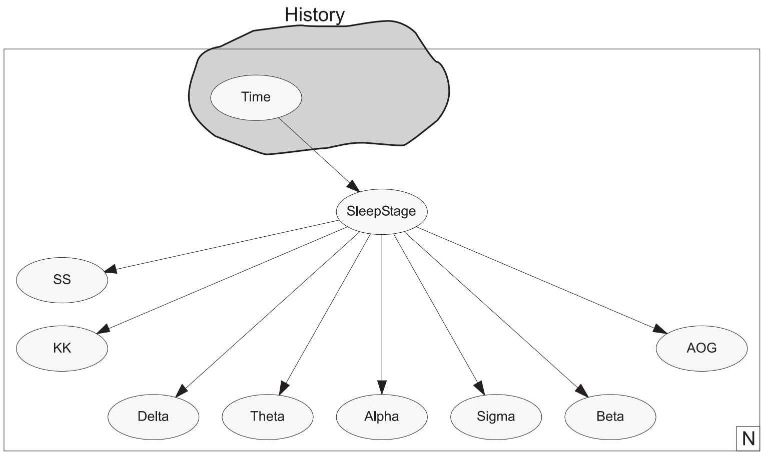

Figure 4.12: The model where the scorings depend on the amount of time the subject has been asleep.

Epochs	Stage W		Stage REM		Stage 1		Stage 2		Stage 3		Stage 4	
	#	%	#	%	#	%	#	%	#	%	#	%
$[1, 120]$	100	21.3	0	0.0	35	7.5	77	16.4	45	9.6	212	45.2
$[121, 240]$	35	7.5	53	11.4	11	2.4	204	43.7	85	18.2	79	16.9
$[241, 360]$	3	0.6	183	38.1	0	0.0	183	38.1	45	9.4	66	13.8
$[361, 480]$	7	1.5	133	27.7	1	0.2	246	51.3	40	8.3	53	11.0
$[481, 600]$	20	4.2	199	41.5	3	0.6	203	42.3	14	2.9	41	8.5
$[601, 799]$	20	5.7	104	29.6	4	1.1	217	61.7	7	2.0	0	0.0

Table 4.3: Prior probabilities for each one hour interval taken from the data set consisting of visually scored sleep recordings of the training data set. The # symbol denotes the number of epochs falling into each cell, and the % symbol denotes the percentage of each count given the total count of the respective interval. Note, since a lot of missing values exist in the end of each of the sleep recordings, the last interval has been chosen to be 100 minutes in stead of one hour; for general purpose it should be ∞.

From this table it can be seen that the amount of sleep stage W is largest in the beginning and in the end, and almost absent in the middle of the sleep. The amount of REM sleep increases during the sleep while the amount of sleep stage 4 decreases. Hence, it would be reasonable to include the time history component in a history model.

Changes of the Transition Probabilities During the Sleep

Examining the changes of the transition probabilities of the Markov chain during the sleep, we divided the sleep time into segments of 90 minutes, i.e. 180 epochs, each; this corresponds approximately to a sleep cycle, i.e. the time from one REM period to the next. The simple-Bayes history model using both the Markov chain and the time histories is depicted in Figure 4.13. For each of the epochs we will calculate (by counting the frequencies) the probabilities for *SleepStage*. The results of this investigation are depicted in Table 4.4.

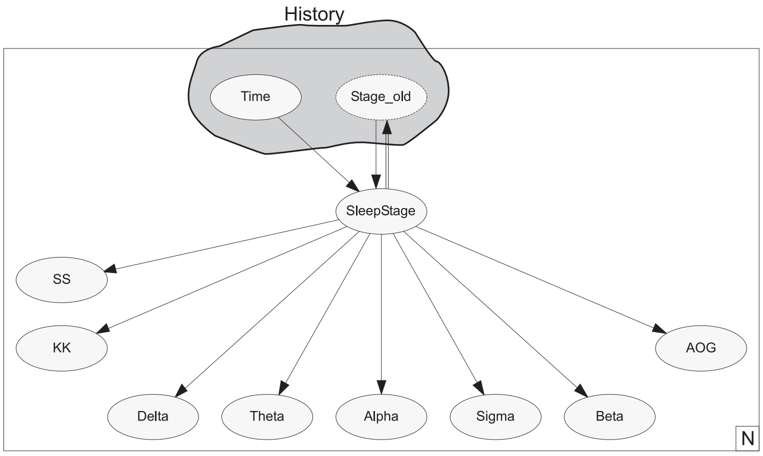

Figure 4.13: The model where the scorings depend on the amount of time the subject has been asleep and the last scoring.

Analyzing the numbers this table is little more complicated than before because there are not as many clear tendencies as before. But, it can though be concluded that the transition probabilities changes over time. For example, the transition probabilities for going from sleep stage 2 to sleep stage 3 decreases during the night, and the probability for changing from sleep stage REM to sleep stage 2 also decreases. Therefore, it would probably be a good idea to investigate whether this combined history yield better classifications than the other approaches.

Changes of the Probabilities Conditioned on the Number of REM Periods

We will now investigate the probabilities when taking the number of REM periods visited into account. We will look at both prior probabilities of *SleepStage* and the transition probabilities of the Markov chain. In Figure 4.14 the model where the scoring is depending on the number of visited REM periods is depicted.

New stage / Present stage	Stage W #	Stage W %	Stage REM #	Stage REM %	Stage 1 #	Stage 1 %	Stage 2 #	Stage 2 %	Stage 3 #	Stage 3 %	Stage 4 #	Stage 4 %
[0, 180]												
Stage W	103	83.74	0	0.00	11	8.94	7	5.69	1	0.81	1	0.81
Stage REM	1	6.25	9	56.25	0	0.00	6	37.50	0	0.00	0	0.00
Stage 1	5	13.89	0	0.00	24	66.67	7	19.44	0	0.00	0	0.00
Stage 2	9	4.81	7	3.74	1	0.53	156	83.42	14	7.49	0	0.00
Stage 3	1	1.16	0	0.00	0	0.00	12	13.95	61	70.93	12	13.95
Stage 4	3	1.14	0	0.00	0	0.00	0	0.00	10	3.79	251	95.08
[180, 360]												
Stage W	10	52.63	0	0.00	4	21.05	5	26.32	0	0.00	0	0.00
Stage REM	1	0.45	208	94.12	0	0.00	12	5.43	0	0.00	0	0.00
Stage 1	1	10.00	0	0.00	6	60.00	3	30.00	0	0.00	0	0.00
Stage 2	3	1.08	13	4.69	0	0.00	241	87.00	19	6.86	1	0.36
Stage 3	2	2.25	0	0.00	0	0.00	15	16.85	64	71.91	8	8.99
Stage 4	2	2.13	0	0.00	0	0.00	1	1.06	6	6.38	85	90.43
[360, 540]												
Stage W	7	38.89	3	16.67	2	11.11	6	33.33	0	0.00	0	0.00
Stage REM	2	1.06	172	91.01	0	0.00	15	7.94	0	0.00	0	0.00
Stage 1	0	0.00	0	0.00	1	33.33	2	66.67	0	0.00	0	0.00
Stage 2	7	1.83	14	3.67	0	0.00	351	91.88	10	2.62	0	0.00
Stage 3	1	2.22	0	0.00	0	0.00	8	17.78	29	64.44	7	15.56
Stage 4	1	1.16	0	0.00	0	0.00	0	0.00	6	6.98	79	91.86
[540, 799]												
Stage W	21	75.00	1	3.57	3	10.71	3	10.71	0	0.00	0	0.00
Stage REM	1	0.40	240	96.00	0	0.00	9	3.60	0	0.00	0	0.00
Stage 1	0	0.00	0	0.00	2	40.00	3	60.00	0	0.00	0	0.00
Stage 2	6	2.08	8	2.78	0	0.00	264	91.67	9	3.13	1	0.35
Stage 3	0	0.00	0	0.00	0	0.00	8	50.00	7	43.75	1	6.25
Stage 4	1	12.50	0	0.00	0	0.00	1	12.50	0	0.00	6	75.00

Table 4.4: Transition probabilities for each 180 epoch interval taken from the data set consisting of visually scored sleep recordings of the four subjects in the training data set.

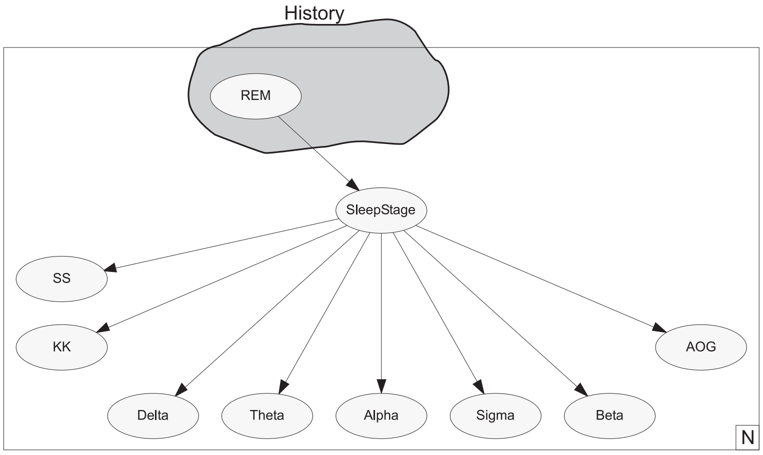

Figure 4.14: The model where the scorings depend on the number of REM periods visited to a particular point of the sleep.

In order to solve this problem, we had to add an extra column to the data set containing the number of REM periods visited to each point. The number indicating the number of REM periods visited is increased each time the subject leaves a REM period and no new sleep stage REM is scored within the e.g. next 12 epochs.

The probabilities of *SleepStage* when taking the number of REM periods into account are computed by counting, and the result of this is depicted in Table 4.5. The results show that sleep stage 4 is almost only present in the 0-0 and the 2-2 intervals, whereas sleep stage 2 is most present in the 1-1, 3-3, and 4-9 intervals. As for time history, sleep stage REM is more present in the end of the sleep than in the beginning, except for the 4-9 interval because the quality of the data and the scorings are doubtful in the end of the sleep.

No. of REM periods	Stage W		Stage REM		Stage 1		Stage 2		Stage 3		Stage 4	
	#	%	#	%	#	%	#	%	#	%	#	%
0-0	121	13.8	92	10.5	36	4.1	206	23.4	123	14.0	301	34.2
1-1	21	4.2	144	28.7	10	2.0	259	51.6	45	9.0	23	4.6
2-2	11	3.0	122	33.1	2	0.5	91	24.7	26	7.0	117	31.7
3-3	7	1.2	255	43.8	1	0.2	291	50.0	26	4.5	2	0.3
4-9	29	7.3	59	14.8	5	1.3	283	70.8	16	4.0	8	2.0
0-9	189	6.9	672	24.7	54	2.0	1130	41.4	236	8.6	451	16.5

Table 4.5: The prior probabilities of the *Sleep Stage* of the simple model given the number of REM periods visited. The last row indicates the probabilities without take the number of REM periods visited into account. The probabilities (frequencies) are based on the training data set.

Similar to Figure 4.13 the combination of REM periods and Markov chain is investigated. The transition probabilities from the investigation are depicted in Table 4.6. As concluded for the transition probabilities in Table 4.4 the probability for changing from sleep stage 2 to sleep stage 3 is decreasing during the night.

New stage / Present stage	Stage W #	Stage W %	Stage REM #	Stage REM %	Stage 1 #	Stage 1 %	Stage 2 #	Stage 2 %	Stage 3 #	Stage 3 %	Stage 4 #	Stage 4 %
0 − 0												
Stage W	101	83.5	0	0.0	11	9.1	7	5.8	1	0.8	1	0.8
Stage REM	3	3.3	82	90.1	0	0.0	6	6.6	0	0.0	0	0.0
Stage 1	5	13.9	0	0.0	24	66.7	7	19.4	0	0.0	0	0.0
Stage 2	7	3.4	10	4.9	1	0.5	165	80.1	23	11.2	0	0.0
Stage 3	2	1.6	0	0.0	0	0.0	21	17.1	83	67.5	17	13.8
Stage 4	2	0.7	0	0.0	0	0.0	0	0.0	16	5.3	283	94.0
1 − 1												
Stage W	12	57.1	0	0.0	4	19.1	5	23.8	0	0.0	0	0.0
Stage REM	1	0.7	134	93.7	0	0.0	8	5.6	0	0.0	0	0.0
Stage 1	1	10.0	0	0.0	6	60.0	3	30.0	0	0.0	0	0.0
Stage 2	5	1.9	10	3.9	0	0.0	235	90.7	9	3.5	0	0.0
Stage 3	1	2.2	0	0.0	0	0.0	6	13.3	36	80.0	2	4.4
Stage 4	1	4.3	0	0.0	0	0.0	1	4.3	0	0.0	21	91.3
2 − 2												
Stage W	7	63.6	2	18.2	1	9.1	1	9.1	0	0.0	0	0.0
Stage REM	1	0.8	111	91.7	0	0.0	9	7.4	0	0.0	0	0.0
Stage 1	0	0.0	0	0.0	1	50.0	1	50.0	0	0.0	0	0.0
Stage 2	1	1.1	9	9.89	0	0.0	77	84.6	4	4.396	0	0.0
Stage 3	0	0.0	0	0.0	0	0.0	2	7.7	17	65.4	7	26.9
Stage 4	2	1.7	0	0.0	0	0.0	0	0	5	4.3	110	94.0
3 − 3												
Stage W	0	0	2	28.6	1	14.3	4	57.1	0	0.0	0	0.0
Stage REM	2	0.8	245	96.5	0	0.0	7	2.8	0	0.0	0	0.0
Stage 1	0	0.0	0	0.0	0	0.0	1	100.0	0	0.0	0	0.0
Stage 2	4	1.4	8	2.7	0	0.0	272	93.5	7	2.4	0	0.0
Stage 3	1	3.8	0	0.0	0	0.0	6	23.1	18	69.2	1	3.8
Stage 4	0	0.0	0	0.0	0	0.0	0	0.0	1	50.0	1	50.0
4 − 9												
Stage W	21	75.0	0	0.0	3	10.7	4	14.3	0	0.0	0	0.0
Stage REM	0	0.0	57	96.6	0	0.0	2	3.4	0	0.0	0	0.0
Stage 1	0	0.0	0	0.0	2	40.0	3	60.0	0	0.0	0	0.0
Stage 2	7	2.5	2	0.7	0	0.0	264	93.3	9	3.2	1	0.4
Stage 3	0	0.0	0	0.0	0	0.0	8	50.0	7	43.8	1	6.3
Stage 4	1	12.5	0	0.0	0	0.0	1	12.5	0	0.0	6	75.0

Table 4.6: Transition probabilities for each sleep cycle interval (the interval between REM periods) based on the training data set.

Generally, when using Markov chain histories the amount of data is too little, in many cases of both Table 4.4 and Table 4.6 the percentage cannot be compared since only a few counts form the basis for the calculation of these percentages. Hence, investigations are needed to confirm or deny that the use of Markov chain histories where the transition probabilities are based on the training data set improve the model.

4.4 Type of Person

The type of person is also relevant when classifying sleep stages. For example, if a person has a very thick skull with lots of tissue the amplitude of the EEG signals will not be as strong and bone could work as a filter of certain frequencies. The amount of δ activity is known to decrease with age, [Nielsen, 1992], but whether this is because of true changes in the sleep behavior or simply due to changes in the structure of the tissue filtering especially δ activity is not known. In order to investigate whether there might be a similar phenomena such as skull type present in our data set, we plot the band averages for all six persons, see Figure 4.15.

The figure shows that for sleep stage 4 the first two persons (cont01 and cont03) differ from the others, this difference is especially distinct for the δ and α bands. This confirms our hypothesis about the existence of the *PersonType* phenomena. The reason for the variations in persons could then be either e.g. type of skull or possibly difference in the experiment setups. The latter could for example be due to use of two different brand of electrodes. A fruitful idea to improve the quality of learning could be to include *PersonType* in the network and then use adaptation as known from [Olesen *et al.*, 1992] to adapt the classifier to the patient's type, thus improving classifications. Adding *PersonType* results in a network as in Figure 4.16.

Unfortunately, with this structure the template has to be expanded when learning. Our data set does not contain any information about person type, hence the variable will never be instantiated. Furthermore, we judge that a data material of six persons is too sparse to confirm the existence of a person type variable. Therefore, no *PersonType* variable is used for now.

As an aside, the person type variable could also represent a category variable containing several sleep disorders, which is highly relevant when classifying sleep stages. For example, if a person is suffering from Insomnia, changes in the EEG occurs, e.g. reduced amount of slow wave sleep (sleep stages 3 and 4), higher amount of high frequency EEG activity, longer sleep latency (also called REM periods), and several awakenings during the sleep, [Nielsen, 1992]. If the insomnia is chronic, the number of sleep spindles per minute might also be reduced. Therefore, a variable indicating the disorder of a person will make the classification more accurate, or it can indicate whether the person is suffering from some sleep disorder or not. All our subjects are normal controls, i.e. they are not suffering from any sleep disorder, therefore no nodes of this kind are added.

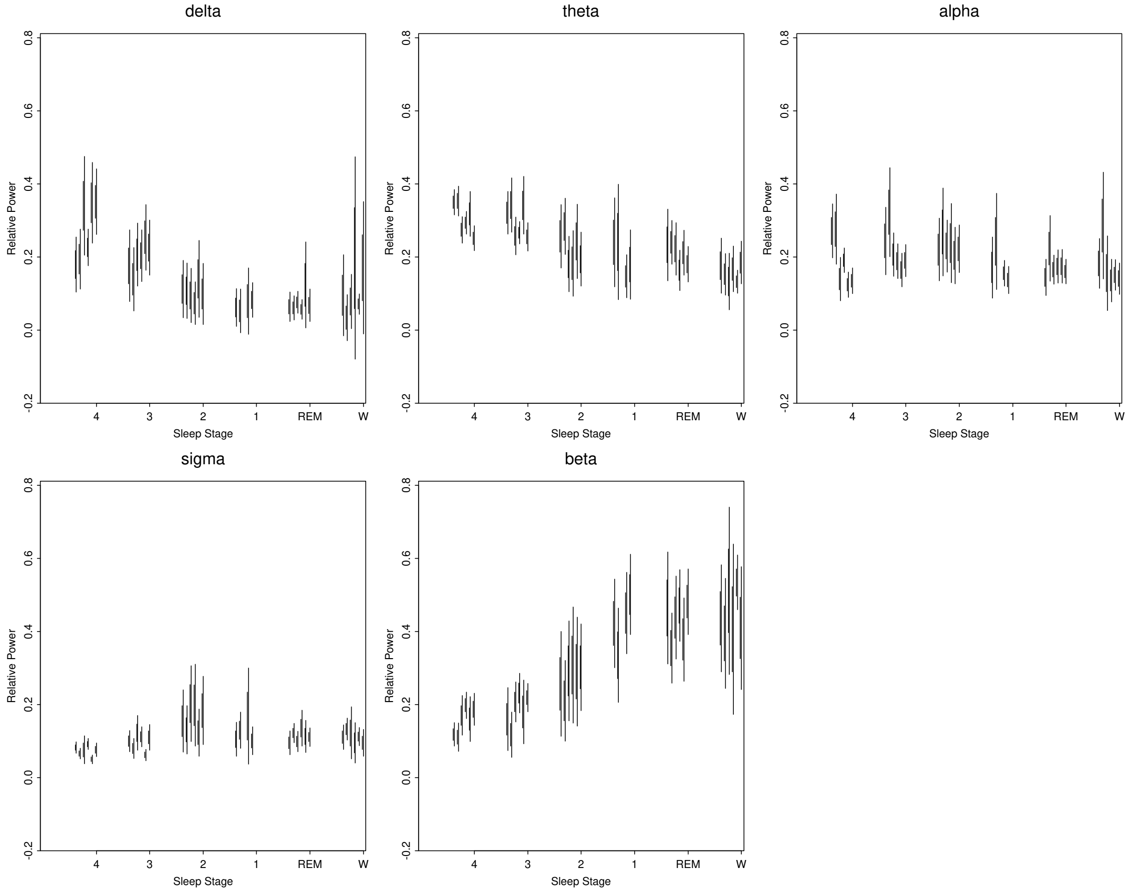

Figure 4.15: Relative band power value distributions for the six normal controls for each sleep stage. For each band feature, f, each sleep stage, s, and each person, p, the sample mean, $\hat{\mu}_{fsp}$, and the sample variance, $\hat{\sigma}^2_{fsp}$, are calculated of all f-measurements of p classified as s. The plot then contains two lines illustrating the intervals $\hat{\mu}_{fsp} \pm \hat{\sigma}_{fsp}$ and $\hat{\mu}_{fsp} \pm 2\hat{\sigma}_{fsp}$, respectively. Note, to make the plots less noisy only epochs where the two experts agree have been used. For each stage the patients are plotted by increasing order from left to right.

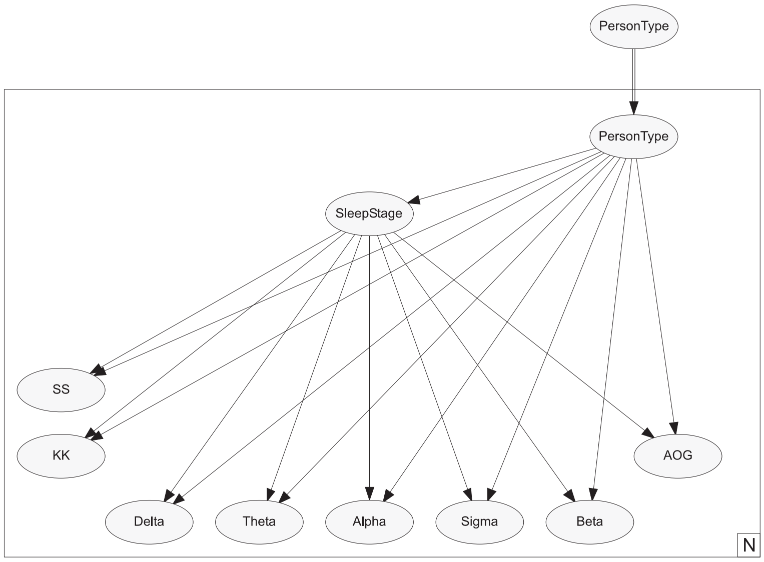

Figure 4.16: Template specification for a network containing a *PersonType* variable.

4.5 Summary

In this chapter we presented two single-epoch sleep classification models, the simple-Bayes model and the constrained model. By analyzing the data we saw that the independence assumptions of the simple-Bayes model were too coarse, thus the constrained model copes with some of the identified correlations between input features. The single-epoch models were extended with history information. The history information considered in the rest of this thesis will be the time and the Markov chain histories. Calculating the current state of the REM history will be difficult to handle in the sleep classifier and it is doubtful that it will yield better results than the time history.

Hence, the models to be evaluated on the cross-validation data set in Chapter 7 will be the two single-epoch models without any history, with Markov chain history, with time history, and finally with the histories in combination. These evaluations will reveal which model gives the best results for the sleep classification task. Before any evaluation can be performed, we need to asses the quantitative structure of the proposed networks.

Chapter 5

Learning and Sampling Algorithms

In this chapter we describe the different learning algorithms used in the solution of the sleep classification problem. We start by considering two types of learning algorithms, the EM-algorithm as the sound statistical procedure, and the more ad-hoc based gradient descent algorithms. The latter equipped with the right metric function enables us to transform the training of neural networks into a Bayesian network context, this is especially suitable in classification problems like our sleep stage classification task. Our aim is to later perform tests between the EM algorithm and neural network type training to compare them in the context of classification problems. To perform these kind of tests we need to be able to sample data cases from a predefined known distribution. This gives us a tool to simulate the sampling step in the ideal learning cycle, see Figure 5.1.

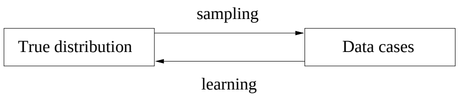

Figure 5.1: The ideal learning cycle. In real world problems only observed data cases are in our possession. These are assumed to be sampled from the true distribution which we try to find by learning methods.

5.1 Learning Bayesian Networks

The learning of a Bayesian network is a two-step procedure, the first step is the learning of the structure, and the second is the learning of the conditional probabilities produced by this structure. Various tools exist for learning the structure, e.g. Bifrost [Thiesson, 1996] and Power Constructor [Jie Cheng, 1999]. Learning of the structure will not be discussed further in this thesis because experience shows that building the structure with help of experts is relatively straightforward, [Coupé and van der Gaag, 1998a]. Instead we focus on learning the quantitative structure (the conditional probability tables) in general.

Let $B = (G, P)$ be a Bayesian network with underlying graph G and probability distribution P. Assume the structure of B is known (and fixed), and let $\boldsymbol{\theta} = \theta_1, \ldots, \theta_n$ be the unknown parameters in the conditional probability tables constituting P. Note, not all tables necessarily contain unknown parameters, and not all parameters in a table are necessarily unknown. The parameter space, Θ, is the set of all possible values $\boldsymbol{\theta}$ can attain.

We focus our study of learning methods on batch learning. In the sleep stage classification task the training set is given, no new training observations arrive online, and no desire to adapt to changing contexts exists. This makes the task more suitable for batch learning than for adaptation. Let $\mathbf{d} = \{d_1, \dots, d_N\}$ be the set of data cases from which $\boldsymbol{\theta}$ will be learned, where N is called the **sample size**. Each data case d_l, $l \in \{1, \dots, N\}$, is a configuration of the variables $V_l \subseteq V(G)$. If $V_l \neq V(G)$ the data case is called an **incomplete observation**. Assume $v \in V(G)$, if $\forall l : V_l \cap \{v\} = \emptyset$, v is called a **hidden variable** because it is never observed. On the other hand, if $\nexists l : V_l \cap \{v\} = \emptyset$, the node v is **fully observable**. If all variables in the network are fully observable, the entire network is called fully observable, [Russel *et al.*, 1995]. Furthermore, by $\mathcal{I}_{V(G)}(\mathbf{d}) \subseteq \mathcal{I}_{V(G)}$ we denote the set of distinct cases in $\mathbf{d}$.

These definitions give us the framework for describing incomplete data problems. We distinguish between hidden variables and variables that are only unobserved from time to time. This kind of sporadic missing data could be a result of omissions when writing patient journals, unstable measurement equipment, or results of tests which are too costly to perform on all individuals. Mixed up data problems as described in [Thiesson, 1991], where one is not sure about the correctness of the observed states, are not treated in this thesis since this problem does not occur in our sleep stage classification task.

The following sections describe various algorithms for learning Bayesian networks with incomplete data, starting with the EM algorithm. Hereafter, we focus on training algorithms especially suited for classification problems.

5.2 Learning Bayesian Networks with the EM-Algorithm

The **EM-algorithm** is a general purpose algorithm for maximum likelihood estimation in missing data problems. A special version for hierarchical log-linear models and Bayesian networks is given in [Lauritzen, 1991], this includes adaptation of the EM-algorithm to a Bayesian framework by calculating a maximum a posteriori estimate. This section is mainly based on the work and notation in [Lauritzen, 1991] and [Thiesson, 1996].

The EM-algorithm in its basic form is an iterative procedure of computing a **maximum likelihood estimate** (MLE). Let $\boldsymbol{\theta}^n$ denote the values of the parameters after the n-th iteration. If no missing data exist the MLE is found in one iteration by performing simple frequency counting in the database. If missing data exists, we insert a preliminary step where we use propagation in the network based on $\boldsymbol{\theta}^n$ to estimate the values of the missing variables for each case with missing data. This estimation step yields an equivalent fully observed database in which we again can perform frequency counting. The hereby obtained values, $\boldsymbol{\theta}^{n+1}$, are now used to spawn a new run of the procedure. It can be shown that infinite repetition of the procedure implies convergence towards an extrema of the likelihood function (usually a maximum). Good approximations of this extrema are already obtained after a small number of iterations.

We now treat the EM-algorithm in a more formal way. The above description shows that the EM-algorithm consists of two fundamental steps.

- **Estimation-step:** Form the conditional expectation of the log-likelihood function for complete data given both the observed data and current parameter values,
$$Q(\boldsymbol{\theta}|\boldsymbol{\theta}^n, \mathbf{d}) = \mathbb{E}_{\boldsymbol{\theta}^n}[\log f(\mathbf{X}|\boldsymbol{\theta})|\mathbf{d}], \tag{5.1}$$
where $\mathbf{X}$ is a random vector corresponding to the complete data (unobserved) with density function f, and $\mathbf{d}$ denotes the observed data which can contain incomplete observations. In case of complete data we have $\mathbf{X} = \mathbf{d}$, hence $Q(\boldsymbol{\theta}|\boldsymbol{\theta}^n, \mathbf{d}) = \log f(\mathbf{d}|\boldsymbol{\theta})$, i.e. the log-likelihood. If $\mathbf{d}$ is not fully observed, the current parameters are used to estimate the log-likelihood of the cases with missing data.

- **Maximizing-step:** Maximize $Q(\boldsymbol{\theta}|\boldsymbol{\theta}^n, \mathbf{d})$ in $\boldsymbol{\theta}$, i.e. determine
$$\boldsymbol{\theta}^{n+1} = \arg\max_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} Q(\boldsymbol{\theta}|\boldsymbol{\theta}^n, \mathbf{d}). \tag{5.2}$$

It can be shown that the sequence $\{\boldsymbol{\theta}^k; k \to \infty\}$ under general conditions converges to a stationary value, which is either a saddlepoint, a local minimum, or a local maximum of the likelihood function, [Thiesson, 1991, p. 25] and [McLachlan and Krishnan, 1997, p. 85]. In many practical applications this stationary value will be a local maximum. Proving these results is beyond the scope of this thesis and we refer to [McLachlan and Krishnan, 1997, Chapter 3] for further information. Generalizations of the EM-algorithm exists, e.g. GEM where the Q-function is not maximized but only strictly increased in the M-step.

In the Bayesian framework $\boldsymbol{\theta}$ is regarded as a random vector and information about $\boldsymbol{\theta}$ is expressed by a prior distribution $\pi(\boldsymbol{\theta})$. From this the posterior distribution is given by Bayes rule

$$P(\boldsymbol{\theta}|\mathbf{d}) \propto P(\mathbf{d}|\boldsymbol{\theta})\pi(\boldsymbol{\theta}).$$

Since we are trying to maximize the log likelihood, knowledge about proportionality is sufficient because the value $\boldsymbol{\theta}$ maximizing an arbitrary function $f(\boldsymbol{\theta})$ also maximizes $a \cdot f(\boldsymbol{\theta}) + b$, for any constants a and b.

By considering the posterior distribution as a posterior likelihood we are interested in finding a **maximum a posteriori estimate** (MAPE). This is obtained by replacing the E-step in the MLE version with expectation of the posterior log-likelihood $Q^*(\boldsymbol{\theta}|\boldsymbol{\theta}^n, \mathbf{d})$, hence the E-step is replaced by

$$Q^*(\boldsymbol{\theta}|\boldsymbol{\theta}^n, \mathbf{d}) \;=\; \mathbb{E}_{\boldsymbol{\theta}^n}[\log P(\boldsymbol{\theta}|X)|\mathbf{d}] \;=\; Q(\boldsymbol{\theta}|\boldsymbol{\theta}^n, \mathbf{d}) + \log \pi(\boldsymbol{\theta}). \tag{5.3}$$

The M-step now consists of maximizing Q^* instead of Q. Note, in case of a uniform prior, i.e. $\pi(\boldsymbol{\theta}) \equiv 1$, MAPE reduces to MLE.

We now proceed by considering MLE which was a question of finding an expression for the Q function and then maximizing it. Hereafter, we consider performing MAPE once an expression for the prior distribution has been obtained. Finally, we describe a method for translating information about the parameters into a prior distribution on their values.

5.2.1 Maksimum Likelihood Estimation

The unknown probability distribution P for a Bayesian network, with nodes Δ is assumed to factorize according to

$$P(i) \quad = \quad \prod_{\delta \in \Delta} P(i_\delta | i_{\mathrm{pa}(\delta)}), \tag{5.4}$$

where $i \in \mathcal{I} = \times_{\delta \in \Delta} \mathcal{I}_\delta$ is the Cartesian product of the individual state spaces, and the **cell** i_A denotes a configuration of the variables in the set A. By considering the Bayesian network as based upon multinomial sampling, [DeGroot, 1985, pp. 297–299], the likelihood function for complete data can be factorized by exploiting (5.4),

$$L(P) \quad \propto \quad \prod_{i \in \mathcal{I}} P(i)^{n(i)} \quad = \quad \prod_{i \in \mathcal{I}} \left(\prod_{\delta \in \Delta} P(i_\delta | i_{\mathrm{pa}(\delta)}) \right)^{n(i)} \quad = \quad \prod_{\delta \in \Delta} \prod_{i \in \mathcal{I}} P(i_\delta | i_{\mathrm{pa}(\delta)})^{n(i)},$$

where $n(i)$ is the number of counts for cell i. Let $\mathrm{cl}(\delta) = \delta \cup \mathrm{pa}(\delta)$ be the closure of δ. Now, for all $i \in \mathcal{I}$ resulting in the same cell i_c, $c \in \mathrm{cl}(\delta)$; when we only classify according to $\mathcal{I}_{\mathrm{cl}(\delta)}$, the value of $P(i_\delta | i_{\mathrm{pa}(\delta)})$ is identical which means that the exponents can be added. By $n(i_a)$ we denote the marginal counts, $n(i_a) = \sum_{j: j_a = i_a} n(j)$, i.e. the number of count when the cases are only classified according to the criteria in a. Hence,

$$L(P) \quad \propto \quad \prod_{\delta \in \Delta} \prod_{i_{\mathrm{cl}(\delta)} \in \mathcal{I}_{\mathrm{cl}(\delta)}} P(i_\delta | i_{\mathrm{pa}(\delta)})^{n(i_{\mathrm{cl}(\delta)})} \quad = \quad \prod_{\delta \in \Delta} L_\delta(p). \tag{5.5}$$

L_δ is the likelihood function obtained by sampling the variables in $\mathrm{cl}(\delta)$ with fixed $i_{\mathrm{pa}(\delta)}$. As there are no further restrictions the overall likelihood can be maximized by maximizing each of the factors L_δ.

To obtain the E-step we take the conditional expectation of the log-likelihood function as instructed by (5.1). Taking the conditional expectation in (5.5) corresponds to estimating the number of expected counts $n(i_{\mathrm{cl}(\delta)})$ using the current parameters. The table $P(i_\delta | i_{\mathrm{pa}(\delta)})$ only consists of elements from $\boldsymbol{\theta}$ and has no relation to $\mathbf{d}$, thus it is not considered. Hence,

$$Q(P' | P, \mathbf{d}) \quad = \quad \sum_{\delta \in \Delta} \sum_{i_{\mathrm{cl}(\delta)} \in \mathcal{I}_{\mathrm{cl}(\delta)}} n^*(i_{\mathrm{cl}(\delta)}) \log P'(i_\delta | i_{\mathrm{pa}(\delta)}), \tag{5.6}$$

where $n^*(i_{\mathrm{cl}(\delta)}) = \mathbb{E}_P[N(i_{\mathrm{cl}(\delta)}) | \mathbf{d}]$ is the expected number of counts, P is the current distribution, and P' the distribution as a function of $\boldsymbol{\theta}$. The random variable $N(i_{\mathrm{cl}(\delta)})$ denotes the (unobserved) number of counts for the cell $i_{\mathrm{cl}(\delta)}$ in the situation with no missing data.

To calculate n^* we assume that $\mathbf{d}$ consists of N independent cases $i^1_{b^1}, i^2_{b^2}, \ldots, i^N_{b^N}$, where case v, $v \in \{1, \ldots, N\}$, consists of evidence for the variables in the set $b^v \subseteq \Delta$. Note, $i^v_{b^v}$ is equivalent to d_v. Furthermore, let $i^{(1)}_{b^{(1)}}, \ldots, i^{(M)}_{b^{(M)}}$ be the elements of $\mathcal{I}_\Delta(\mathbf{d})$ with counts

$c_1, \dots, c_M$. Hence,

$$
\begin{aligned}
n^*(i_{\mathrm{cl}(\delta)}) &= \mathbb{E}_p[N(i_{\mathrm{cl}(\delta)})|i^1_{b^1}, i^2_{b^2}, \dots, i^N_{b^N}] \\
&= \sum_{v=1}^{N} \mathbb{E}_p[\ \chi^v(i_{\mathrm{cl}(\delta)})|i^1_{b^1}, i^2_{b^2}, \dots, i^N_{b^N}] \\
&= \sum_{v=1}^{N} \mathbb{E}_p[\ \chi^v(i_{\mathrm{cl}(\delta)})|i^v_{b^v}] \\
&= \sum_{v=1}^{N} P(i_{\mathrm{cl}(\delta)}|i^v_{b^v}) \\
&= \sum_{v=1}^{M} c_v \cdot P(i_{\mathrm{cl}(\delta)}|i^{(v)}_{b^{(v)}}),
\end{aligned}
\tag{5.7}
$$

where $\chi^v(i_{\mathrm{cl}(\delta)})$ is an indicator function checking whether case v falls into cell $i_{\mathrm{cl}(\delta)}$ or not,

$$
\chi^v(i_{\mathrm{cl}(\delta)}) = \begin{cases} 1 & \text{if } i^v_{\mathrm{cl}(\delta)} = i_{\mathrm{cl}(\delta)} \\ 0 & \text{otherwise.} \end{cases}
$$

In order to obtain efficient implementations of the EM-algorithm, our objective is to calculate $P(i_{\mathrm{cl}(\delta)}|i^{(v)}_{b^{(v)}})$ in an efficient way. This is done by using the Hugin architecture, [Jensen, 1996], which operates on a junction tree J with cliques C corresponding to the Bayesian network. First, $i^{(v)}_{b^{(v)}}$ is inserted as evidence into the network and propagated. Now, a clique $c \in C$ is chosen such that $\mathrm{cl}(\delta) \subseteq c$. By [Jensen, 1996, Theorem 4.8] we know that after a full round of propagation the potential on c fulfills $\phi_c = P(c, i^{(v)}_{b^{(v)}})$, hereby

$$
P(i_{\mathrm{cl}(\delta)}|i^{(v)}_{b^{(v)}}) = \sum_{c \setminus \mathrm{cl}(\delta)} \frac{P(c, i^{(v)}_{b^{(v)}})}{P(i^{(v)}_{b^{(v)}})}.
$$

Thus, we have calculated all the terms necessary for $Q(P'|P)$.

The M-step is now identified by observing that the Q function corresponds to the likelihood function in (5.5), just with the observed data replaced by the estimated. Standard theory about MLE in Bayesian networks with complete data given by [Lauritzen, 1996, Theorem 4.36] yields that Q is maximized by updating the table for each $v \in \Delta$ by

$$
P(i_v|i_{\mathrm{pa}(v)}) = \frac{n^*(i_{\mathrm{cl}(v)})}{n^*(i_{\mathrm{pa}(v)})},
\tag{5.8}
$$

where $n^*(i_\emptyset) = N$ for variables with no parents. Notice how this formula corresponds to simple frequency counting in situations with complete observations.

Until now we have let $\boldsymbol{\theta} = (\theta_{i_{\mathrm{cl}(\delta)}}; \delta \in \Delta \wedge i_{\mathrm{cl}(\delta)} \in \mathcal{I}_{\mathrm{cl}(\delta)})$, i.e. each cell of each conditional probability table in the network is represented by a parameter. If we have additional information making an entire table $P(v|\,\mathrm{pa}(v))$ known and fixed, the M-step has to be modified.

From (5.4) it can be seen that the likelihood of such a restricted model, where one or more tables are fixed, is proportional to the likelihood of the unrestricted model. Let $F \subset \Delta$ be the set of nodes with fixed tables. Thus, $L(p) \propto \prod_{\delta \in \Delta \setminus F} L_\delta(p)$. Repeating the earlier calculations for this expression yields an updating scheme as (5.8), but this time updating is only performed for variables $v \in \Delta \setminus F$, the remaining nodes are left unaffected. With the EM-algorithm all cells in a conditional probability table must either be fixed or parametric, i.e. no mixture is allowed.

The behavior of the EM-algorithm is monitored by considering the value of the log-likelihood, $\log P(\mathbf{d}|\boldsymbol{\theta}) = \sum_{v=1}^{M} c_v \cdot \log P(i_{b^{(v)}}^{(v)})$. The EM-algorithm is stopped when the difference $|\log P(\mathbf{d}|\boldsymbol{\theta}^n) - \log P(\mathbf{d}|\boldsymbol{\theta}^{n+1})|$ is below some chosen threshold.

5.2.2 Maximum a Posteriori Estimation

Often we are not completely ignorant about the possible values of the parameters, e.g. this could be an idea about a possible interval or an experts opinion. Such knowledge reduces the number of cases necessary in **d** before proper values are attained. In a Bayesian framework $\boldsymbol{\theta}$ is regarded as a random vector and the prior information is translated into a prior distribution of $\boldsymbol{\theta}$. The global and local independence properties of the parameters, see [Thiesson, 1996], imply that the distribution of the parameters factorizes according to

$$\pi(\boldsymbol{\theta}) = \prod_{\delta \in \Delta} \prod_{\pi_\delta \in \mathcal{I}_{\mathrm{pa}(\delta)}} \pi(\boldsymbol{\theta}_{\delta|\pi_\delta}), \tag{5.9}$$

where $\boldsymbol{\theta}_{\delta|\pi_\delta} = (\theta_{i_\delta|\pi_\delta}; i_\delta \in \mathcal{I}_\delta)$ denotes the parameters in the parent configuration π_δ in $P(\delta|\,\mathrm{pa}(\delta))$. By (5.9) each local component, $\boldsymbol{\theta}_{\delta|\pi_\delta}$, of the distribution can be considered individually. It is common to assume $\boldsymbol{\theta}_{\delta|\pi_\delta} \sim \mathcal{D}(\alpha(i_\delta, \pi_\delta); i_\delta \in \mathcal{I}_\delta)$, where $\mathcal{D}(\cdot)$ denotes an $|\mathcal{I}_\delta|$-dimensional **Dirichlet distribution** with parameters $\alpha(i_\delta, \pi_\delta) > 0$. The probability density function of $\boldsymbol{\theta}_{\delta|\pi_\delta}$ is given by

$$f(\boldsymbol{\theta}_{\delta|\pi_\delta}) = \frac{\Gamma(\alpha(\pi_\delta))}{\prod_{i_\delta \in \mathcal{I}_\delta} \Gamma(\alpha(i_\delta, \pi_\delta))} \prod_{i_\delta \in \mathcal{I}_\delta} P(i_\delta|\pi_\delta)^{\alpha(i_\delta, \pi_\delta)-1}, \tag{5.10}$$

where $\Gamma(\cdot)$ is the gamma function as defined in e.g. [DeGroot, 1985, p. 286], $\alpha(\pi_\delta) = \sum_{i_\delta \in \mathcal{I}_\delta} \alpha(i_\delta, \pi_\delta)$ is called the **virtual sample size**, and $P(i_\delta|\pi_\delta) = \theta_{i_\delta|\pi_\delta}$. Notice that a 2-dimensional Dirichlet distribution corresponds to a Beta-distribution. In principle any continuous distribution could be chosen for the prior, but the Dirichlet distribution has the advantage that it is a **conjugate family of prior distributions** for multinomial sampling, i.e. a Dirichlet prior results in Dirichlet posterior, see [Heckerman *et al.*, 1994] and [DeGroot, 1985, Section 6.3]. This facilitates the method for use in adaptation as described in [Olesen *et al.*, 1992].

Thus, (5.10) yields

$$\pi(\boldsymbol{\theta}_{\delta|\pi_\delta}) \propto \prod_{i_\delta \in \mathcal{I}_\delta} (\theta_{i_\delta|\pi_\delta})^{\alpha(i_\delta, \pi_\delta)-1}.$$

This enables (5.3) to be rewritten using (5.6) and (5.9),

$$\begin{aligned}
Q^*(\boldsymbol{\theta}|\boldsymbol{\theta}^n, \mathbf{d}) &= Q(\boldsymbol{\theta}|\boldsymbol{\theta}^n, \mathbf{d}) + \log \pi(\boldsymbol{\theta}) \\
&= \sum_{\delta \in \Delta} \sum_{i_{\mathrm{pa}(\delta)} \in \mathcal{I}_{\mathrm{pa}(\delta)}} \Bigg[\sum_{i_\delta \in \mathcal{I}_\delta} n^*(i_\delta, i_{\mathrm{pa}(\delta)}) \log P'(i_\delta | i_{\mathrm{pa}(\delta)}) + \\
&\qquad \sum_{i_\delta \in \mathcal{I}_\delta} (\alpha(i_\delta, i_{\mathrm{pa}(\delta)}) - 1) \log P'(i_\delta | i_{\mathrm{pa}(\delta)}) \Bigg] \\
&= \sum_{\delta \in \Delta} \sum_{i_{\mathrm{cl}(\delta)} \in \mathcal{I}_{\mathrm{cl}(\delta)}} \left[(n^*(i_{\mathrm{cl}(\delta)}) + \alpha(i_\delta, i_{\mathrm{pa}(\delta)}) - 1) \log P'(i_\delta | i_{\mathrm{pa}(\delta)}) \right]
\end{aligned}$$

This identifies the E-step in MAPE. Observe how the parameters of the Dirichlet distribution are added to the estimated number of counts. Therefore, the $\alpha(i_\delta, \pi_\delta)$ can be interpreted as some kind of prior observed counts expressing our prior knowledge. If we are quite certain about the value of one of the parameters this will be translated to a large number of prior counts and it will take a lot of observed cases to change the value of the parameter.

To derive an expression for the M-step we note that the above corresponds to the likelihood function with $n(i_{\mathrm{cl}(\delta)})$ replaced by $n^*(i_{\mathrm{cl}(\delta)}) + \alpha(i_\delta, \pi_\delta) - 1$. Since the model sets no further restrictions, we can again use [Lauritzen, 1996, Theorem 4.36] about maximum likelihood estimation in Bayesian networks based on multinomial sampling. Hereby, the M-step is given by

$$P(i_v | i_{\mathrm{pa}(v)}) = \frac{n^*(i_{\mathrm{cl}(v)}) + \alpha(i_v, i_{\mathrm{pa}(v)}) - 1}{n^*(i_{\mathrm{pa}(v)}) + \alpha(i_{\mathrm{pa}(v)}) - |\mathcal{I}_v|}. \tag{5.11}$$

A problem occurs in the above expression if the number of prior counts is less than one. Then, the expression can turn negative which indicates that the posterior cannot be maximized within $[0, 1]$. The problem is in [Lauritzen, 1991] solved by simply ignoring the subtractions; thus, considering $\alpha(i_\delta, \pi_\delta)$ as regular counts. Hence,

$$P(i_v | i_{\mathrm{pa}(v)}) = \frac{n^*(i_{\mathrm{cl}(v)}) + \alpha(i_v, i_{\mathrm{pa}(v)})}{n^*(i_{\mathrm{pa}(v)}) + \alpha(i_{\mathrm{pa}(v)})}. \tag{5.12}$$

In [Thiesson, 1996, pp. 76–78] this expression is obtained by using an exponential representation for the parameters. For our use we adopt (5.12) and observe that an **non-informative prior** in this update scheme is given by $\alpha(i_\delta, \pi_\delta) \equiv 0$.

Specification of a prior distribution of the parameters can also be used to qualify the discussion about which start value to use for $\boldsymbol{\theta}$. Two approaches exist: the parameter values of each $\boldsymbol{\theta}_{\delta|\pi_\delta}$ can be sampled from the probability density function of $\boldsymbol{\theta}_{\delta|\pi_\delta}$, or the starting value is simply chosen to be $\mathbb{E}[\boldsymbol{\theta}_{\delta|\pi_\delta}]$. In case of a uniform prior this means $\mathbb{E}[\boldsymbol{\theta}_{i_\delta|\pi_\delta}] = 1/|\mathcal{I}_\delta|$ otherwise we have $\mathbb{E}[\boldsymbol{\theta}_{i_\delta|\pi_\delta}] = \alpha(i_\delta, \pi_\delta)/\alpha(\pi_\delta)$

5.2.3 Deriving the Parameters of the Dirichlet distribution

The preceding section assumed that we could express our prior knowledge of the parameters for a specific parent configuration by a Dirichlet distribution with parameters $\alpha(i_\delta, \pi_\delta)$. The problem is to choose $\alpha(i_\delta|\pi_\delta)$ such that the prior distribution expresses our prior knowledge correctly. We use the method suggested by [Thiesson, 1996, p. 65], [Heckerman *et al.*, 1994] and [Olesen *et al.*, 1992], where the parameters are derived by specifying individual mean and variance for each random variable. The prior information for a parameter is now given on the form

$$\mathbb{E}[P(i_v|\pi_v)] \pm \sigma(i_v|\pi_v),$$

i.e. a best guess, $\mathbb{E}[P(i_v|\pi_v)]$, and an interval of possible variation, $\sigma(i_v|\pi_v)$. For a specific parent configuration the best guesses for the individual states must sum to 1. The above specification is quite intuitive and can thus be used to query experts for prior knowledge.

The individual mean and variance of each variable in an $|\mathcal{I}_v|$-dimensional Dirichlet distribution are given by

$$\mathbb{E}[P(i_v|\pi_v)] = \frac{\alpha(i_v, \pi_v)}{\alpha(\pi_v)} \tag{5.13}$$

$$\mathbb{V}[P(i_v|\pi_v)] = \frac{(\alpha(\pi_v) - \alpha(i_v, \pi_v))\alpha(i_v, \pi_v)}{\alpha(\pi_v)^2(\alpha(\pi_v) + 1)} \tag{5.14}$$

The variance of a parameter $P(i_v|\pi_v)$ is also called the user's confidence in the mean for the parameter. In this terminology $\alpha(\pi_v)$ can be interpreted as an virtual sample size, i.e. the number of observations we would have had to make before we had the same confidence about the parameter as provided by our prior knowledge.

Combining (5.13) and (5.14) gives an expression for the virtual sample size for each i_v.

$$\alpha^{i_v}(\pi_v) = \frac{(1 - \mathbb{E}[P(i_v|\pi_v)])\mathbb{E}[P(i_v|\pi_v)]}{\sigma(i_v|\pi_v)^2} - 1.$$

If $\sigma(i_v|\pi_v)^2$ is very large, the above can become negative. This is regarded as non informative prior information and is expressed by setting $\alpha^{i_v}(\pi_v) = 0$. If two $\alpha^{i_v}(\pi_v)$ are not equal for different i_v, this means that it takes a different amount of observations for the two parameters before confidence by observation is equal to confidence by prior knowledge. Fundamentally seen, this is inconsistent since we see the parameters for a specific parent configuration to be "equally" uncertain, e.g. for a binary variable it does not make sense to be very confident in $P(i_1|\pi_v)$ while being less confident in $P(i_2|\pi_v)$. An exception occurs when we, for a given parent configuration, know that one or more states of v are impossible, i.e. $P(i_v|\pi_v) = 0$ which implies $\sigma(i_v|\pi_v) = 0$. For now, we do not allow this situation.

To get a consistent specification for $P(\mathcal{I}_v|\pi_v)$ it is thus required that $\alpha^{i_v}(\pi_v)$ is the same for all $i_v \in \mathcal{I}_v$. We translate the phrase "a chain is not stronger than its weakest link" to confidence for parameter values. Therefore, in the case of inconsistency on the $\alpha^{i_v}(\pi_v)$, we choose the smallest virtual sample size

$$\alpha(\pi_v) \quad = \quad \min\left\{\alpha^{i_v}(\pi_{i_v}) \mid i_v \in \mathcal{I}_v\right\}. \tag{5.15}$$

Example 5.1

Consider $P(\mathcal{I}|\pi_v) = (0.2[0.1 - 0.3], 0.8[0.75 - 0.85])$ we compute $\alpha^{i_1}(\pi_v) = 15$ and $\alpha^{i_2}(\pi_v) = 63$. Hence by (5.15), $\alpha(\pi_v) = 15$. Now, (5.13) yields $\alpha(i_1, \pi_v) = 3$ and $\alpha(i_2, \pi_v) = 12$. The prior Beta distribution for $P(i_1|\pi_v)$ with the calculated parameters is shown in Figure 5.2.

$\diamond$

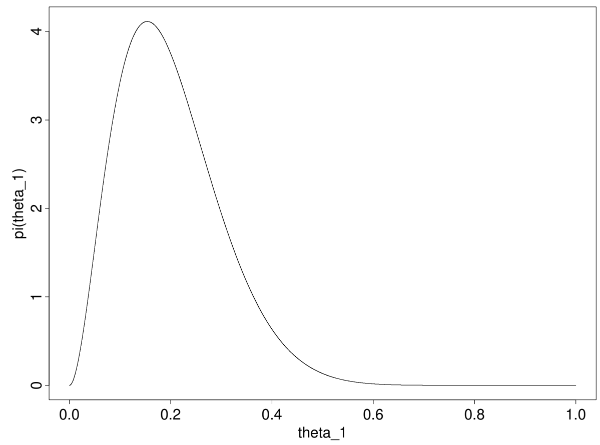

Figure 5.2: The prior distribution of $P(i_1|\pi_v)$, where $\pi(P(i_1|\pi_v)) \sim \mathcal{B}(3, 12)$.

The EM-algorithm is known to converge very slowly if high accuracy of the stationary point is desired in MLE or MAPE, [Thiesson, 1996, Article 5]. A possible way to increase the speed of convergence is to accelerate the EM-algorithm with a **conjugate gradient method**, [Thiesson, 1996, Article 5] and [Press *et al.*, 1992, Section 10.6] contain further information.

Empirical investigations performed by [Thiesson, 1991] show that in situations with data missing massively, e.g. due to many hidden variables, the likelihood function has many local extrema. Hereby, the EM-algorithm for MLE can become trapped in a local maximum close to the starting point. This maximum will most likely be sub-optimal. MAPE with reasonable priors seems to perform much better in these situations.

Our aim is now to investigate how gradient descent based neural network type training performs on classification problems compared to MAPE. But, before we can discuss neural network type training we need to describe gradient-descent learning in general.

5.3 Gradient-Descent Learning of Bayesian Networks

In this section we will discuss gradient-descent learning algorithms to be used for the learning of Bayesian networks, especially suited for classification problems. This section is mainly based upon the theory and notation of [Jensen, 1999] and [Russel *et al.*, 1995].

The simplest gradient-descent algorithm is to compute the gradient vector of some kind of metric function $f(\boldsymbol{\theta})$ with respect to the unknown parameters, i.e. $\boldsymbol{\nabla}_{\boldsymbol{\theta}} = (\frac{\partial f}{\partial \theta_1} \cdots \frac{\partial f}{\partial \theta_n})$. Then, the parameters take a little step in the direction of the gradient (or in the opposite direction, depending whether the metric function should be maximized or minimized),

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} \pm \delta \boldsymbol{\nabla}_{\boldsymbol{\theta}},$$

where δ is the **damping** of the step, also known as the **step size**.

Different approaches can be taken with respect to choosing a metric function. The function can either be a distance measure between actual and desired probabilities [Jensen, 1999], which should be minimized, or the likelihood of the data cases [Russel *et al.*, 1995], $P(\mathbf{d})$, which should be maximized. The distance measure proves beneficial when performing neural network type training. In the following subsections we elaborate on performing gradient descent with the distance measure and the likelihood of the data cases, this is is followed by a description on how to update the unknown parameters with some kind of damping. Finally, an extension of the gradient-descent method by generalized parameters is considered.

5.3.1 The Distance Measure

Let $B = (G, P)$ be the Bayesian network to be learned, where G is the underlying graph of B and P is probability distribution of the network. For simplicity, assume we only have one data case, d. For a variable $A \in V(G)$, let $\mathbf{z} = P_\theta(A|d) = (z_1(\boldsymbol{\theta}), \ldots, z_l(\boldsymbol{\theta}))$ be the probability calculated by the network with $l = |\mathcal{I}_A|$. Furthermore, assume that $\mathbf{y} = (y_1, \ldots, y_l)$ is a prior request for $P(A|d)$. This prior request could be an expert's opinion or be derived from frequency counting in a database. The latter is discussed in section 5.4. The task is now to tune the network such that $P_\theta(A|d) = \mathbf{y}$.

To tune the network, we introduce the distance measure, $\mathrm{dist}(\mathbf{z}(\boldsymbol{\theta}), \mathbf{y})$. The two most frequently used distance measures are the *Euclidean distance*

$$\mathrm{dist}_E(\mathbf{z}(\boldsymbol{\theta}), \mathbf{y}) = \sum_{i=1,\ldots,l} (z_i(\boldsymbol{\theta}) - y_i)^2, \tag{5.16}$$

and the *Kullback-Leibler distance*

$$\mathrm{dist}_K(\mathbf{z}(\boldsymbol{\theta}), \mathbf{y}) = \sum_{i=1,\ldots,l} y_i(\log y_i - \log z_i(\boldsymbol{\theta})). \tag{5.17}$$

Both metrics are so called **strictly proper scoring rules**, i.e. if dist $= 0$ then $\mathbf{z}(\boldsymbol{\theta}) = \mathbf{y}$. Kullback-Leibler is not applicable for gradient descent training in our sleep network because here situations where $\exists i : y_i = 0$ occur. Only the Euclidean distance measure will therefore be discussed in this thesis. Refer to [Jensen, 1999] for further calculations with the Kullback-Leibler distance measure.

Tuning the unknown parameters such that $P_{\boldsymbol{\theta}}(A|d) = \mathbf{y}$ is the same as tuning the parameters such that $\text{dist}_E(\mathbf{z}(\boldsymbol{\theta}), \mathbf{y})$ is as close to 0 as possible, meaning that the task is to minimize the dist_E function, [Jensen, 1999]. Thus, the gradient-descent algorithm is:

Algorithm 5.2

1. Calculate the gradient, $\boldsymbol{\nabla_\theta} = \boldsymbol{\nabla}\, \text{dist}_E(\mathbf{z}(\boldsymbol{\theta}), \mathbf{y})$, with respect to the parameters, $\boldsymbol{\theta}$.
2. Adjust $\boldsymbol{\theta}$ in the direction opposite to $\boldsymbol{\nabla_\theta}$.
3. Repeat until $|\boldsymbol{\nabla_\theta}|$ is below some threshold close to zero. An alternative condition is to stop once the difference between two successive distance values gets below some threshold.

Calculating the gradient is now done by calculating each partial derivative independently. From (5.16) we get

$$\frac{\partial}{\partial \theta_j} \text{dist}_E(\mathbf{z}(\boldsymbol{\theta}), \mathbf{y}) = \sum_{i=1,\dots,l} 2(z_i(\theta_j) - y_i) \frac{\partial z_i(\theta_j)}{\partial \theta_j}, \tag{5.18}$$

where $z_i(\theta_j)$ is an abbreviation for $z_i(\boldsymbol{\theta})$ with $\{\theta_i | i \neq j\}$ kept fixed at their current values. Note, that the expression for $z_i(\theta_j)$ is available through propagation in B. Gathering the individual components, (5.18) yields

$$\boldsymbol{\nabla_\theta}\, \text{dist}_E(\mathbf{z}(\boldsymbol{\theta}), \mathbf{y}) = \sum_{i=1,\dots,l} 2(\mathbf{z}(\boldsymbol{\theta}) - \mathbf{y}) \boldsymbol{\nabla_\theta} \mathbf{z}(\boldsymbol{\theta}). \tag{5.19}$$

Hence, the gradient $\boldsymbol{\nabla_\theta}$ is $\mathbf{0}$ if and only if either $\mathbf{z}(\boldsymbol{\theta})$ is independent of $\boldsymbol{\theta}$ or $\mathbf{z}(\boldsymbol{\theta}) = \mathbf{y}$.

Now, we only need to show how to determine $P_{\boldsymbol{\theta}}(A|d) = \mathbf{z}(\boldsymbol{\theta})$. Let $\theta_{ijk} \equiv P(X_i = x_{ij} | \pi(X_i) = \pi_{ik})$, i.e. θ_{ijk} is the jth entry of the conditional probability table of the variable X_i where the parent configuration is π_{ik}. With this specification, $\mathbf{z}(\boldsymbol{\theta})$, and hence also dist_E, is considered an unrestricted function in θ. Since the parameters are entries of conditional probability tables, the function is in reality subject to the constraints

$$\forall \theta_{ijk} \in \boldsymbol{\theta} \;:\; 0 \leq \theta_{ijk} \leq 1, \text{ and} \tag{5.20}$$

$$\forall i, k \;:\; \sum_j \theta_{ijk} = 1. \tag{5.21}$$

Translating from the unconstrained function to the constrained function

Our actual goal is to minimize the dist_E function subject to the constraints (5.20) and (5.21). Two different approaches exist: the first approach calculates the gradient of the unconstrained function which is then projected onto the constrained surface, the second approach deals with the constraints by re-parameterizing such that the new parameters automatically fulfill the constraints. The latter method is mainly used in statistical approaches, e.g. modeling with local exponential models as in [Thiesson, 1996]. We decide only to deal with the projection method because re-parameterization can, by use of the chain rule, also be performed within the projection approach.

The gradient projection approach is in detail described in [Binder *et al.*, 1997]. Here, the gradient of the constrained function, $\boldsymbol{\nabla_\theta}$, is obtained by performing an orthogonal projection of the unconstrained gradient $\boldsymbol{\nabla}'_{\boldsymbol{\theta}}$, see Figure 5.3(a), onto the constrained surface. To perform an orthogonal projection, the projection of $\boldsymbol{\nabla}'_{\boldsymbol{\theta}}$ onto a vector $\mathbf{n}$ orthonormal to the constrained surface is subtracted from $\boldsymbol{\nabla}'_{\boldsymbol{\theta}}$. Hence,

$$\boldsymbol{\nabla_\theta} = \boldsymbol{\nabla}'_{\boldsymbol{\theta}} - (\boldsymbol{\nabla}'_{\boldsymbol{\theta}} \cdot \mathbf{n})\mathbf{n} \tag{5.22}$$

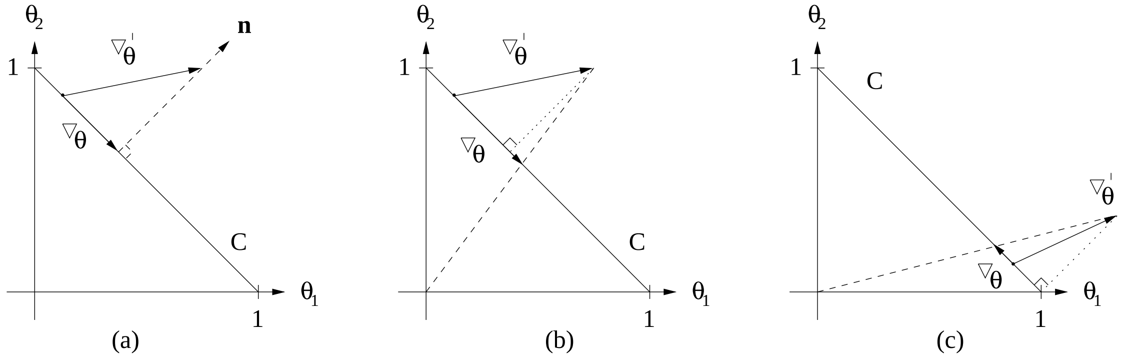

Figure 5.3: (a) Orthogonal projection of the unconstrained gradient $\boldsymbol{\nabla_\theta}'$ onto the constrained surface C in a situation with $\boldsymbol{\theta} = (\theta_1, \theta_2)$. (b) Shows the unconstrained gradient from (a) being projected by our normalization approach. (c) Illustrates a situation where our normalization approach makes the constrained gradient point in the wrong direction.

We first consider the constraint that each row in a conditional probability table, $T_i = P(X_i|\,\mathrm{pa}(X_i))$, has to sum to one. By assumption the parameters of a column in T_i are independent of parameters in other columns or other tables. Thus, the orthogonal projection can be decomposed by adding the results of performing local orthogonal projections for each column.

For one column, c, in T_i we have $\boldsymbol{\theta}_c = (\theta_1, \ldots, \theta_k)$, where $k = |\mathcal{I}_{X_i}|$. A local projection is given by only considering the $\boldsymbol{\theta}_c$ component of $\boldsymbol{\nabla}'_{\boldsymbol{\theta}}$, i.e. an orthogonal projection of $\boldsymbol{\nabla}'_{\boldsymbol{\theta}} \in \Theta$ onto $\boldsymbol{\nabla}'_{\boldsymbol{\theta}}(c) \in \Theta_c$.

In this local vectorspace, a vector, $\mathbf{n}_c$, orthonormal to the constrained surface is found by the procedure given in [Fraleigh and Beauregard, 1990, p. 278], this yields

$$\mathbf{n}_c = (\frac{1}{\sqrt{k}}, \ldots, \frac{1}{\sqrt{k}}).$$

Combining this expression with (5.22) yields

$$\boldsymbol{\nabla}_{\boldsymbol{\theta}}(c) = (c_1, \ldots, c_k) - \left(\frac{1}{k}\sum_{j=1}^{k} c_j\right)(1, \ldots, 1), \tag{5.23}$$

where $(c_1, \ldots, c_k)$ are the coordinates of $\boldsymbol{\nabla}'_{\boldsymbol{\theta}}(c)$. Since the sum of all coordinates of $\boldsymbol{\nabla}_{\boldsymbol{\theta}}(c)$ is zero, advancing the parameters by $\boldsymbol{\nabla}_{\boldsymbol{\theta}}(c)$ does not change the column sum of c. That is, if we start on the constrained surface we will stay on it.

Unfortunately, we became aware of [Binder *et al.*, 1997] at a too late stage, instead our implementation uses an ad-hoc projection based on our understanding of [Russel *et al.*, 1995]. Here, we simply advance the parameters by the unconstrained gradient and then renormalize by performing a normalization within each column containing parameters, see Figure 5.3(b). In most cases our normalization approach gives a direction parallel to what would have been obtained by the gradient projection approach, but in other situations we actually move in the wrong direction, see Figure 5.3(c). Fortunately, our procedure performs quite well in practice because it adds some random movement which allows us to possibly escape sub-optimal minima. Further investigations need to be performed to investigate the influence on the convergence properties.

To enforce the constraint that each parameter must be between 0 and 1, we apply a simple method investigating whether advancing the parameters according to the projected parameter leads to negative values. If yes, we increase the overall damping. As long as there are no negative values our renormalization procedure ensures that the values do not exceed 1.

Calculating the unconstrained gradient of the distance function

We now return to the task of calculating the unconstrained gradient of the distance measure. Since $P_{\boldsymbol{\theta}}(A|d) = \frac{P_{\boldsymbol{\theta}}(A,d)}{P_{\boldsymbol{\theta}}(d)}$, then

$$\begin{aligned}
\frac{\partial P_{\boldsymbol{\theta}}(A|d)}{\partial \theta_{ijk}} &= \frac{1}{P_{\boldsymbol{\theta}}(d)^2}\left(\frac{\partial P_{\boldsymbol{\theta}}(A,d)}{\partial \theta_{ijk}} P_{\boldsymbol{\theta}}(d) - P_{\boldsymbol{\theta}}(A,d)\frac{\partial P_{\boldsymbol{\theta}}(d)}{\partial \theta_{ijk}}\right) \\
&= \frac{1}{P_{\boldsymbol{\theta}}(d)}\left(\frac{\partial P_{\boldsymbol{\theta}}(A,d)}{\partial \theta_{ijk}} - P_{\boldsymbol{\theta}}(A|d)\frac{\partial P_{\boldsymbol{\theta}}(d)}{\partial \theta_{ijk}}\right)
\end{aligned} \tag{5.24}$$

Here, we require $P_{\boldsymbol{\theta}}(A,d) > 0$, otherwise the model claims that the probability of observing a case is zero despite that we have have actually observed it. In this situation the parameters are inconsistent with the data which we consider to be a modeling error.

It has been proven that both $P(A,d)$ and $P(d)$ are linear functions in θ_{ijk}, [Russel *et al.*, 1995], and [Castillo *et al.*, 1997]. Considering $P(d)$ from (5.24) the linearity can be seen from

$$\begin{aligned} \frac{\partial P_{\boldsymbol{\theta}}(d)}{\partial \theta_{ijk}} &= \frac{\partial \left(\sum_{j',k'} P_{\boldsymbol{\theta}}(d|x_{ij'}, \pi_{ik'}) P_{\boldsymbol{\theta}}(x_{ij'}, \pi_{ik'}) \right)}{\partial \theta_{ijk}} \\ &= \frac{\partial \left(\sum_{j',k'} P_{\boldsymbol{\theta}}(d|x_{ij'}, \pi_{ik'}) P_{\boldsymbol{\theta}}(x_{ij'}|\pi_{ik'}) P_{\boldsymbol{\theta}}(\pi_{ik'}) \right)}{\partial \theta_{ijk}} \end{aligned} \tag{5.25}$$

where the term $\theta_{ijk} = P_{\boldsymbol{\theta}}(x_{ij'}|\pi_{ik'})$ appears only once in the summation, namely for $j' = j, k' = k$. Thus, it can be concluded that $P_{\boldsymbol{\theta}}(d)$ is linear in $\boldsymbol{\theta}$.

Since we are interested in the derivatives of $P(A, d)$ and $P(d)$, it is enough to compute the coefficient for the θ_{ijk} term of each linear function. The straightforward method to find these coefficients is to first insert d as evidence, propagate, and then store the normalization constant (the likelihood of the evidence, here d). Next, we change the unknown parameter to θ'_{ijk}, and do the same. Then we can set up two equations with two unknowns:

$$\begin{aligned} \alpha\theta_{ijk} + \beta &= P_{\theta_{ijk}}(d) \\ \alpha\theta'_{ijk} + \beta &= P_{\theta'_{ijk}}(d) \end{aligned}$$

Hence, $\frac{\partial P(d)}{\partial \theta_{ijk}} = \alpha$ in (5.24). And the same is done for $\frac{\partial P(A,d)}{\partial \theta_{ijk}}$ where the virtual evidence $A = a$ is inserted as further evidence.

This approach is rather time consuming since it is necessary for each distinct data case to propagate for each unknown parameter; therefore, if the network or the number of parameters is large, the computation will take intractable long time. But, it is possible to do the computations locally, needing only two propagations per data case, after which the coefficients for each parameter can be computed locally. Finding the derivative in (5.25) with respect to θ_{ijk} corresponds to finding the derivative when $j' = j$ and $k' = k$, i.e. finding

$$\frac{\partial P_{\boldsymbol{\theta}}(d|x_{ij}, \pi_{ik}) \theta_{ijk} P_{\boldsymbol{\theta}}(\pi_{ik})}{\partial \theta_{ijk}}.$$

Since the term θ_{ijk} is not contained in the summation of (5.25) for all other values of j' and k', making the derivatives of all the other terms of the summation zero. This yields

$$\begin{aligned} \frac{\partial P_{\boldsymbol{\theta}}(d)}{\partial \theta_{ijk}} &= P_{\boldsymbol{\theta}}(d|x_{ij}, \pi_{ik}) P_{\boldsymbol{\theta}}(\pi_{ik}) \\ &= \frac{P_{\boldsymbol{\theta}}(x_{ij}, \pi_{ik}|d) P_{\boldsymbol{\theta}}(d) P_{\boldsymbol{\theta}}(\pi_{ik})}{P_{\boldsymbol{\theta}}(x_{ij}, \pi_{ik})} \\ &= \frac{P_{\boldsymbol{\theta}}(x_{ij}, \pi_{ik}|d) P_{\boldsymbol{\theta}}(d)}{P_{\boldsymbol{\theta}}(x_{ij}|\pi_{ik})} \\ &= \frac{P_{\boldsymbol{\theta}}(x_{ij}, \pi_{ik}|d) P_{\boldsymbol{\theta}}(d)}{\theta_{ijk}} \end{aligned} \tag{5.26}$$

And similarly we have that

$$\frac{\partial P_{\boldsymbol{\theta}}(A,d)}{\partial \theta_{ijk}} = \frac{P_{\boldsymbol{\theta}}(x_{ij},\pi_{ik}|A,d)P_{\boldsymbol{\theta}}(A,d)}{\theta_{ijk}} \tag{5.27}$$

The result in (5.26) and (5.27) can be calculated directly from by-products of the Hugin propagation scheme. The θ_{ijk} term is simply an entry in the conditional probability table of the variable X_i, and $P_{\boldsymbol{\theta}}(d)$ is the normalization constant calculated by the propagation method in Hugin. But, the $P_{\boldsymbol{\theta}}(x_{ij},\pi_{ik}|d)$ has to be found in a clique potential hosting the variables $X_i \cup \pi(X_i)$, e.g. the clique to which X_i is assigned at the compilation of the Bayesian network.

In situations where $\theta_{ijk} = 0$ the above formulae cannot be used. Here, the approach with inserting two values has to be used, i.e. using two different instances of the table. With lazy or Shafer-Shenoy propagation we have all the relevant functions stored for controlling multiplication on our own, furthermore the zero value of the parameter has not been used in any of the functions necessary for multiplication. Therefore, it is easy to replace the required table instances, see [Jensen, 1999]. Unfortunately, this is not the case for the Hugin architecture, where the zero value could have been distributed to other cliques influencing the messages they send back to the clique, where we want to exchange the table.

5.3.2 The Likelihood of the Data Cases Measure

Using the likelihood of the data cases, $P(\mathbf{d})$, as a metric for the gradient-descent algorithm, $P_{\boldsymbol{\theta}}(\mathbf{d})$ must be maximized with respect to the unknown parameters specified by $\boldsymbol{\theta}$. This means that the unknown parameters must be tuned such that $P_{\boldsymbol{\theta}}(\mathbf{d})$ is as close as possible to 1, or at least as large as possible.

The approach is a little different from the distance method, but as before we can calculate each contribution to the gradient from each data case separately, we use the log-likelihood, $\log P_{\boldsymbol{\theta}}(\mathbf{d})$, to replace multiplications by summations which eases differentiation, and recall that $\theta_{ijk} \equiv P(X_i = x_{ij}|\pi(X_i) = \pi_{ik})$. Using the chain rule and simple calculus we get

$$\begin{aligned}
\frac{\partial \log P_{\boldsymbol{\theta}}(\mathbf{d})}{\partial \theta_{ijk}} &= \frac{\partial \log \prod_{l=1}^{N} P_{\boldsymbol{\theta}}(d_l)}{\partial \theta_{ijk}} \\
&= \sum_{l=1}^{N} \frac{\partial \log P_{\boldsymbol{\theta}}(d_l)}{\partial \theta_{ijk}} \\
&= \sum_{l=1}^{N} \frac{\partial P_{\boldsymbol{\theta}}(d_l)}{\partial \theta_{ijk}} \frac{1}{P_{\boldsymbol{\theta}}(d_l)} \qquad (5.28)
\end{aligned}$$

As before we need an efficient way to calculate (5.28). Rewriting (5.28) by (5.26) yields

$$\frac{\partial \log P_{\boldsymbol{\theta}}(\mathbf{d})}{\partial \theta_{ijk}} = \sum_{l=1}^{N} \frac{P_{\boldsymbol{\theta}}(x_{ij},\pi_{ik}|d)}{\theta_{ijk}} \tag{5.29}$$

As previously these computations happen locally in the clique tables. Again, the gradient of $P_{\boldsymbol{\theta}}(\mathbf{d})$ is a vector of all the expressions of (5.29) with respect to the parameters, $\boldsymbol{\theta}$. By using the likelihood as metric function we are using the same evaluation function as the EM-algorithm. This enables switching between the method described in this chapter and the the EM algorithm for MLE to speed up convergence, [Thiesson, 1996]. The problem is that the likelihood is a very general metric, which does not incorporate any knowledge about the nature of the learning problem, e.g. a classification.

5.3.3 Damping of the Gradient Step

The gradient descent methods presented in the previous sections were all concerned with calculating the gradient and then loyally move all the parameters in the direction of the gradient. If we are more certain about the value of one parameter, and very uncertain about the value of another, we would prefer that the value of the secure parameter does not change much from its starting value, whereas the insecure value is allowed to change much. This can be accomplished by replacing the overall *damping* δ by individual damping for each parameter. This idea originates from [Jensen, 1999] but is in the following further developed for our needs.

In our terminology **damping** is about reducing the speed of parameter advancement, i.e. by decreasing the size of the gradient. If the network contains n parameters we have $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_n)$ with $\forall \delta_i : \delta_i \geq 0$ and an update step of

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} \pm D_{\boldsymbol{\delta}} \boldsymbol{\theta},$$

where $D_{\boldsymbol{\delta}}$ is a $n \times n$ diagonal matrix with diagonal elements equal to $\boldsymbol{\delta}$. By using individual damping coefficients, we are not moving in a direction parallel to the gradient anymore, but only in a direction pointing the same "way" as the gradient, i.e. only the sign of the gradient's coordinates is preserved.

One should differ between two types of damping. The first type is used to prevent the parameter from attending undesirable values, e.g. values outside $[0, 1]$ or values which according to the prior probability density function of the parameters are very unlikely. The second type reflects resistance against changing the value because we are confident that the current value is close to the correct one. The two types may seem equivalent, but for us they differ by using different values to calculate the camping: Damping to avoid undesirable values uses the values to which we would have advanced, whereas damping due to resistance against moving, uses the current values. Otherwise the two methods are alike. Our task is now find a prudent way of quantifying $\boldsymbol{\delta}$ based on confidence in parameters.

We choose to construct a counterpart to the use of Dirichlet priors in the statistical framework of the EM-algorithm. For the EM-algorithm we had $\boldsymbol{\theta}_{v|\pi_v} \sim \mathcal{D}(\alpha(i_v, \pi_v); i_v \in \mathcal{I}_v)$. In the context of damping we translate the value of $\pi(\boldsymbol{\theta}_{v|\pi_v})$ into confidence about $\boldsymbol{\theta}_{v|\pi_v}$.

We use the following heuristic; the higher the value of $\pi(\boldsymbol{\theta}_{v|\pi_v})$ the higher the confidence, and therefore the higher the damping on the parameters $\boldsymbol{\theta}_{v|\pi_v}$. Since the variables within the parent configuration are to be seen simultaneously we apply the same damping $\delta_{v|\pi_v}$ to all $\boldsymbol{\theta}_{v|\pi_v}$. With this concept, damping is only interesting when there are parameters in more than one parent configuration. Otherwise, damping just corresponds to choosing a smaller overall step size. The question is now how to compute $\delta_{v|\pi_v}$ from $\pi(\boldsymbol{\theta}_{v|\pi_v})$, in a way that is consistent with parameters without a prior.

A non-informative prior for a parent configuration is given by $\alpha(i_v, \pi_v) = 1, \forall i_v \in \mathcal{I}_v$. Hence, for a parent configuration without prior knowledge we should have $\delta_{v|\pi_v} = \delta$. As a further constraint we do not allow the damping coefficient to exceed the overall damping δ. If the damping is above δ, we argue that the confidence is so low that the parameter should be damped as a parameter without priors. We choose the following heuristic computation formula

$$\delta_{v|\pi_v} = \delta \cdot \min\left\{ \left(f(\boldsymbol{\theta}_{v|\pi_v}) \cdot f(\boldsymbol{\theta}^0_{v|\pi_v}) \right)^{-1}, 1 \right\}, \tag{5.30}$$

where f is the probability density function of $\boldsymbol{\theta}_{v|\pi_v}$ given by (5.10), and $\boldsymbol{\theta}^0_{v|\pi_v}$ are the start values of $\boldsymbol{\theta}_{v|\pi_v}$, i.e. the vector of best guesses, $\mathbb{E}(\boldsymbol{\theta}_{v|\pi_v})$, as specified in Section 5.2.3. Multiplication by $f(\boldsymbol{\theta}^0_{v|\pi_v})$ is performed to make the values of the damping function disperse more between zero and one.

Example 5.3

For $P(i_v|\pi_v) = (0.2[0.1 - 0.3], 0.8[0.75 - 0.85])$ we calculated $\boldsymbol{\theta}_{v|\pi_v} \sim \mathcal{B}(3, 12)$. Figure 5.4 shows the probability density function and $\delta_{v|\pi_v}$ as a function of $\theta_{i_1|\pi_v}$.

$\diamond$

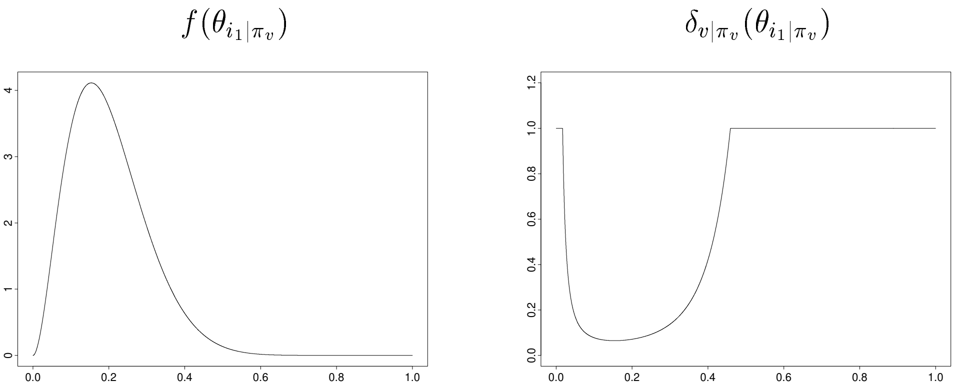

Figure 5.4: An example of our damping heuristic where $\pi(\theta_{i_1|\pi_v})$ is translated into $\delta_{v|\pi_v}$, with $\delta = 1$.

The advantage of using damping is the same as when using MAPE. With many parameters the metric function will have a large number of extrema, such that the gradient descent method quickly becomes trapped in an extrema which is likely to be lousy. By adding damping we guide the convergence in a preferred direction. Furthermore, the prior information should, just as for MAPE with the EM-algorithm, allow additional knowledge not represented in the database to be included. This should give better results when applying the learned network on cross-validation sets.

5.3.4 Extending the Gradient-Descent Methods with Generalized Parameters

The gradient-descent methods described above only apply to networks where no relations exist between the cells of the conditional probability tables. Without any relations the number of parameters can quickly reach intractable amounts. Often the network contains relations which reduce the amount of cells necessary for specifying the network, e.g. in noisy-or models. Also expanded template models (e.g. time slices models) or networks for pedigree analysis contain many identical (duplicated) tables. In order to keep this structural knowledge when considering the conditional probability table cells we now introduce *generalized parameters* for the gradient-descent methods.

Using generalized parameters we want the network to be defined from these generalized parameters, and as before we want to learn these parameters from a set of data cases, $\mathbf{d}$. The previously described algorithms remain unchanged; though, the gradient-descent algorithms are performed with respect to the generalized parameters instead.

Now, let $\boldsymbol{\lambda}$ be the vector of the generalized parameters which is the parameters that are to be tuned according to some metric. Then, each parameter can be viewed as a function of $\boldsymbol{\lambda}$, i.e. $\theta_{ijk}(\boldsymbol{\lambda})$. Assume these functions are differentiable, and by using the chain rule for derivatives we obtain,

$$\frac{\partial \log P_{\boldsymbol{\theta}}(\mathbf{d})}{\partial \lambda_m} = \sum_{i,j,k} \frac{\partial \log P_{\boldsymbol{\theta}}(\mathbf{d})}{\partial \theta_{ijk}} \frac{\partial \theta_{ijk}(\boldsymbol{\lambda})}{\partial \lambda_m}. \tag{5.31}$$

Previously in this section we have shown that the first term of (5.31) can be computed as a by-product from e.g. the Hugin inference engine. The second term requires only simple calculus, i.e. finding the partial derivative of $\theta_{ijk}(\boldsymbol{\lambda})$ with respect to λ_m.

Expansion of Templates Using Generalized Parameters

Expansion of time sliced templates needs generalized parameters because of the identical parameters that appear when copying the template. For example, in the sleep classifier specified in Figure 4.16 on page 43 a *PersonType* node is present which has a link to e.g. each *SleepStage* of each time slice, but *PersonType* itself only exists once in the expanded network. The *PersonType* node is called a **global node** because it is a parent to a node of every time slice template and only exists once in the expanded network.

In this situation the template has to be expanded, meaning that the template must be copied a number of times corresponding to the number of epochs of the recorded sleep in question. This means that the size of the expanded network varies for each sleep recording. An example of a part the expanded network is depicted in Figure 5.5.

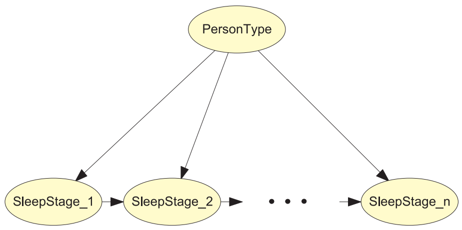

Figure 5.5: A part of an expanded network of the sleep classifier template.

From the figure it can be seen that the parameters in the conditional probability tables for all $SleepStage_i$ where $i \in \{2, \ldots, n\}$ must be identical, which can be formulated by using generalized parameters. Because the *SleepStage* nodes produce a Markov chain, of which the parameters of the transition tables (conditional probability tables between each *SleepStage* node) must be identical.

5.4 Learning Neural Network Like Bayesian Networks

The learning algorithms described so far have been very general, simply increasing the likelihood of the parameters or minimizing the distance to some prior request. For classification problems, such as sleep stage classification, we are interested in achieving the best classifier. This is also the idea of **neural network type training** which we try to phrase in a Bayesian Network context, [Jensen, 1999].

First, define the set of **input variables**, I, to be the information variables in the network on which the classification has to be based. Next, define the set of **output variables**, O, to be the hypothesis variables of the network. Note, in the sleep classification network we only have one output variable, the sleep stage. The remaining variables of the network are thus all *hidden variables*.

Each case in the database $\mathbf{d}$ is now on the form $d_l = (i^l_{V_l}, i^l_O)$, where $i^l_{V_l} \in \mathcal{I}_{V_l}$ is called an **input configuration** with $V_l \subseteq I$ allowing missing values on input variables. Furthermore, $i^l_O \in \mathcal{I}_O$, where missing values are not allowed, since we are only considering supervised training. This observation of the hypothesis variables has been obtained by e.g. consulting an expert, using a costly method which we want to replace, or at a later point observe the true states of the output variables. Denote $\mathcal{I}_I(\mathbf{d})$ the set of distinct input configurations observed in $\mathbf{d}$.

The set of data cases, $\mathbf{d}$, is called **functional** if each time an input configuration appears it results in the same output configuration; otherwise $\mathbf{d}$ is called **stochastic**, [Jensen, 1999]. The database for the sleep classification problem is indeed stochastic, e.g. two different experts have different opinions of the current sleep, or the same expert can classify the same input configuration differently. This is a result of the high degree of user interpretation required by [Rechtschaffen and Kales, 1968]. Furthermore, our model does not necessarily take the same input features into account as the expert uses when performing manual analysis according to [Rechtschaffen and Kales, 1968]. Hence, we assume in the following that $\mathbf{d}$ is stochastic.

For each input configuration, $i \in \mathcal{I}_I(\mathbf{d})$, let $\{o|o \in \mathcal{I}_O\}$ be the output configurations associated with i, and let $z_{io}(\theta) = P_{\boldsymbol{\theta}}(o|i)$. As previously we want to tune the parameters, $\boldsymbol{\theta}$, such that we approach $z_{io}(\boldsymbol{\theta}) = y_{i0}$, where y_{io} is the frequency of o given i. Let w_i be the frequency of i. Hereby, the network is trained with respect to the distance, [Jensen, 1999]

$$\mathrm{dist}_E(\mathbf{d}, \boldsymbol{\theta}) = \sum_{i \in \mathcal{I}_I(\mathbf{d})} w_i \sum_{j \in \mathcal{I}_O} \left(z_{ij}(\boldsymbol{\theta}) - y_{ij}\right)^2 . \tag{5.32}$$

Hence, using (5.18), (5.24) (5.27), and (5.26), respectively, the partial derivative of dist_E in (5.32) with respect to a parameter, $\theta_{ljk} \in \boldsymbol{\theta}$, is

$$\begin{aligned}
\frac{\partial\, \mathrm{dist}_E(\mathbf{d}, \boldsymbol{\theta})}{\partial \theta_{ljk}} &= 2 \sum_{i \in \mathcal{I}_I(\mathbf{d})} w_i \sum_{o \in \mathcal{I}_O} \left(z_{io}(\theta_{ljk}) - y_{io}\right) \frac{\partial z_{io}(\theta)}{\partial \theta_{ljk}} \\
&= 2 \sum_{i \in \mathcal{I}_I(\mathbf{d})} w_i \sum_{o \in \mathcal{I}_O} \left(z_{io}(\theta_{ljk}) - y_{io}\right) \frac{1}{P_{\boldsymbol{\theta}}(i)} \left(\frac{\partial P_{\boldsymbol{\theta}}(i,o)}{\partial \theta_{ljk}} - z_{io}(\theta_{ljk}) \frac{\partial P_{\boldsymbol{\theta}}(i)}{\partial \theta_{ljk}} \right) \\
&= 2 \sum_{i \in \mathcal{I}_I(\mathbf{d})} w_i \sum_{o \in \mathcal{I}_O} \left(z_{io}(\theta_{ljk}) - y_{io}\right) Q_{ljk}(i,o),
\end{aligned} \tag{5.33}$$

where $\theta_{ljk} = P(X_l = x_{lj}|\pi_{lk})$ and

$$Q_{ljk}(i,o) = \frac{P_{\boldsymbol{\theta}}(x_{lj}, \pi_{lk}|i,o) P_{\boldsymbol{\theta}}(i,o) - z_i(\theta_{ljk}) P_{\boldsymbol{\theta}}(x_{lj}, \pi_{lk}|i) P_{\boldsymbol{\theta}}(i)}{\theta_{ljk} P_{\boldsymbol{\theta}}(i)}.$$

Thus, the gradient, $\nabla_{\boldsymbol{\theta}}\, \mathrm{dist}_E(\mathbf{d}, \boldsymbol{\theta})$, is a vector of all the partial derivatives w.r.t. $\boldsymbol{\theta}$ computed as in (5.33).

5.4.1 Calculating the Gradient in a Small Example

Consider the Bayesian network in Figure 5.6 with conditional probability tables as in Table 5.1.

$$P(B) = \begin{array}{c|cc} B & 0 & 1 \\ \hline & \theta_1 & \theta_2 \end{array} \quad P(A|B) = \begin{array}{c|cc} A \mid B & 0 & 1 \\ \hline 0 & 0.6 & 0.2 \\ 1 & 0.4 & 0.8 \end{array} \quad P(C|B) = \begin{array}{c|cc} C \mid B & 0 & 1 \\ \hline 0 & 0.9 & 0.7 \\ 1 & 0.1 & 0.3 \end{array}$$

Table 5.1: The conditional probability tables for the simple Bayesian network in Figure 5.6. Note, $P(B)$ contains two parameters, thus $\boldsymbol{\theta} = (\theta_1, \theta_2)$

Assume that the Bayesian network represents a classification scenario where $I = \{A, C\}$ and $O = \{B\}$ with two parameters $\boldsymbol{\theta} = (\theta_1, \theta_2)$ as given by Table 5.1. No informative prior knowledge exist for $\boldsymbol{\theta}$, therefore the startvalue is $\boldsymbol{\theta}^0 = (0.5, 0.5)$.

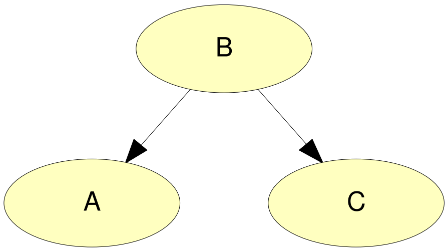

Figure 5.6: A simple Bayesian network used for experiments with neural network type training.

For simplicity we assume $\mathbf{d}$ consists only of the case $(i, o) = ((A = 0, C = 0), (B = 0))$. Hence, $y_{i\{B=0\}} = 1$ and $y_{i\{B=1\}} = 0$. In the example it is fairly easy to compute the joint probability $P(A, B, C)$. From this we extract,

$$\begin{aligned} P_{\boldsymbol{\theta}^0}(B = 0, i) &= 0.27, \\ P_{\boldsymbol{\theta}^0}(B = 1, i) &= 0.07, \\ P_{\boldsymbol{\theta}^0}(i) &= 0.34 \end{aligned}$$

Hence, $z_{i\{B=0\}} = 0.794$, and $z_{i\{B=1\}} = 0.206$.

Using these values in (5.26) and (5.27) yields the six partial derivatives necessary.

$$\frac{\partial P_{\theta^0}(i, B)}{\partial \theta_i} = \begin{array}{c|ccc} \theta_i \mid B & 0 & 1 & \emptyset \\ \hline 1 & 0.54 & 0 & 0.54 \\ 2 & 0 & 0.14 & 0.14 \end{array}$$

Inserting these values into (5.33) yields the unconstrained gradient.

$$\nabla'_{\boldsymbol{\theta}^0} \operatorname{dist}_E(\mathbf{d}, \boldsymbol{\theta}^0) = (-0.269, 0.269)$$

As the sum of the individual components of the unconstrained gradient is already zero it is equal to the constrained gradient. Multiplying the obtained values by the overall damping $\delta = 0.1$ and advancing the parameters opposite the gradient gives $\boldsymbol{\theta}^1 = (0.5269, 0.4731)$. By repeating these computations the parameters are moved arbitrary close to $(1, 0)$ as as expected.

5.5 Theoretical Considerations

In this section we perform a theoretic comparison between neural network type training (NNTT), i.e. gradient descent with metric $\operatorname{dist}_E(\mathbf{d})$, and the likelihood approaches operating on $\log P(\mathbf{d})$. Our aim is to let the two approaches compete against each other in classification scenarios in order to see which approach performs best.

The main difference between the two approaches is the metric. Likelihood approaches carry out a form of unsupervised learning because they do not distinguish the output variables from the input variables. The objective is to find a distribution which best describes the probability distribution of the observed data. NNTT performs supervised learning, which requires observations on the variables in the output set. Here, the network is tuned such that it classifies as consistent as possible with the training set. This difference has several implications.

As NNTT tries to calculate $P(O = o|I = i)$ for each configuration (i, o) in $\mathbf{d}$, sensitivity analysis as known from [Coupé and van der Gaag, 1998b] reveals that not all parameters actually have influence on $P(o|i)$. A parameter θ_i is only influential if it is located in the conditional probability table of a node V, such that $V \in \mathrm{Sen}(O, I)$, where the sensitivity set $\mathrm{Sen}(O, I)$ can be determined by e.g. the Bayes-Ball algorithm [Shachter, 1998]. Depending on the evidence, the number of parameters actually adjusted by NNTT can be very small compared to the total number of parameters in the setup.

If $\mathbf{d}$ is fully observed, the set of observed input nodes for each case is equal in all cases, i.e. I. In case of missing data the number of input sets $n_I = |\mathcal{I}_I(\mathbf{d})|$ can be up to $|\mathcal{P}(I)|$. If n_I is relatively small, we can, before performing NNTT, apply Bayes-Ball to each evidence set in order to extract the relevant subset of $\boldsymbol{\theta}$ which require calculation of gradients. A major disadvantage occurs once a net trained by NNTT is used on a cross-validation set with different input variables than used for training. Here, parameters, which have not been adjusted at all by the training procedure, suddenly become influencial thus giving poor results. In these cases likelihood based approaches should perform better since all parameters are adjusted, albeit the aim is to describe the data.

An alternative way to view NNTT is by looking at the y_{io} coefficients. The perfect classifier for NNTT has $z_{io}(\boldsymbol{\theta}) = y_{io}$ for all i and o. With complete observations, the perfect classifier fulfilling this requirement is shown in Figure 5.7(a).

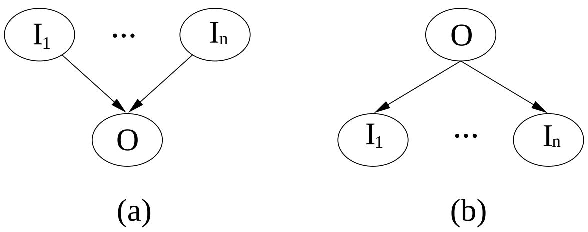

Figure 5.7: (a) The perfect classifier for NNTT when |O|=1. Here, all $P(I_i)$ contain a uniform prior and $P(O|I_1, \ldots, I_n)$ is obtained by frequency counting. (b) Naive-Bayes classifier

In other words, the distance metric is fundamentally comparing the result of the network to be trained against the perfect classifier trying to tune the trained network to behave just as the perfect classifier. If $|\mathcal{I}_I|$ is large the number of probabilities which have to be obtained by frequency counting becomes intractable. NNTT has the advantage that it does not need to represent the entire $P(O|I_1, \ldots, I_n)$ table because only input configurations actually observed in $\mathbf{d}$ are relevant, i.e. n_I input configurations. In case n_I is also large, the situation does not improve.

Assume now we try to learn the naive classifier shown as in Figure 5.7(b) with **d** fully observed. Learning as done with the EM-algorithm corresponds to simple frequency counting. Unless the underlying distribution found by frequency counting in the database fulfills $\forall i, j : i \neq j : I_i \perp\!\!\!\perp I_j | O$, simple frequency counting in (b) results in a different distribution for $P(O|I_1, \ldots, I_n)$ than in (a). Hence, frequency counting in (b) does not necessarily give the best distance. In other words, even if **d** is fully observed, NNTT can train most networks to yield better $\text{dist}_E(\mathbf{d})$ than frequency counting can.

5.6 Sampling

In this section we describe how to sample cases from a Bayesian network such that the distribution of the cases follows the one specified by the conditional probability tables. The sampling of data cases following a specific distribution will be used to examine and validate the learning algorithms, especially when implementing the learner.

Several algorithms for sampling (also called *simulation*) of data cases exist, each having different purposes and different time complexities, [Shachter and Peot, 1990]. For the purpose of this thesis a simple algorithm to sample data cases without handling evidence is satisfactory. Therefore, we choose *Logic Sampling*[1] which is the simplest sampling algorithm.

5.6.1 Logic Sampling

Let $B = (G, P)$ be a Bayesian network, and let $V(G) = X_1, \ldots, X_n$ be the nodes in B such that each X_i is considered as a random variable. The DAG, G, induces a partial order, $\prec$, on the variables such that when $i \neq j$, $X_i \prec X_j$ if $X_i \in \pi^*(X_j)$, i.e. X_i is an ancestor of X_j. Assume $X_1, \ldots, X_n$ are ordered with respect to $\prec$. Then, each data case sample can be performed as a sequence of samplings performed in this order, i.e. first select a state in X_1, then X_2, etc. The ordering ensures that when sampling variable X_i all its parents have already been instantiated; hence, sampling X_i is simply selecting a state x_i of X_i with propability $P(X_i = x_i | \pi(X_i) = \pi_i)$. Thus, the probability of selecting $\mathbf{x} = x_1, \ldots, x_n$ is given by

$$P(\mathbf{x}) = \prod_{i=1,\ldots,n} P(X_i = x_i | \pi(X_i)),$$

which is independent of observed evidence, [Shachter and Peot, 1990]. Evidence is irrelevant for our purposes, therefore see [Shachter and Peot, 1990] or [Jensen, 1996] for further discussion about evidence and sampling.

[1] logic sampling is also known as *forward sampling*, see [Jensen, 1996].

5.6.2 Experiments with Logic Sampling

We will now briefly do some experiments with the Logic Sampling algorithm to see how good it is for different number of sample sizes for a simple Bayesian network with all variables fully observed. Again consider the Bayesian network in Figure 5.6 with conditional probability tables as specified by Table 5.1 except that $\theta_1 = 0.2$ and $\theta_2 = 0.8$.

To validate the sampling algorithm the Bayesian network is sampled with different sample sizes, and the distributions is tested with a χ^2 test of goodness-of-fit, see [DeGroot, 1985], and a **mean error** determined by, [Shachter and Peot, 1990],

$$\frac{1}{|V(G)|}\sqrt{\sum_{i \in \mathcal{I}_{\{A,B,C\}}} \frac{(\hat{P}(i) - P(i))^2}{P(i)(1-P(i))}}.$$

The **goodness-of-fit** test is defined as follows. Let $P(A,B,C) = P(B)P(A|B)P(C|B)$ be the joint distribution of the sample network and $\hat{P}(A,B,C)$ be the estimate of $P(A,B,C)$ given by the sampled data cases. Let $i \in \mathcal{I}_{\{A,B,C\}}$ be a configuration of the joint distribution, and let $P(i) \geq 0$ and $\hat{P}(i) > 0$ be the table cells of $P(A,B,C)$ and $\hat{P}(A,B,C)$, respectively, where $\sum_{i \in \mathcal{I}_{\{A,B,C\}}} P(i) = 1$ and $\sum_{i \in \mathcal{I}_{\{A,B,C\}}} \hat{P}(i) = 1$. The following hypotheses are to be tested:

$$\begin{aligned} H_0: \quad & \forall i \in \mathcal{I}_{\{A,B,C\}} : \hat{P}(i) = P(i) \\ H_1: \quad & \exists i \in \mathcal{I}_{\{A,B,C\}} : \hat{P}(i) \neq P(i) \end{aligned} \tag{5.34}$$

Now, let

$$Q = \sum_{i \in \mathcal{I}_{\{A,B,C\}}} \frac{(n(i) - NP(i))^2}{NP(i)}.$$

As the sample size $N \to \infty$ the density function of Q converges to the density function of the χ^2 distribution with $|\mathcal{I}_{\{A,B,C\}}| - 1$ degrees of freedom. The H_0 hypothesis is rejected when $Q > c$, where c is an appropriate constant. Carrying out the test at a level of significance α is equal to choosing c such that $P(Q > c) = \alpha$. The value of c can thus be determined from $Q \sim \chi^2(|\mathcal{I}_{\{A,B,C\}}| - 1)$, [DeGroot, 1985, Section 9.1]. Rejecting when $Q > c$ is equivalent to rejecting if the p-value is less than α.

In the goodness-of-fit test of our example we choose the level of significance to be $\alpha = 0.05$, and as $Q \sim \chi^2(7)$, this means that $c = 14.07$, [DeGroot, 1985]. The result of the goodness-of-fit test and the calculation of the mean error is depicted in Table 5.2.

From the test results in Table 5.2 it can be seen that we will not reject any of the sample sizes tested; but, it is not recommended to use sample sizes below 100 because of the high mean error. The heavily varying p-values is probably due to the bad random number generator implemented in the standard C library used, see [Press *et al.*, 1992, Chapter 7] for further discussion on random numbers and possible improvements.

N	d.f.	Q	p-value	mean error
1	7	1.2321	0.990	0.455
5	7	4.2262	0.752	0.324
10	7	3.6491	0.812	0.222
50	7	12.8092	0.081	0.187
100	7	3.3094	0.835	0.063
500	7	3.3676	0.831	0.028
1000	7	8.9425	0.257	0.034
5000	7	5.4092	0.620	0.012
10000	7	7.7745	0.355	0.010
25000	7	14.0095	0.051	0.008

Table 5.2: The result of the goodness-of-fit test with level of significance at $\alpha = 0.05$ and mean error of the sampling of the simple Bayesian network.

5.7 Implementational Dimension

An implementational dimension is inevitable when we want to use the neural network type training, gradient descent with likelihood metric and the EM-algorithm in practice. A powerful general purpose tool named tsl (Time Sliced Learner) has been implemented by us which fulfills the needs for learning, sampling, and succeeding evaluation for classification scenarios.

Action of the tsl is controlled by specifying versatile script files written in the tslsl language. Using Flex/Yacc a tslsl compiler is implemented. Observed cases are written to or fetched from a remote Oracle SQL database allowing concurrent access and exploitation of data already present in SQL databases. The architectural design of tsl is based on a philosophy that tsl should be more than a quick hack used for evaluations in this project. This results in a flexible component design allowing for future extension with e.g. adapatation and sensitivity analysis.

Further knowledge about tsl is not required, but we *strongly* encourage the reader to read Appendix A containing an extensive description of tsl.

5.8 Summary

In this chapter we described three different learning methods for Bayesian networks, two likelihood approaches and the neural network type training. We hope these methods can help us in learning the networks used in the sleep classification task.

The idea of training in neural networks was adopted to the context of Bayesian Networks by formulating an appropriate classification evaluation metric, dist_E. Training is performed by gradient descent on dist_E. Using results from sensitivity analysis we showed that calculation of the gradient can be performed locally for each parameter requiring only two propagations.

The strength of Bayesian networks compared to neural networks is that all parameters have a semantic. We use the semantic understanding to query for possible prior knowledge on the values of the parameters. To encompass prior knowledge into the training process, we propose a damping function which should guide the updating of the parameters. With this guidance we hope that possible sub-optimal extrema might be avoided. Furthermore, prior expert knowledge is based on knowledge outside the training set. Thus, better results on cross-validation sets should be obtained.

The described theory for learning and sampling has been implemented in a general purpose learner (tsl) for Bayesian networks representing classification scenarios.

According to our knowledge, no experiments have been conducted to evaluate the applicability of neural network type training in practice. The next task is therefore to perform a preliminary analysis of the quality and running behavior of neural network training extended with damping compared to the likelihood approaches in a small but realistic example where we have full control over the true distribution. Hereby we can get an idea of which method to emphasize on in the sleep classification task. Working on a manageable example also allows for an extensive testing of the tsl tool before proceeding to the much more comprehensive sleep stage classification problem.

Chapter 6

Experimental Results - The Chest Clinic

In Chapter 5 and Appendix A we described the theory and corresponding implementation of statistical and gradient descent algorithms to learn the probabilities in a Bayesian network. Since the method of neural network type training in Bayesian networks is relatively new, no tests have been performed to evaluate the performance and applicability of it. In the current and following chapter we perform such an investigation by comparing the performance of neural network type training in Bayesian networks against the likelihood metric methods, i.e. gradient descent with likelihood as metric and the EM-algorithm. This chapter focuses on comparing the methods, whereas the next chapter concentrates on solving the sleep classification task. To enhance the results of this comparison, we use a contrived example, where the cases constituting the database have been artificially generated by sampling from a known distribution.

The chapter begins by describing the contrived example, and the different test setups within this example, under which the methods should be compared. Hereafter, we analyze in detail an illustrative subset of the test setups for the running behavior of neural network type training. Finally, the end-results of the three methods are compared for different groupings of the setups in order to extract strengths and weaknesses of the methods.

6.1 The Chest Clinic Classification Problem

In order for the test to be descriptive and tractable we require the example network to be formulated as a classification task of manageable size. On the other hand we desire a minimum of realism, e.g. the possibility to interpret the variables enabling a better discussion of the behavior. The *chest clinic* example, see [Lauritzen and Spiegelhalter, 1988], fulfills these requirement and both [Thiesson, 1991] and [Olesen *et al.*, 1992] use it in their experimental testing.

In the chest clinic a doctor is treating a patient who complains about dyspnoea (shortness of breath). The set of likely diagnoses is the powerset of the diseases tuberculosis, lung cancer and bronchitis. Furthermore, smoking is known to increase the probability of lung cancer and bronchitis, whereas a recent visit to Asia increases the probability of tuberculosis. An X-ray investigation of the lungs is not able to discriminate between lung cancer and tuberculosis. All the information is combined in Figure 6.1 which shows the structure of the network.

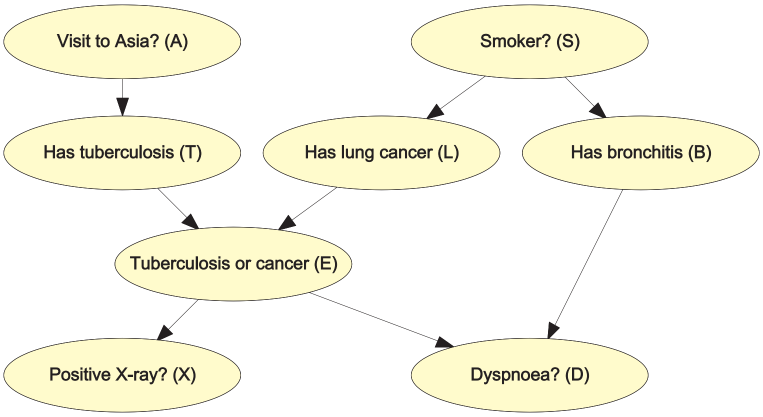

Figure 6.1: The chest clinic network as a Bayesian network. In our classification context, T is chosen as the output node. The node E is a logical OR node.

Translating the chest clinic into a classification task, the output variables are T, L, and B, but for tsl to operate we reduce the set of output variable to contain only one element. Tuberculosis is the only of the three diseases which is contagious, and the chest clinic network could be seen as part of screening system for tuberculosis where all adults are offered a checkup. The probabilities assigned to the tables of the network are as described in [Lauritzen and Spiegelhalter, 1988], except for the prior probability of S where we due to the screening context and for the sake of moral applicability decided to replace the uniform distribution by 1997 numbers for Danish smoking habits [Tobaksskaderådet, 1999]. Hence, $P(S = y) = 0.35$. This distribution will be denoted the *true distribution* of the chest clinic network.

As all the variables in the network are binary the total state space of the model is $|\mathcal{I}_{V(G)}| = 2^8 = 256$, and the total number of conditional probabilities to be specified is 36.

Our investigation now takes place within the framework of the chest clinic example. The task is to assess the conditional probabilities from a combination of observed cases and, if available, expert information. The number of observed cases, the quality of the expert knowledge, etc. cause different conditions which influence the quality of the learning algorithms. Our goal is to make the chest clinic network as good a classifier as possible, therefore we measure performance by dist_E as in (5.32) on page 64.

6.2 Test Setups

To get a systematic methodology describing the different test setups, we use a factorized test based on the strategy from [Thiesson, 1991] and customized by us for the purpose of testing neural network type training for Bayesian networks. We use a total of six factors: data acquisition, observation scheme, sampling scheme, learning method, parameter knowledge, and number of unknowns. A detailed description of the levels in each factors follows.

The chest clinic is a contrived example, therefore the cases of the database are generated by sampling as described in Section 5.6. Using sampling from the true distribution also has the advantage that we for the likelihood approaches know what kind of results we would like to obtain by learning, i.e. the same distribution from which we sampled. As discussed in Section 5.5 the distance measure is a different concept since the distance is a measure on how good the trained network scores compared to the network in Figure 5.7(b) on page 66. Thus, the true distribution does not necessarily give the best results in neural network type training.

The levels of the *data acquisition* factor, A, describe three different database sizes, which should be compared to $|\mathcal{I}_{V(G)}| = 256$. The problem with small databases is that some cells of $|\mathcal{I}_{V(G)}|$ are never observed even if their probability is non zero.

A1 A large sample of 10.000 cases.

A2 A medium sample of 1.000 cases.

A3 A small sample of only 100 cases. Nevertheless, in real world applications this might be all we have since it is very time consuming to gather patient records, to have experts classify, etc.

The *observation scheme* factor, O, contains three levels. It reflects the size of the input set, i.e. how many of the variables the doctor was able to observe. Since we perform only supervised training we require that the state of the tuberculosis variable is observed for each case.

O1 Input set consists of $V(G)\backslash\{T\}$. This ideal situation is included for establishing a baseline which we can compare the other levels to.

O2 Input set consists of $V(G)\backslash\{T, E\}$. Since E is an logic OR node, knowing $E = n$ implies $T = n$, and hence a certain diagnosis which is unrealistic since a doctor would never be able to observe E.

O3 Input set consists of S, A, X, and D. This level is regarded as the most realistic situation for the doctor.

Sporadic missing data, S, denotes whether the variables in O are fully observed or can be sporadic missing.

S1 The variables in the O factor are fully observed.

S2 Full observations on A,T,L,S, and B. The variables X and D are missing with probability 0.7 and 0.3, respectively. This should reflect a true scenario where the doctor did not always have resources to perform an X-ray or was not able to determine whether the patient really suffered from dyspnoea.

The structure of the database is now described by the Cartesian product of the levels in the observation scheme and sporadic missing factors. Similarly, the setup of the parameters is described by the level of parameter knowledge and number of unknowns.

Parameter knowledge, P, describes how good the job of finding the initial values of the parameters could be done. The first three levels concern variables containing parameters with even priors. Hence, the start values should be the mean of the even prior for each column in the conditional probability tables. For the likelihood approaches we are however not completely ignorant about the parameters as we expect them to end up in the true distribution. We thus choose three different start values with increasing distance between the start value and the value from the true value distribution as in [Thiesson, 1991]. The next three levels contain different qualities of prior knowledge by increasing distance between the best guess and the true value and increasing the intervals of variance and thus decrease the virtual sample size. The virtual sample sizes of the columns are not equal — some columns have a lower virtual sample size than average and some have one higher. The variation is however not systematic.

P1 All variables have non-informative priors each with their start value equal to the value of the true distribution. This level is used to generate a baseline for comparisons.

P2 Non-informative priors where the start values suffer from small random perturbations from the true values.

P3 Non-informative priors with start values suffering from large perturbations from the true values.

P4 Priors with the true means as best guesses and small intervals of variation, i.e. large virtual sample size.

P5 Same best guesses as the start value in P2, medium intervals of variation, i.e. medium virtual sample sizes.

P6 Best guesses far away from true values and large intervals of variation, i.e. small virtual sample sizes.

P7 Mixture of good and bad priors, i.e. mixed qualities of starting values and confidence intervals. This level is meant to reflect the typical behavior of a real specification where we in cooperation with an expert managed to obtain good values for some probabilities while we on the other hand are quite ignorant about others.

Note that the end-result of the EM-algorithm is independent of the choice of starting value for parameters with non-informative priors in setups with full observation. The EM-algorithm is known to give poor results in situations with non-informative priors for networks with hidden variables. Both [Lauritzen, 1996] and [Thiesson, 1991] describe how the use of priors improves the results. With our investigation we hope to show that damping based on priors achieves a similar result in neural network type training.

The factor, *number of unknowns*, U, describes the number of conditional probability table entries in the network, whose values are unknown to us. If an entry is known, it is equal to the true value. No experiments with faulty "known" entries are performed.

U1 All conditional probability tables except $P(E|T,L)$ are unknown. The conditional probability table of E is known since it is an OR node. Hence, $|\boldsymbol{\theta}| = 28$.

U2 The conditional probability tables for T, L, B, D are unknown. We assume that the probabilities for A and S could be obtained by consulting general health and travel statistics. Furthermore, the quality of X-ray examinations to detect lung cancer or tuberculosis would have been investigated by the vendor of the X-ray equipment. Hence, $|\boldsymbol{\theta}| = 20$.

U3 The conditional probability tables for T, D are unknown. A lot of research has been performed on the impact of smoking on lung cancer and bronchitis. These results could possibly be adopted to the chest clinic example. Hence, $|\boldsymbol{\theta}| = 8$.

Note, for neural network type training the size $|\boldsymbol{\theta}|$ is not necessarily the number of parameters which can be adjusted since not all parameters have influence on $\mathrm{dist}_E(\mathbf{d})(\boldsymbol{\theta})$. Sensitivity set considerations as performed in [Coupé and van der Gaag, 1998b] reveal that answering the question, whether a parameter has influence on the probability we try to tune or not, depends on which variables are instantiated by the evidence. Let $d_l = (i_l, o_l)$, where $i_l \in \mathcal{I}_I(\mathbf{d})$ and $o_l \in \mathcal{I}_O$, be a single case. For the distance metric a parameter θ_i has only influence on $P(T|i_l)$ if it is located in a node V such that $V \in \mathrm{Sen}(T, i_l)$. For example, if the database is on the form $(O1, S1)$ all variables are instantiated in the network. Hence, $\mathrm{Sen}(T|i_l) = \{T, E\}$. In this situation, the actual number of influential parameters is four. These four parameters are the only parameters which can be adjusted while training. Unfortunately, such a trained network will yield poor results for a cross-validation set with a different observation scheme. A different observation scheme might result in a different sensitivity set such that indifferent and thus un-adjusted parameters from the training set suddenly become influential.

For $(O2, S1)$ and $(O3, S1)$ we have $\mathrm{Sen}(T|i_l) = \{T, E, X, D\}$ and $\mathrm{Sen}(T|i_l) = \{T, E, X, D, L, B\}$, respectively. Note, the parameters in A and S have no influence on $P(T|i_l)$ for any combination of O and S. To determine the sensitivity set for the database configurations $(O_i, S2)$, $i \in \{1, 2, 3\}$, the Bayes-Ball algorithm needs to be applied to all combinations of instantiations of the sporadic missing variables X and D.

Running tsl unfortunately reveals numeric imprecisions since the partial derivatives of $\mathrm{dist}_E(\mathbf{d})$ with respect to indifferent parameters are not zero, but it obtains values fluctuating between $1 \cdot 10^{-19}$ to $1 \cdot 10^{-21}$. These fluctuations do not however constitute a real problem in practice.

The Cartesian product of the levels in the five mentioned factors constitute our $|\mathcal{I}| = 3 \cdot 3 \cdot 2 \cdot 7 \cdot 3 = 378$ different test setups. Each method described by the *method factor* is applied to all the 378 test setups.

M1 Neural network type training, i.e. gradient descent with frequency weighted distance, $\text{dist}_E(\mathbf{d})$ as metric, see (5.32) on page 64.

M2 Gradient descent with log-likelihood as metric, see Section 5.3.2.

M3 The EM-algorithm with either MLE or MAPE depending on the P factor, see Section 5.2.

The above factorization gives us a total of $3 \cdot 378 = 1134$ tests to perform. A C++ program was implemented for automatic generation of the necessary 1134 tsl specification files. When referring to a specific test scenario, we use the abbreviation $A1M1O1P1S1U1$ to describe the element $(A1, M1, O1, P1, S1, U1)$ in the Cartesian product of the factor levels. Appendix C contains a quick-reference card over the different levels which might be handy to use in the following analysis.

To perform all 1134 tests in an efficient way, shell scripts where developed which split the task according to the different database setups needed. A total of $|\mathcal{I}_O| \times |\mathcal{I}_S| = 6$ different database tables are created with corresponding cache. Main influence on the execution time is the number of distinct input cases. The sub-tasks can then be run in parallel on different machines, letting the Oracle DBMS control concurrent access to the database. Parallelizing the test on three SparcUltra II machines and running them during low-traffic periods gave running times of $A1 = 355$ minutes, $A2 = 157$ minutes, and $A3 = 148$ minutes.

Investigation within these test setups can now be split into two parts. First, an internal investigation is performed, where we look at the influence of step size, thresholds, number of runs, etc. on the running behavior of neural network type training. This discussion leads to choice of appropriate constants for these variables which will be used to compare the running behavior to the other methods. In the second part, this consists of an external investigation where the end-results of methods M1-M3 for the distance functions are compared to each other.

6.3 Analysis of the Running Behavior

Running the neural network type training for the necessary 378 setups requires specification of the variables: step size, threshold and maximum number of runs. We pick out a few of the setups and look on how the running behavior is influenced by these variables. No real study of the running behavior of the likelihood methods $M2$ and $M3$ is performed, consult [Russel *et al.*, 1995] or [Thiesson, 1991], respectively, for such investigations.

The first variable to study is the effect of the overall step size. Figure 6.2 illustrates $\text{dist}_E(\mathbf{d})$ as a function of the number of runs for four different step sizes.

Note how the step sizes of $\delta = 1$ and $\delta = 0.1$ result in oscillations in the value of $\text{dist}_E(\mathbf{d})$. By advancing the parameters by large steps we reach locations in the parameter space where $\text{dist}_E(\mathbf{d})$ has an entirely different topology. For example, we could get near a minimum, to

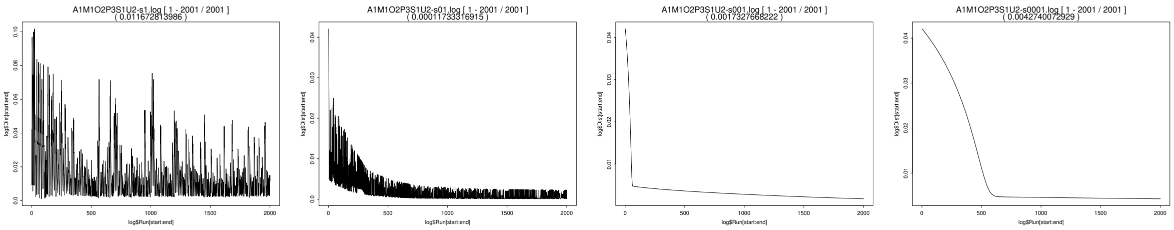

Figure 6.2: The distance metric (vertical axis) as a function of the number of runs (horizontal axis) for neural network type training with step sizes $\delta = 1, 0.1, 0.01$ and 0.001 for the setup $A1M1O2P3S1U2$.

which we then converge, having higher $\text{dist}_E(\mathbf{d})$ value than the current location. Of course the new minimum could also yield a lower value for $\text{dist}_E(\mathbf{d})$. Oscillations become more distinct the more bent and twisted the shape of $\text{dist}_E(\mathbf{d})(\boldsymbol{\theta})$ is. Despite of the oscillations the step size $\delta = 0.1$ manages to converge to a minimum.

The figure also illustrates how smaller step sizes manage to take away the oscillation, but also give poorer end-results. With a smaller step size we are more likely to converge to the extrema closest to the start values. Such a behavior is usually not suitable since this extrema might not be very good. A good choice for the step size is a very complicated matter as the distance function contains a large number of extrema, which heavily depends on the test setup. What might be good step size in one setup is not necessarily good in another. By conducting some ad-hoc investigations we decided to use $\delta = 0.1$ as step size. Choice of a fixed step size is necessary since tsl currently does not implement dynamic step size.

The number of runs carried out by the neural network type training algorithm is determined by a combination of the `threshold` and `maxruns` variables. Our goal is to obtain as good a classification as possible, i.e. we would prefer to run tsl for as many loops as necessary for the difference to get below the threshold. Unfortunately, plots reveal that the method in some cases converges very slow. In order for the 1134 tests to finish in foreseeable future, we limit the number of runs to 2000, thus loosing some of the precision. Figure 6.3 shows a situation where we would have obtained quite a better result by performing a substantially larger amount of runs.

The above discussion leads to a choice for the constants for neural network type training. Similar ad-hoc investigations are conducted for the other methods, which leads to the following `method` choices in the tsl specification files.

```
(gd method=dist stepsize=0.1 threshold=1E-9 maxruns=2000
    logfile="foo.log" ofile="/dev/null/")
(gd method=likelihood stepsize=0.01 threshold=1E-8 maxruns=2000
    logfile="foo.log" ofile="/dev/null/")
(em threshold=1E-8 maxruns=2000 logfile="foo.log" ofile="/dev/null/")
```

Note, [Thiesson, 1991] recommends a threshold of $1 \cdot 10^{-4}$ for the EM-algorithm, as this

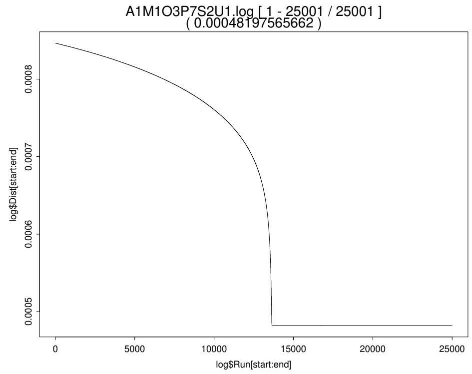

Figure 6.3: A test setup where the distance function for neural network type training suddenly decreases rapidly after approximately 14000 runs. The bend point in the figure occurs because one of the parameters after applying the maximum number of extra damping still is negative.

threshold is reached after few runs, obtaining higher precision practically takes forever. For our investigation we although prefer the high precision since we want to compare the end-results and not convergence rates. Furthermore, the threshold for the log-likelihood methods cannot immediately be compared to the threshold for the distance approach since they are thresholds on different metrics.

The running behavior of the three methods for the same test setup is best compared by a plot as in Figure 6.4. The plot is a good representative of the general pattern exhibited in the plots. The EM-algorithm is trying to maximize the log-likelihood which it achieves in very few steps (here 10-12 steps). The values for the parameters obtained by this process result in a decreasing distance measure, although the convergence does not appear to be as rapidly as for the log-likelihood. Gradient descent with log-likelihood as metric displays a quite oscillating behavior in its attempt to maximize the log-likelihood. This might indicate either too large step size or that our normalization approach does not work as well for the log-likelihood function as for dist_E. Despite the oscillations the likelihood slowly manages to converge to a maximum. The corresponding distance of the obtained parameter values equally oscillates, but also converges, albeit not to an optimal minimum. This plot shows that the goal of maximizing the log-likelihood does not necessarily result in minimizing the distance. Finally, the neural network type training displays quick convergence in the beginning but looses speed after approximately 20 runs. The corresponding log-likelihood increases in a similar manner. By evaluating the end-results it can be seen that both gradient descent methods become trapped in a suboptimal minimum, whereas the EM-algorithm manages to find a better minimum despite the fact that it is merely looking for a maximum of the log-likelihood.

Similar to the gradient descent with likelihood metric example, the goal of minimizing the distance does not always imply maximization of the log-likelihood, see Figure 6.5 for an example. The two functions are although not totally uncorrelated, oscillations in the distance function clearly imply similar behavior for the log-likelihood and vice versa. Browsing

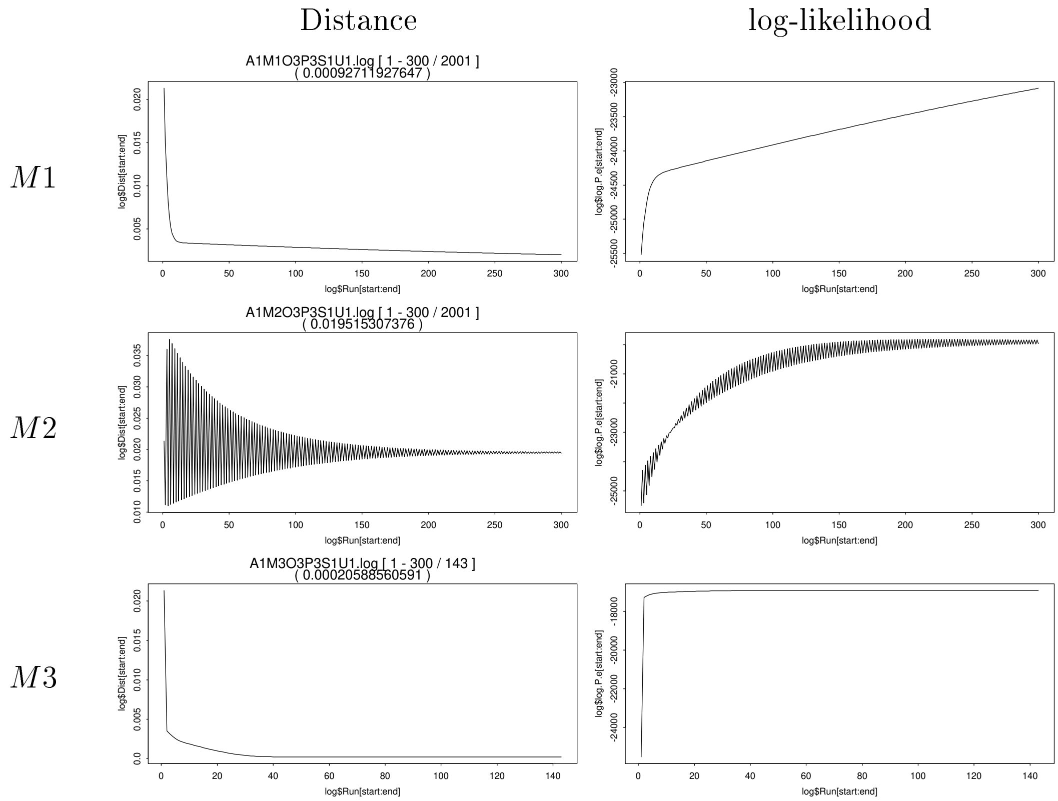

Figure 6.4: The running behavior of the three methods for the setup $A1O3P3S1U1$, i.e. a case with many parameters and many hidden variables. The first column shows the value of $\text{dist}_E(\mathbf{d})$ for methods $M1$, $M2$, and $M3$ as a function of the number of runs. The second column shows $\log P(e)$. For comparisons note the different scalings of the plots.

through the plots of $3 \cdot 378$ setups indicates that the phenomena of non-correlation seems to be more distinct for the gradient descent methods, even though it also occurs for the EM-algorithm as shown i Figure 6.5 between run 3 and 4. This phenomena constitutes a devaluation of the idea of using the quick convergence properties of the EM-algorithm to get near a good minimum of $\text{dist}_E(\mathbf{d})$, after which we switch to neural network training.

Studies of the plots reveal that neural network type training, despite odd mid-life behavior in some plots, always manages to find quite good values of $\text{dist}_E(\mathbf{d})$ compared to the EM-algorithm. To get a better insight into the general behavior and influence of the test setups we perform a statistical analysis of the end-results.

6.4 Evaluating the Quality of the End-Results

For the sake of simplicity we only evaluate quality as distance of the training set. No cross validation set is used, thus over-training small sets or degrading quality in situations, where the input set of the cross-validation scenario differs from the training situation, are never

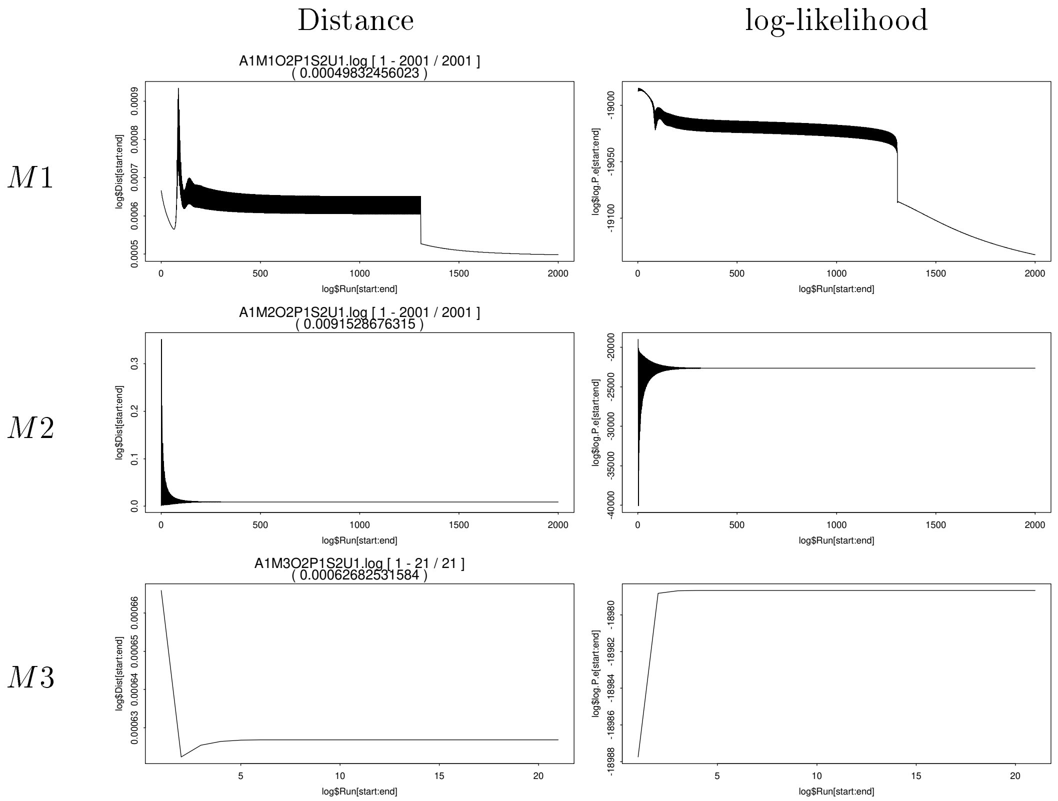

Figure 6.5: Running behavior of $M1$, $M2$, and $M3$ in the setup $A3O2P1S2U1$. Note how minimizing the distance does not always correspond to maximization of the log-likelihood and vice-versa. The mid-life oscillation of $M1$ could indicate the position of a valley which it takes a while to get into.

really revealed. Furthermore, since we are only interested in how good the classification task is performed, no distance measures are taken between the distribution obtained by the network and the true distribution. Before continuing it should be noted that extracting general conclusions from sparse datasets is a risky task, but we think our test is extensive enough to justify certain statements. We choose to investigate four specific topics.

6.4.1 Influence of the Database Size

As the first topic to investigate, we consider the overall quality grouped by the data acquisition (size of the database) factor A, see Table 6.1.

The $\mathbb{E}$ column in the table shows how the end-results for $\text{dist}_E(\mathbf{d})(\boldsymbol{\theta})$ increase as the number of cases decreases. Actually, the $\mathbb{E}$ value for $A3$ is better compared to both $A1$ and $A2$, but this is explained by the following. In the true network the probability is $P(T = y) = 1.04\%$ with only 100 sampled cases, we did not for $A3S2$ obtain any case with $T = y$. Similar for $P(L = y) = 4.15\%$. Hence, all input cases contain $E = n$. Thus, when inserting each input set as evidence and propagating, we get $dist_E = 0$ independently of the parameters.

	M1			M2			M3		
	$\mathbb{E}$	$\mathbb{V}$	won	$\mathbb{E}$	$\mathbb{V}$	won	$\mathbb{E}$	$\mathbb{V}$	won
A1	6.38e-04	2.25e-06	50	1.07e-02	9.67e-05	0	3.42e-04	2.72e-07	76
A2	1.82e-03	4.81e-06	91	1.23e-02	9.59e-05	1	1.92e-03	4.38e-06	34
A3	1.34e-03	1.40e-05	76	1.04e-02	2.99e-04	51	6.11e-03	2.41e-04	41

Table 6.1: End-results for $M1 - M3$ sorted by the A factor. The $\mathbb{E}$ column contains the average of $\text{dist}_E(\mathbf{d})(\boldsymbol{\theta})$ over all O, P, S, T levels, whereas $\mathbb{V}$ contains the variance. The `won` column shows the number of test setups (out of 126) where method M_i had the best end-result for $\text{dist}_E(\mathbf{d})$. In cases of ties both methods are awarded a victory.

For neural network type training this means that no parameters are adjusted at all. Using the obtained network in situations without evidence on E will yield very poor results, here the likelihood approaches would perform much better. A problem although occurs for the EM-algorithm in situations where no $E = y$ case exists in the database. For complete observations the EM-algorithm corresponded to frequency counting, thus for some node V such that $E \in \text{pa}(V)$ we have for the table $P(V \mid \text{pa}(V))$ that

$$\exists i_{\text{pa}(V)} \in \mathcal{I}_{\text{pa}(V)} : \forall i_V \in \mathcal{I}_V : P(i_V | i_{\text{pa}(V)}) = 0$$

The result is that the column $P(\mathcal{I}_V | i_{\text{pa}(V)})$ becomes a so called **zero-sum-column**, which results in an error in the Hugin API, exiting the execution of the EM-algorithm without changing the parameters.

The fact that the the distance measure increases as the size of the database decreases actually comes as a surprise as we expected that it is easier to over-train a network to give exact matches on a small number of cases than a large number. A possible explanation could be that statements in small databases become more "sharp", i.e. the training set is more functional. Here the frequency weighted distance function $\text{dist}_E(\mathbf{d}, \boldsymbol{\theta})$ might not be the best choice, a more function oriented metric might perform better. Further investigations are necessary to support this claim.

The number of total victories for the $A3$ level, $76 + 51 + 41 = 168$, confirms our explanation of how the $(O1, O2)$ level results in $\text{dist}_E(\mathbf{d}) = 0$. The total number of $A3$ setups the three methods compete about is $|\mathcal{I}_P| \cdot |\mathcal{I}_S| \cdot |\mathcal{I}_U| \cdot |\mathcal{I}_O| = 126$, hence the number of ties is $(168 - 126)/2 = 21$. This equals the number of $(A3, O1)$ setups.

Considering the number of won setups, it appears as if the number of victories for neural network type training, $M1$, compared to the EM-algorithm, $M3$, increases in situations with fewer cases. This observation is valuable since the data for our sleep classification task is quite sparse, thus establishing $M1$ as the better method for the problem. Gradient descent with the likelihood metric, $M2$, appears to perform inferior compared to the two other methods.

6.4.2 Influence of the Damping Function

Our second goal is to determine whether adding damping to the gradient descent methods is able to improve the results in case of hidden variables. For this purpose we group the end-results according to the factors P and O. Table 6.2 depicts these results. The factor O is included in order to show variations for different levels of hidden variables.

		M1			M2			M3		
		$\mathbb{E}$	$\mathbb{V}$	won	$\mathbb{E}$	$\mathbb{V}$	won	$\mathbb{E}$	$\mathbb{V}$	won
O1	P1	1.85e-05	1.29e-09	18	5.36e-04	1.48e-07	3	3.69e-05	2.30e-09	3
	P2	2.09e-05	1.34e-09	18	5.36e-04	1.48e-07	3	4.49e-05	2.05e-09	3
	P3	3.38e-05	1.72e-09	12	5.36e-04	1.48e-07	3	1.12e-04	2.63e-08	9
	P4	2.70e-05	2.08e-09	15	5.36e-04	1.48e-07	3	2.99e-05	1.93e-09	6
	P5	7.13e-05	1.79e-09	6	5.36e-04	1.48e-07	3	3.68e-05	2.19e-09	15
	P6	9.24e-05	2.87e-09	3	5.36e-04	1.48e-07	3	4.11e-05	2.65e-09	18
	P7	2.77e-05	3.33e-09	9	5.36e-04	1.48e-07	3	2.74e-05	3.27e-09	12
O2	P1	1.76e-03	6.92e-06	11	1.62e-02	1.64e-04	3	1.90e-03	2.77e-06	4
	P2	2.54e-03	1.65e-05	11	1.67e-02	1.98e-04	3	8.59e-03	2.70e-04	4
	P3	1.81e-03	5.86e-06	10	1.75e-02	2.46e-04	3	2.27e-02	1.03e-03	5
	P4	1.20e-03	1.93e-06	13	1.62e-02	1.63e-04	3	1.96e-03	3.78e-06	2
	P5	1.81e-03	4.42e-06	5	1.67e-02	1.98e-04	3	2.11e-03	5.09e-06	10
	P6	1.18e-03	1.95e-06	14	1.72e-02	2.26e-04	3	2.61e-03	1.05e-05	1
	P7	7.88e-03	3.84e-05	2	1.77e-02	2.60e-04	5	6.39e-03	4.27e-05	11
O3	P1	5.99e-04	7.35e-07	14	1.60e-02	1.32e-04	0	1.42e-03	4.28e-06	4
	P2	5.43e-04	6.84e-07	13	1.60e-02	1.32e-04	0	1.42e-03	4.28e-06	5
	P3	7.72e-04	8.95e-07	7	1.60e-02	1.32e-04	0	1.42e-03	4.28e-06	11
	P4	7.91e-04	1.17e-06	12	1.60e-02	1.31e-04	2	1.20e-03	1.93e-06	4
	P5	8.54e-04	8.25e-07	11	1.60e-02	1.32e-04	2	1.23e-03	1.97e-06	5
	P6	7.17e-04	6.65e-07	9	1.60e-02	1.32e-04	3	1.86e-03	6.52e-06	6
	P7	3.83e-04	1.01e-05	4	1.60e-02	1.32e-04	1	3.47e-03	1.10e-05	13

Table 6.2: End-results categorized by the O and P factors.

The end-results for $M1$ in the table show that damping is only beneficial in cases with hidden variables. Of special interest are the duels $P2$ vs. $P5$ and $P3$ vs. $P6$, i.e. medium and low quality of start value against medium and low quality of prior knowledge. These duels reveal that damping decreases the quality in situations where we started relatively close to the true value, whereas they are advantageous when starting far away from the true value, allowing for large variance. Unfortunately, this does not really reveal the quality of damping as guidance because the virtual sample size is almost equal for all parameters. Therefore, the net effect practically amounts to a decrease in the overall step size by a constant factor. To improve the evaluation of damping, extra levels should have been added to the P factor where we systematically investigate the effect of mixed prior knowledge. That is, in the same specification file to have some columns with best guess close to the true value and small interval of variance, while other columns contain a best guess far away from the true value but with a large interval of variance.

The mixture of good and bad priors performs quite bad in all scenarios, the only encouraging fact is that for neural network type training it behaves better in realistic observation schemes, $O3$, than in $O2$. This is promising as our specification for the sleep network quite likely will be of the mixture type and the same time we have a lot of hidden nodes in the constrained model. The EM-algorithm performs opposite, for $O1$ only sporadic missing data occurs and here the priors really manage to give better results, the pattern is repeated for $O2$. Although, in $O3$ the quality decreases. A possible explanation could be that with the many hidden variables, prior information weights heavily as there are no observations to contradict it. Hence, bad quality of priors results in bad overall results.

Furtheremore, $M1$ performs much better for $O3$ than for $O2$. This is a result of the sensitivity set considerations performed earlier. For $O3$ the actual number of influential parameters is much higher, giving a higher ability to adjust to the data.

We are puzzled by the behavior of $M2$ in the $O1$ and $O3$ rows, investigations show that for e.g. $O1$ the method converges to the same distribution independent of start values and damping. The picture is repeated even when the tests are rerun with a smaller step size. A possible explanation could be that the likelihood function has only one maximum, but this is contradicted by investigations of the end-likelihood results of $M3$ which show larger values. Further investigations are necessary before statements about the behavior of can be made.

Analysis of the number of won setups clearly shows that neural network type training beats the EM-algorithm for non-informative priors. The victory is most notifyable in situations with lots of observations. The reason for this behavior is that the distance function similar to the log-likelihood contains many more sub-optimal local extrema, in situations with a lot of missing data, [Thiesson, 1991, pp. 58-62]. Therefore, it is more likely that the $\mathrm{dist}_E(\mathbf{d})$ method becomes trapped in a sub-optimal minimum. Damping does not appear to be optimal and can in no way be a match for the use of prior knowledge in the EM-algorithm.

6.4.3 Influence of Missing Data

For a further analysis on the effect of missing data in the form of hidden variables and sporadic missing data the end-results are grouped according to the factors S and O, see Table 6.3.

To avoid the $\mathrm{dist}_E(\mathbf{d}) = 0$ situations present in $A3$, only $A1$ and $A2$ setups have been used in the analysis. The numbers confirm the pattern observed in Table 6.2, but the increased quality for $O3$ is not as distinct as before due to the addition of $P7$. Furthermore, for $O2$ and $O3$ the $S1$ level always yields better results than $S2$. This is not surprising for the likelihood based approaches since a multiple number of stationary points exists for the EM-algorithm in situations with sporadic missing data, [Thiesson, 1991, p.58]. Hence, an increased number of sub-optimal extrema exist for the likelihood function. This property appears to exist for the dist_E funchction as well, though investigations are necessary to qualify the statement.

		M1			M2			M3		
		$\mathbb{E}$	$\mathbb{V}$	won	$\mathbb{E}$	$\mathbb{V}$	won	$\mathbb{E}$	$\mathbb{V}$	won
O1	S1	8.18e-05	3.25e-09	21	6.86e-04	1.73e-08	0	7.74e-05	3.71e-09	21
	S2	3.47e-05	1.57e-09	24	8.36e-04	8.18e-08	0	2.66e-05	3.04e-10	18
O2	S1	1.96e-03	1.07e-05	30	1.52e-02	7.00e-05	1	1.57e-03	5.31e-06	11
	S2	2.72e-03	3.56e-06	18	1.65e-02	2.19e-05	0	2.61e-03	3.84e-06	24
O3	S1	1.03e-03	2.02e-06	24	1.91e-02	1.07e-05	0	8.58e-04	1.48e-06	18
	S2	1.54e-03	1.59e-06	24	1.65e-02	3.09e-05	0	1.65e-03	2.20e-06	18

Table 6.3: End-results categorized by the O and S factors. $A3$ results were not used in the analysis.

For example, it could be worthwhile to investigate whether the number of extrema for dist_E increases in situations with sporadic missing data, or if the degrading end-results are simply due to a larger variance of the results obtained in the extrema.

6.4.4 Influence of the Number of Parameters

As a last topic we look on how the number of parameters effects the results, i.e. classification is performed according to U, see Table 6.4. The results are as expected since the true distribution does not necessarily yield the lowest distance measure, thus locking values to their true value takes away some freedom of manoeuvring. Furthermore, it is worth noting that $M1$ through all setups achieves a substantial amount of victories against $M3$. This number degrades as we get fewer parameters, which must mean that the EM-algorithm is better squeezing good adjustments out of fewer parameters. Since we have many unknowns in our sleep network, we do not worry about this property.

	M1			M2			M3		
	$\mathbb{E}$	$\mathbb{V}$	won	$\mathbb{E}$	$\mathbb{V}$	won	$\mathbb{E}$	$\mathbb{V}$	won
U1	1.19e-03	3.90e-06	49	8.98e-03	6.23e-05	1	1.12e-03	2.92e-06	34
U2	1.22e-03	3.86e-06	47	9.90e-03	6.89e-05	0	1.13e-03	2.96e-06	37
U3	1.28e-03	3.93e-06	45	1.55e-02	1.35e-04	0	1.14e-03	3.02e-06	39

Table 6.4: End-results sorted according to the U factor. Only the databases A1 and A2 were used.

6.5 Consequences with respect to tsl

In this section we discuss the consequences of the obtained test results with respect to our tsl implementation.

Static step size seems to be a handicap. A more effective solution could have been to use a large step size in the beginning resulting in large jumps in the parameter space. After a number of iterations, we choose the best result and continue from this result with a smaller step size. Nestings of this procedure can be performed. Due to oscillations, tsl should during the entire program execution remember the instance of $\boldsymbol{\theta}$ which gave the best metric value so far.

From the tests it appears as if our normalization approach results in added oscillation in the plots of metric value against run number. Translation to the constrained surface should be performed by the gradient projection approach instead. It should then be investigated if the new approach manages to reduce oscillations, and whether better end-results are obtained.

To evaluate the choice of damping function in tsl, we should try to perform a more in depth investigation on smaller example where it is possible to obtain an analytical expression for $\text{dist}_E(\mathbf{d})(\boldsymbol{\theta})$. Within this framework, the effects of the damping function can be investigated with respect to the used distance measure and advances in the parameter space.

Further work is necessary to include prior knowledge into gradient descent methods. Our choice of damping function does not appear to work optimal in cases with hidden nodes. But before discarding it, one might try to translate further concepts of neural network training into the context of Bayesian networks. A possible extension could be random-restart gradient-descent, where we save the current distance value and parameters, and once the difference threshold is reached, we perform a random perturbation of the parameters and continue the run. This process can repeat itself a number of times, finally we choose the parameters yielding the lowest distance. Another possibility could be to incorporate simulated annealing methods, where we first find a random direction to move in. Now, if $\text{dist}_E(\mathbf{d})$ is decreased by moving in this direction, we advance the parameters accordingly, otherwise advancing is only performed with probability P_m, where $P_m < 1$. That is, we accept an increase in distance hoping that the move managed to bring us near a better extremum, see [Russel and Norvig, 1995, Section 4.4] for further details.

6.6 Summary

In this chapter, we performed an evaluation of the practical applicability of neural network type training in Bayesian networks compared to especially the EM-algorithm. A factorized test containing 1134 different scenarios was formulated which should test some strengths and weaknesses of neural network type training compared to both gradient descent with likelihood metric and the EM-algorithm. An ad-hoc investigation established values for the variables controlling the running behavior of each of the three methods.

The effect of damping is only noticeable when the alternative is non-informative priors with starting values far away from the true values. Use of damping in these situations is although no match for the results obtained by the EM-algorithm with MAPE. A more systematic investigation with mixed priors in the same specification file is required before more precise statements about the effect of the damping function can be made. Also, we only investigated the guidance property of damping, no studies of the effect of damping in cross-validation sets were performed.

The neural network type training performed quite well in comparison with the EM-algorithm with respect to end-results of $\text{dist}_E(\mathbf{d})$. Of course, neural network is operating on exactly this metric, but the gradient descent approach could e.g. have had a higher tendency to get stuck in sub-optimal extrema than the EM-algorithm. For our 378 comparisons, the neural network type training achieved 217 wins against 151 of the EM-algorithm, of which 21 were ties. Good quality for neural network type training is obtained in situations with small databases and a great amount of hidden variables, this particularly favors the method as a tool in solving the sleep classification task. Gradient descent with likelihood as metric performs very poor and should be discarded in the remaining analysis.

All together the analysis of the 1134 test setups revealed neural network type training as a well suited method for solving classification problems. The task is now to use neural network type training to solve our sleep classification task.

Chapter 7

Training and Evaluation of the Sleep Stage Classifier

In this chapter we will learn the Bayesian networks for the sleep classification task presented in Chapter 4. As described in Chapter 2 sleep analysis is about scoring or classifying the sleep stages (Wake, REM, 1, 2, 3, or 4) from the EEG and AOG signals recorded during a full night's sleep.

The models proposed in Chapter 4 cover a simple-Bayes model and a more advanced model, the constrained model, with constraints on the "true" values on some of the EEG frequency bands. Both models will be examined with and without time and Markov chain property history extensions.

In the next sections we will train these sleep classification models with a training set consisting of four out of six patients' sleep data, where the data sets are as described in Section 3.2. Both the training set and the cross-validation set consisting of the remaining two patients' sleep data will be used to compare the resulting classifiers with the experts' manual scorings and the scorings produced by Nightingale.

Before any learning can be performed using the database described in Section 3.2.6, modifications must be performed. The training data set is build such that the four data sets it consists of is merged together into another table where the experts' scorings are collected into one column. The table is created using the following SQL statement:

```
create table CONT (CONT integer, [...], AOG integer, STAGE varchar(1))
```

The merging is performed by adding one data set and one expert at a time, which can be done by a serie of the following SQL statements:

```
insert into CONT (select CONT, [...], AOG, AMD from CONT01)
insert into CONT (select CONT, [...], AOG, GOR from CONT01)
[...]
```

Note, this merging method rises a small problem, the database cannot be unique on the super and sub cases (`CONT` and `EPOCH`, respectively) anymore. The design and implementation of the learner, described in Appendix A, has taken this into account.

First, we examine the simple-Bayes model, next the constrained model is investigated. Finally, we will examine the extensions of the two models with different combinations of history information: the Markov chain property and the time history. Throughout these examinations we will discuss and conclude upon the results.

7.1 The Simple-Bayes Model

In this section we will learn the Bayesian network of the simple-Bayes sleep classification model described in Section 4.1. The different methods for learning Bayesian networks described in Chapter 5 will be used and discussed. Two different approaches will be taken, one using the prior information available, and one without using the information, i.e. even priors.

7.1.1 The Setups of the Evaluation for the Simple-Bayes Model

Consider the simple-Bayes sleep classification model depicted in Figure 7.1. This network will now be trained with the training data set, where the training methods will be the EM-algorithm, the gradient descent method with log-likelihood as metric, and the neural network type training which is gradient descent with the Euclidian distance, $\text{dist}_E(\mathbf{d})$, as metric. All the learning methods are described in detail in Chapter 5.

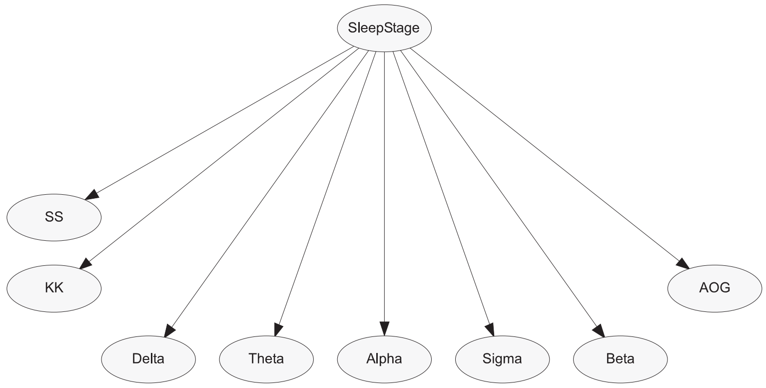

Figure 7.1: The Bayesian network of the simple-Bayes sleep classification model.

Parameters are located in every cell of each conditional probability table of the network in Figure 7.1. Based upon Figure 2.6 on page 11, stemming from [Nielsen, 1992, Fig. 1.5], it is possible to extract prior information for the parameters in the probability table of the *SleepStage* node. However, no informative prior information can be extracted for the parameters of the remaining nodes. Using the best guesses presented in Figure 2.6 with a surrounding interval of $\pm 5\%$ of the value of each best guess, the prior distributions used are as depicted in Table 7.1. The start value of each parameter is thus the mean of its prior distribution.

Six different rounds of training of the simple sleep classification network is performed, the setups of each round are divided such that the three learning methods will be used both with and without the use of informative priors for *SleepStage*. The setups are as follows.

Method	Priors	Step size	Threshold	Max. runs
GD-distance	±5%	0.1	10e-10	2000
GD-likelihood	±5%	0.01	10e-10	2000
EM-algorithm	±5%		10e-03	2000
GD-distance	none	0.1	10e-10	2000
GD-likelihood	none	0.01	10e-10	2000
EM-algorithm	none		10e-03	2000

Stage W	Stage REM	Stage 1	Stage 2	Stage 3	Stage 4
0.90[±0.05]	26.62[±1.33]	4.31[±0.22]	48.96[±2.45]	5.74[±0.29]	13.48[±0.67]

Table 7.1: The priors to use for the simple-Bayes sleep classification model. The best guesses are based on Figure 2.6, and the surrounding intervals are ±5% of its value.

7.1.2 The Learning of the Simple-Bayes Model

Upon these setups corresponding tsl specification files are created. By running the tsl program on these specifications end-results for number of runs, distance, likelihood, and the difference of the last two values of the metric used are obtained. The end-results of the learning procedure are depicted in Table 7.2 for both the cases with informative prior distributions for the parameters of *SleepStage* and the cases with even priors of all the parameters. Now, the reason of termination can be determined by studying the last two values, i.e. it terminated if either the maximum number of runs is exceeded or if the difference is below the threshold.

Method:	No. of runs:	End-distance:	End-log-likelihood:	End-difference:
GD-distance	2000	6.9762e-02	-5.8693e+04	3.0385e-06
GD-likelihood	1346	1.3486e-01	-4.6808e+04	2.9104e-11
EM	2	1.4160e-01	-4.3700e+04	0

(a)

Method:	No. of runs:	End-distance:	End-log likelihood:	End-difference:
GD-distance	2000	6.6995e-02	-5.6717e+04	1.1247e-06
GD-likelihood	2000	1.3980e-01	-4.6791e+04	1.1242e-03
EM	2	1.4210e-01	-4.3699e+04	0

(b)

Table 7.2: The end-results of learning the simple-Bayes sleep classification model using in (a) the three methods and informative priors for the parameters of *SleepStage* with respect to Table 7.1 and non-informative priors for the remaining nodes, and in (b) the parameters have non-informative priors for all nodes.

Comparisons of the results in Tables 7.2 shows that the GD-distance method yields better end-results with respect to end-distance than both the GD-likelihood and the EM-algorithm.

Since GD-distance method is tailormade to suit classification problems, this method was expected to give better result for the end-distance measure; thus, the results of Chapter 6 are re-confirmed. The GD-likehood only gives a slightly better end-distance but slightly poorer end-likelihood than the EM-algorithm, which is probably due to the fact that the GD-likelihood can become trapped in a sub-optimal local maximum and the EM-algorithm simply counts frequencies.

Conversely, log-likelihood value is significantly largest for the EM-algorithm compared to GD-dist since the aim of the EM-algorithm is to maximize the log-likelihood function, i.e. adjust the parameters such that the parameters give the highest probability to the observed data. The difference between the log-likelihood values of the GD-likelihood method and of the EM-algorithm are not significant, the reason is of course that both methods attempt to maximize the same function.

The log-likelihood values in the cases with non-informative priors are generally larger than for the cases with informative priors, which is because the priors restrain the parameters to vary freely. This is also the reason that the end-distance for the GD-distance is a little larger in the case with priors. The reason for this is that the size of the training data set is not extensive enough to modify the the parameters with informative priors, and therefore the prior counts drown out the data set. The worse results are as expected since prior counts are a different type of knowledge, not represented in the data, and are not acknowledged by our evaluation metric either. The advantage of priors begins when cross validation sets are used.

Before concluding which type of priors and what kind of method are the best for training sleep classification networks, we must first evaluate the trained network by comparing the classifications produced by it with the two experts and the results from the Nightingale program.

7.1.3 Evaluation of the Simple-Bayes Model

To evaluate the learned classification networks, the networks are used to compute a scoring for each epoch with respect to the given values of the EEG frequency bands and AOG as described in Chapter 2. Thereafter, these scorings can be compared with the scorings produced by the Nightingale program and the two experts from two different sleep laboratories, [Nielsen, 1992]. The two experts have classified (scored) the epochs manually by visually looking at the raw material and using the [Rechtschaffen and Kales, 1968] manual, [Nielsen, 1992].

The scorings produced by the neural network type trained classifier, the two experts, and the Nightingale program are depicted graphically in terms of hypnograms in Figure 7.2. From this it is easily seen that our classifier does not agree completely with the two experts' scorings or with the scorings from Nightingale, but neither do the experts themselves. Furthermore, the large oscillations produced by our network shows that the single slice approach is too coarse, extension with history information is required to obtain some kind of smoothing.

Note, using a simple 3-point filter can also smooth out the osciallations.

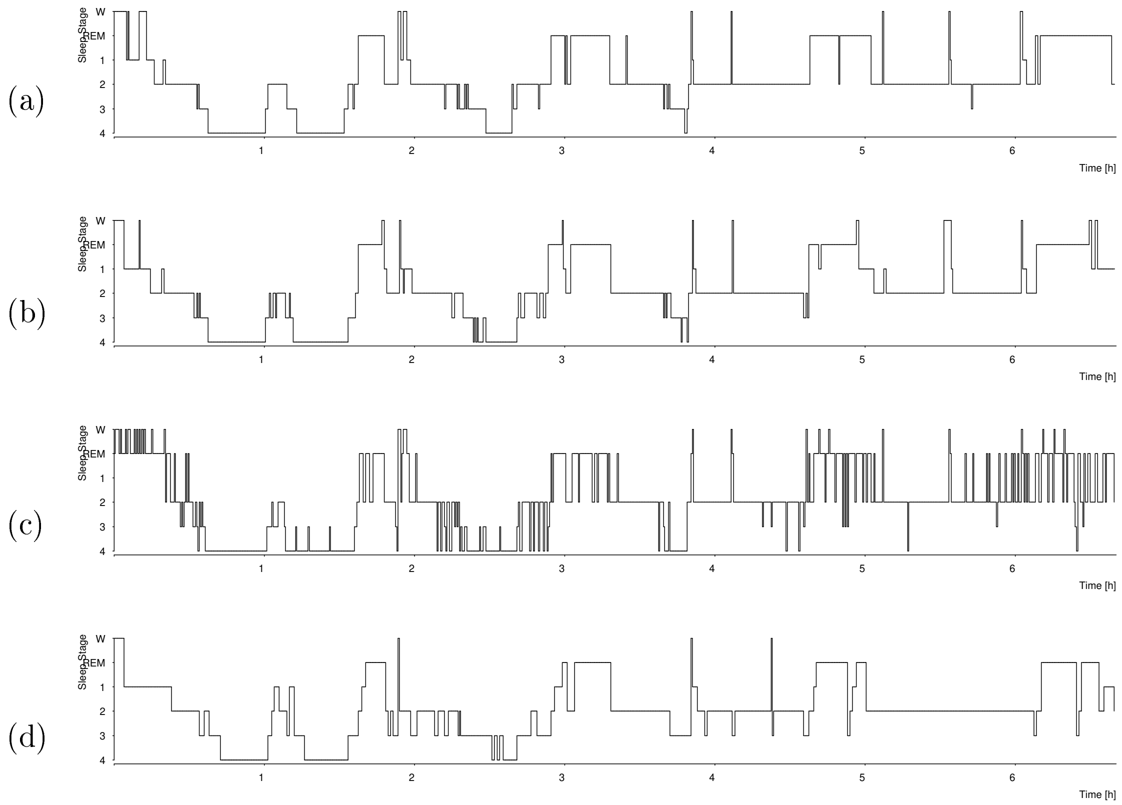

Figure 7.2: The hypnograms for the cont03 scorings produced by (a) the amd expert, (b) the gor expert, (c) the neural network type trained simple-Bayes classifier, and (d) the Nightingale classifier.

To make this comparison more quantitative, it is necessary to automatically compare the scorings epoch by epoch. This is done by creating confusion matrices for the experts versus each other and for each expert versus our classifier. A **confusion matrix** is a comparison between two series of scorings for a data set (corresponding to one patient) for each sleep stage, which the expert has scored; it shows how our classifier has scored for same epochs. From this it can be evaluated how accurate the classifier scores compared to the experts and if there are stages where the classifier scores particularly wrong.

These confusion matrices can be created by the tsl program by using its evaluation mode. The confusion matrices for the gradient descent trainer with distance as metric having the previously described priors for the *SleepStage* node are shown in Appendix D for all six patients (data sets). From the confusion matrices it can be concluded that the simple-Bayes classifiers have difficulties to distinguish between REM and sleep stage 2, and between sleep stage 3 and 4. Furthermore, the simple-Bayes classifier has problems with scoring sleep stage 1, which often become REM or sleep stage 2. This might be due to the too high influence the *KK*, *SS*, and *AOG* features have in a simple-Bayes network.

The overall results of the evaluation of the GD-distance, the GD-likelihood, and the EM-algorithm are depicted in Tables 7.3, 7.4 and 7.5, respectively, for both prior and even prior setups. The results are given in both the agreement in percent and an average penalty value per epoch. The **penalty** value for one epoch is the integer distance between the sleep stage scored by our system and the stage scored by the expert. For example, if the simple-Bayes classifier for an epoch scores sleep stage 4 whereas the expert scored sleep stage 1, the penalty value for that epoch is 3. For one night's sleep, the average penalty per epoch is found by adding the penalties of all epochs and then divide by the number of epochs in the night's sleep.

For all cases (data sets) the gradient descent algorithm with Euclidian distance as metric is the best, it always has a higher agreement percentage and always has the lowest penalty value. Surprisingly, compared to the results from Chapter 6, the gradient descent algorithm with likelihood as metric is a little better than the EM-algorithm when using priors; one would expect the two to be (almost) equal in general.

Furthermore, when the two experts (*amd* and *gor*) are agreeing in their scorings, the simple-Bayes classifier is doing about 10 percent points better in general, and the penalty values are similarly smaller. Also here, the GD-distance is the best.

For each of the three methods it holds that the classifiers on the training data sets in average is better than on the cross-validation data sets, which can be seen in both the penalty values and the agreements. This is not surprising since the classifiers had been trained (probably over-trained) with the training data sets. But on the other hand, the two classifiers trained with the gradient descent methods do a little better on the cross-validation data set when the classifiers are trained with priors than without. The priors used taken from the literature and are based on a large number of cases. Using such extra knowledge, not represented in the database, makes it possible to obtain better scores on cross-validation sets, because the virtual counts from the priors can be interpreted as an increased training set.

Finally, a comparison between the evaluation results of the tsl methods and the evaluation results of the Nightingale program. The overall results of the Nightingale program are included in Table 7.6. Comparing this with the tables for the simple-Bayes classifier shows that the neural network type trainer with and without priors is significantly better than the Nightingale classifier; also on the cross-validation set a higher amount of agreement is obtained by the simple-Bayes classifier compared to the Nightingale classifier. The EM-algorithm and the GD-likelihood methods are only approximately as good as Nightingale when priors are used, otherwise they are inferior to Nightingale. Furthermore, the results of the cross-validation set of the simple-Bayes classifier is approximately as good as the simple-Bayes model proposed in [Nielsen *et al.*, 1997]. The model in [Nielsen *et al.*, 1997] is about 4 (and about 1 for their model without use of KK and SS) percent point better since their model include an advanced method for detecting KK and SS. In [Nielsen *et al.*, 1997] the Jack-knife principle has been used for evalutation, i.e. each case has been use for evaluation such that when e.g. case 1 is evaluated, the network has been trained with the remaining data cases. If we had based our evaluations the same way, we expect our simple-Bayes model to perform better than the one in [Nielsen *et al.*, 1997].

Case	Epo.	Experts	Pen.	Epo.	tsl/amd	Pen.	tsl/gor	Pen.	tsl/all	Pen.
1	728	68.82%	0.43	501	69.23%	0.45	73.08%	0.40	85.03%	0.23
3	800	81.62%	0.24	653	71.25%	0.45	67.00%	0.49	77.03%	0.37
4	577	53.55%	0.71	309	58.93%	0.66	63.08%	0.54	85.11%	0.26
6	627	75.92%	0.43	476	76.08%	0.40	77.67%	0.33	88.66%	0.16
9	718	86.77%	0.17	623	57.80%	0.55	59.75%	0.54	62.60%	0.78
10	788	73.48%	0.35	579	66.50%	0.49	68.78%	0.41	80.48%	0.42
full	4238	74.12%	0.37	3141	66.78%	0.50	68.19%	0.45	78.64%	0.31
train		70.97%	0.43		69.22%	0.48	70.24%	0.44	83.24%	0.26
×-val		79.82%	0.26		62.35%	0.52	64.47%	0.47	71.21%	0.39

(a)

Case	Epo.	Experts	Pen.	Epo.	tsl/amd	Pen.	tsl/gor	Pen.	tsl/all	Pen.
1	728	68.82%	0.43	501	70.05%	0.45	74.04%	0.36	86.23%	0.21
3	800	81.62%	0.24	653	71.88%	0.44	67.00%	0.49	77.18%	0.36
4	577	53.55%	0.71	309	59.45%	0.64	62.91%	0.53	85.11%	0.24
6	627	75.92%	0.43	476	76.87%	0.38	77.67%	0.34	89.08%	0.16
9	718	86.77%	0.17	623	57.38%	0.61	59.61%	0.59	62.44%	0.56
10	788	73.48%	0.35	579	66.24%	0.50	68.15%	0.43	79.97%	0.29
full	4238	74.12%	0.37	3141	67.11%	0.50	68.19%	0.46	78.80%	0.32
train		70.97%	0.43		69.91%	0.47	70.46%	0.43	86.70%	0.25
×-val		79.82%	0.26		62.02%	0.55	64.08%	0.51	70.88%	0.43

(b)

Table 7.3: Evaluation overview of the sleep classification using the simple-Bayes neural network type training. In (a) the priors in Table 7.1 are used for *SleepStage*, and in (b) even priors are used for each node. *General explanation for all the evaluation overview diagrams: The first column is the case/patient identifier, where* 1, 3, 4, *and* 6 *are used to train the network with, and* 9 *and* 10 *constitute the cross-validation set. The next four columns are the total number of epochs in the data set, the inter-agreement between the experts, the average penalty per epoch between the two experts, and the number of epochs the experts are agreeing, respectively. Each of the next two pairs of columns consists of the agreement of the scorings of the simple-Bayes classifier versus each the given expert, and the penalty of the disagreement, respectively. Likewise, the two last columns denote the agreement and the penalty value of the simple-Bayes classification against the epochs where the experts agree. The three last rows of each table contain weighted averages for the full, the training, and the cross-validation dataset.*

Case	Epo.	Experts	Pen.	Epo.	tsl/amd	Pen.	tsl/gor	Pen.	tsl/all	Pen.
1	728	68.82%	0.43	501	68.68%	0.49	70.19%	0.45	82.44%	0.30
3	800	81.62%	0.24	653	62.38%	0.62	58.75%	0.66	68.15%	0.56
4	577	53.55%	0.71	309	58.75%	0.61	58.75%	0.55	85.76%	0.24
6	627	75.92%	0.43	476	72.41%	0.44	73.21%	0.39	85.29%	0.21
9	718	86.77%	0.17	623	60.31%	0.55	61.00%	0.56	65.01%	0.50
10	788	73.48%	0.35	579	68.02%	0.47	63.20%	0.51	76.86%	0.35
full	4238	74.12%	0.37	3141	65.15%	0.53	64.06%	0.52	75.74%	0.38
train		70.97%	0.43		65.59%	0.54	65.12%	0.52	78.86%	0.35
×-val		79.82%	0.26		64.34%	0.51	62.15%	0.54	70.72%	0.43

(a)

Case	Epo.	Experts	Pen.	Epo.	tsl/amd	Pen.	tsl/gor	Pen.	TSL/all	Pen.
1	728	68.82%	0.43	501	64.15%	0.51	71.02%	0.45	80.04%	0.32
3	800	81.62%	0.24	653	57.63%	0.66	55.25%	0.70	61.72%	0.60
4	577	53.55%	0.71	309	57.02%	0.57	55.81%	0.64	75.40%	0.35
6	627	75.92%	0.43	476	74.00%	0.41	75.28%	0.38	85.08%	0.21
9	718	86.77%	0.17	623	55.29%	0.59	53.20%	0.64	58.11%	0.57
10	788	73.48%	0.35	579	60.15%	0.53	55.71%	0.61	66.15%	0.47
full	4238	74.12%	0.37	3141	61.16%	0.55	60.74%	0.57	69.63%	0.44
train		70.97%	0.43		63.00%	0.54	64.17%	0.55	74.37%	0.39
×-val		79.82%	0.26		57.83%	0.56	54.51%	0.63	61.98%	0.52

(b)

Table 7.4: Evaluation overview of the sleep classification using the gradient descent algorithm with likelihood as metric. In (a) the priors in Table 7.1 are used for *SleepStage*, and in (b) even priors are used for each node.

To answer the question if the simple-Bayes classifier is doing better than the Nightingale classifier on other than the six presented data sets, we need more data. Generally, to make an even better classifier, more and probably a lot more data is needed for the learning process.

The performance of the tsl program for gradient descent training is significant. Learning the sleep classification networks takes about five to ten minutes to load and analyze the data sets, each run takes between four and eight seconds of CPU time on a Sparc Ultra II depending on the method with about 200 parameters. When the algorithm chosen does not converge, i.e. execution of 2000 runs, the algorithm takes between $2\frac{1}{4}$ and $4\frac{1}{2}$ hours of CPU time. The evaluation takes about $2\frac{1}{2}$ minutes, of these are only 7 seconds CPU time, the rest of the processing time is used for database access. For the EM algorithm the performance here is much more attractive, only 2 runs are necessary, yielding a total CPU time of about $10\frac{1}{2}$ minutes, where the first 10 minutes are used for preparations as described above.

In short, for simple-Bayes sleep classification network the neural network type training is the best to solve the classification problem, but, the EM-algorithm is clearly the fastest to run.

Case	Epo.	Experts	Pen.	Epo.	tsl/amd	Pen.	tsl/gor	Pen.	TSL/all	Pen.
1	728	68.82%	0.43	501	65.93%	0.47	71.29%	0.43	81.84%	0.28
3	800	81.62%	0.24	653	62.62%	0.58	60.00%	0.61	67.99%	0.50
4	577	53.55%	0.71	309	54.07%	0.66	58.75%	0.60	74.43%	0.37
6	627	75.92%	0.43	476	74.96%	0.38	77.35%	0.34	87.18%	0.17
9	718	86.77%	0.17	623	57.52%	0.63	57.80%	0.64	61.96%	0.58
10	788	73.48%	0.35	579	62.69%	0.51	62.94%	0.48	72.88%	0.36
full	4238	74.12%	0.37	3141	63.00%	0.54	64.51%	0.52	73.45%	0.39
train		70.97%	0.43		64.51%	0.52	66.73%	0.50	77.31%	0.34
×-val		79.82%	0.26		60.23%	0.56	60.49%	0.56	67.08%	0.48

(a)

Case	Epo.	Experts	Pen.	Epo.	tsl/amd	Pen.	tsl/gor	Pen.	TSL/all	Pen.
1	728	68.82%	0.43	501	66.07%	0.47	71.29%	0.43	81.84%	0.28
3	800	81.62%	0.24	653	62.62%	0.58	60.00%	0.61	67.99%	0.50
4	577	53.55%	0.71	309	54.07%	0.66	58.75%	0.60	74.43%	0.37
6	627	75.92%	0.43	476	74.96%	0.38	77.35%	0.34	87.18%	0.17
9	718	86.77%	0.17	623	57.52%	0.63	57.80%	0.64	61.96%	0.58
10	788	73.48%	0.35	579	62.82%	0.50	62.94%	0.48	72.88%	0.36
full	4238	74.12%	0.37	3141	63.05%	0.54	64.51%	0.52	73.45%	0.39
train		70.97%	0.43		64.57%	0.52	66.73%	0.50	77.31%	0.34
×-val		79.82%	0.26		60.29%	0.56	60.49%	0.56	67.22%	0.48

(b)

Table 7.5: Evaluation overview of the sleep classification using the EM-algorithm. In (a) the priors in Table 7.1 are used for *SleepStage*, and in (b) even priors are used for each node.

7.2 The Constrained Model

The constrained model being examined in this section is described in detail in Section 4.2 and is depicted in Figure 7.3. First, we describe the setups of the examination, next we describe and discuss the results of the evaluation.

7.2.1 The Setups of the Evaluation of the Constrained Model

In Chapter 6 and in the examination of the simple-Bayes model in Section 7.1 we concluded that the gradient descent training method with likelihood as metric does not yield better results than if the distance metric was used and it is not significant different from the EM-algorithm, therefore, we will not bring the gradient descent method with log-likelihood as metric into focus. The constrained model being learned and evaluated in this section is depicted in Figure 7.3.

The constraint nodes, *cDA*, *cAS*, and *cDB*, are all as described in Section 4.2, see Table 4.1

Case	Epo.	Experts	Pen.	Epo.	aut/amd	Pen.	aut/gor	Pen.	aut/all	Pen.
1	728	68.82%	0.43	501	69.23%	0.48	68.27%	0.49	81.04%	0.33
3	800	81.62%	0.24	653	72.88%	0.37	71.88%	0.37	79.02%	0.27
4	577	53.55%	0.71	309	52.86%	0.68	61.87%	0.48	83.82%	0.22
6	627	75.92%	0.43	476	70.65%	0.50	75.12%	0.34	82.56%	0.25
9	718	86.77%	0.17	623	50.28%	0.77	50.97%	0.76	53.29%	0.72
10	788	73.48%	0.35	579	61.68%	0.55	63.20%	0.47	72.19%	0.39
full	4238	74.12%	0.37	3141	63.29%	0.55	65.22%	0.48	73.99%	0.38
train		70.97%	0.43		67.17%	0.49	69.55%	0.42	81.18%	0.27
×-val		79.82%	0.26		56.24%	0.66	57.37%	0.61	62.39%	0.56

Table 7.6: Evaluation overview of the sleep classification using the Nightingale program (*aut*); a similar table can be found in [Nielsen, 1992, Table 8.2], the difference is that the presented table is based on fewer data (no missing observations for the *SleepStage* node) than the one in [Nielsen, 1992].

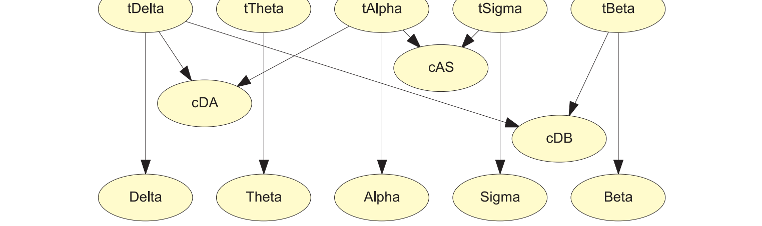

Figure 7.3: The constrained model.

on page 29. These three constraint nodes need to be instantiated to the state y, which is done by creating three new columns in the database view and setting each value of the columns to `'y'`. To do this the following SQL queries were performed on the table corresponding to the data sets:

```
alter table CONT add CDA varchar(1)
alter table CONT add CAS varchar(1)
alter table CONT add CDB varchar(1)
update CONT set CDA = 'y'
update CONT set CAS = 'y'
update CONT set CDB = 'y'
```

The first three queries add the constraint nodes to the database table *CONT*, which corresponds to the four training data sets. By simply adding the three constraint nodes to the list of input variables in the tsl specification file, the three last queries ensures that they will be instantiated automatically to state y when using the tsl program. For evaluation purposes the same queries are applied to the database table corresponding to the cross-validation sets.

Since using priors for the *SleepStage* node gives better results on the cross-validation data sets, we have chosen to use the same priors for the two setups for the training of the constrained sleep classification model:

Method	Priors	Step size	Threshold	Max. runs
GD-distance	±5%	0.1	1e-10	2000
EM-algorithm	±5%		1e-03	2000

7.2.2 The Training and Evaluation of the Constrained Model

The learning of the constrained model for sleep classification is now performed with the above setups. Table 7.7 shows the end-results of the process; the EM-algorithm could not be performed since one or more zero-sum-columns occurred during the first step of the training. A zero-sum-column occurs when no observation for a given state in a variable with children, e.g. state 4 of *Beta* exists in the database. This yields that *tBeta* cannot be in state 4 no matter what state *SleepStage* is in, this results in a zero-sum-column error, see also the explanation in Section 6.4.

Method:	No. of runs:	End-distance:	End-log-likelihood:	End-difference:
GD-dist	2000	$7.9637 \cdot 10^{-2}$	$-7.8293 \cdot 10^{4}$	$5.0348 \cdot 10^{-6}$
GD-likelihood	2000	$1.6635 \cdot 10^{-1}$	$-5.3497 \cdot 10^{4}$	$2.6509 \cdot 10^{-4}$
EM	n/a	n/a	n/a	n/a

Table 7.7: The end-results of learning the constrained sleep classification model using the three methods and priors of *SleepStage* with respect to Table 7.1. The EM-algorithm did not return successfully because of a zero-sum-column error in the network due to no observations in some configurations.

Hence, it is not possible to compare the EM-algorithm with the GD-distance method. Comparing the GD-distance with the one obtained for the simple-Bayes model, see Table 7.2, it can be concluded that the constrained model does not reach as good end-results as the simple-Bayes. Similar conclusions can be made on the evaluation of the classified constrained model, see Table 7.8.

Case	Epo.	Experts	Pen.	Epo.	tsl\amd	Pen.	tsl\gor	Pen.	tsl\all	Pen.
1	728	68.82%	0.43	501	70.33%	0.43	72.53%	0.40	85.23%	0.23
3	800	81.62%	0.24	653	69.88%	0.45	65.25%	0.50	75.50%	0.37
4	577	53.55%	0.71	309	57.89%	0.66	63.43%	0.53	85.44%	0.26
6	627	75.92%	0.43	476	74.00%	0.42	75.76%	0.36	86.34%	0.20
9	718	86.77%	0.17	623	53.20%	0.72	56.13%	0.69	58.27%	0.66
10	788	73.48%	0.35	579	66.37%	0.51	66.75%	0.45	79.27%	0.31
full	4238	74.12%	0.37	3141	65.46%	0.53	66.54%	0.49	76.95%	0.36
train		70.97%	0.43		68.41%	0.48	69.22%	0.45	82.26%	0.27
×-val		79.82%	0.26		60.09%	0.61	61.69%	0.57	68.39%	0.49

Table 7.8: Evaluation overview of the sleep classification using the neural network type training with the priors specified in Table 7.1 for *SleepStage*.

The comparisons between this result and the results of the simple-Bayes model shows that the simple-Bayes model is just a little better on both the average agreements and on the penalty values. This re-confirms statements of [Friedman *et al.*, 1997] that simple-Bayes surprisingly enough tends to be very effective. Another point could be that the absence of knowledge about physological behavior makes it hard to elicit a good qualitative structure. It might be worth investigation whether learning of structure for classification networks as in [Friedman *et al.*, 1997] could be beneficial.

Furthermore, by comparing this evaluation with a similar evaluation of the GD-distance without priors, the result would be that the model trained with priors yields better agreements compared to the experts in the evaluation. Likewise, the GD-distance method would give better results compared to the GD-likelihood method.

7.3 The Models Expanded with History Information

In this section we will consider the two models from the two previous sections, the simple-Bayes model and the constrained model, extended with history information. The possible histories to attach the models are described in Section 4.3, and here the time and the Markov chain histories will be considered. The Markov chain history extension is considered because [Nielsen *et al.*, 1997] concluded that this improved the model, whereas the time history extension is considered because we think it might improve the classification model. The time history has been chosen over the REM time history since it is much easier to implement than the REM time history extension for which is is difficult to distinguish between two REM periods. First, we examine the time history, second the Markov chain history, and third the combination of the two.

7.3.1 The Preparations and Setups

Two ways for training neural network type classification problems with history information will be discussed in this subsection; the first simply takes the entire network including the nodes representing the history information, the second separates the history part from the body part when training. Here, the first step consists of letting the history part contain parameters whereas the body part is fixed. After training the resulting probabilities are inserted into the full network and made fixed. A second run of learning is now performed where the remaining parameters are fixed, Figure 7.4 illustrates the idea for the simple-Bayes classifier. The advantage of this method is that it reduces the maximum number of parameters that have to be learned concurrently. Our feeling is that the greater $|\boldsymbol{\theta}|$ the greater is the chance of getting stuck in a sub-optimal extrema. The two-part procedure can be further developed by performing iterations of it, where the result of learning the body is fed into a new loop of learning the history part, etc. For the sake of simplicity we choose only to perform one iteration.

Note, as long as the conditional probability tables of the body part only contain uniform distributions, the body network is not required in the training process, which saves computation time. Furthermore, as evidence on the *SleepStage* node *d*-seperates the history part from the body part, training of the two parts can be performed individually for the likelihood approaches.

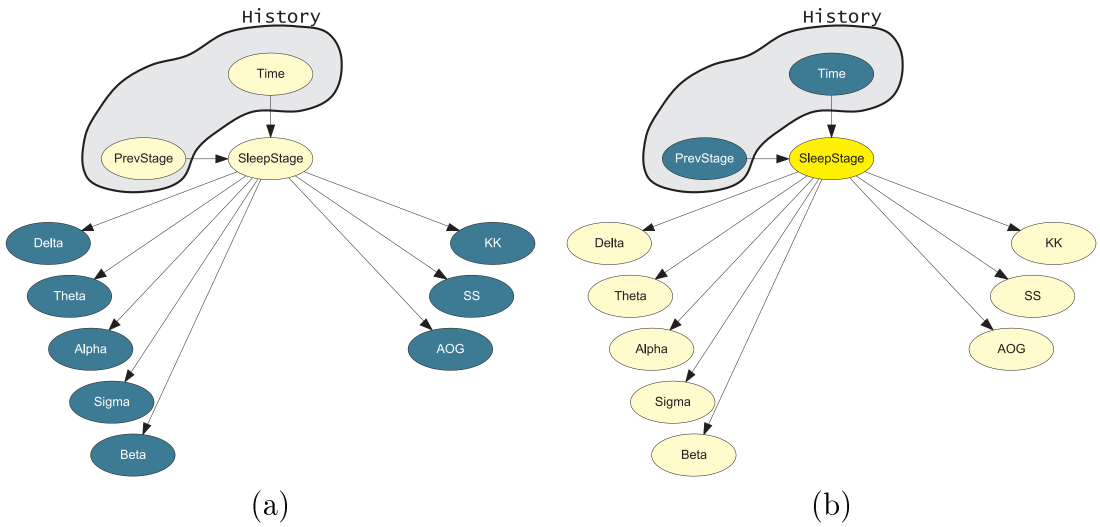

Figure 7.4: The learning process where the history part is learned separately from the body part. All the light nodes contain parameters, whereas the dark nodes are kept fixed. First, in (a) the history part is learned using only the columns from the database corresponding to the light nodes. Secondly, in (b) the history part is kept fixed with the values obtained previously, and the light nodes are then learned using only the columns from the database corresponding to these nodes and *SleepStage*.

Before any of the two methods can be applied, the training view, i.e. the view or table representing all observation of the training set, in the database have to be modified because the tsl program does not explicitly support histories. For the time history we need a *time* column in the training view; in Section 4.3 we concluded that a time interval of 90 minutes (corresponding to 180 epochs) was the most rewarding, the modifications can be performed by the following SQL statements.

```
alter table CONT add TIME integer
update CONT set TIME = 0 where EPOCH < 180
update CONT set TIME = 1 where EPOCH >= 180 and EPOCH < 360
update CONT set TIME = 2 where EPOCH >= 360 and EPOCH < 540
update CONT set TIME = 3 where EPOCH >= 540 and EPOCH < 720
update CONT set TIME = 4 where EPOCH >= 720
```

Similar modifications are performed on the testing view, *CONT_TEST*.

The modifications needed for Markov chain (MC) properties are more complicated since the modifications must be applied before the data set of each patient is merged together in *CONT* and *CONT_TEST*. The modifications consist of adding a new column for each expert containing the previous scoring — in case of missing scorings it is the previous non-missing scoring. For each patient the following SQL statements add new columns for the Markov chain properties.

```
alter table CONTxx add AMD_PREV varchar(1)
alter table CONTxx add GOR_PREV varchar(1)
```

where xx indicates the number of the patient. The next is to loop over all the epochs of each patient and update the *XXX_PREV* values. For each epoch the following SQL statements are performed for the expert, AMD, (and similarly for the expert, GOR).

```
z := '6'
for each y in epochs(xx)
  update CONTxx set AMD_PREV=z where EPOCH=y
  z := select AMD from CONTxx where EPOCH=y
endfor
```

where y is the current epoch number, z is the previous scoring, which is initially set to sleep stage wake, *W*, which corresponds to stage number *6*, and $epochs(xx)$ is the epochs of patient xx. Hereafter, the data sets of the six patients are merged together as described in Section 3.2.6. This process is very time consuming since a lot of database activity is needed.

The setups for each of the three history combinations are based on the following, except when the EM-algorithm cannot perform any results because of zero-sum-column errors.

Method	History	Way	Priors	Step size	Threshold	Max. runs
GD-distance	Time	Full	even	0.1	1e-10	2000
EM-algorithm	Time	Full	even		1e-03	2000
GD-distance	Time	Sep.	even	0.1	1e-10	2000
EM-algorithm	Time	Sep.	even		1e-03	2000
GD-distance	MC	Full	$\pm5\%$	0.1	1e-10	2000
EM-algorithm	MC	Full	$\pm5\%$		1e-03	2000
GD-distance	MC	Sep.	$\pm5\%$	0.1	1e-10	2000
EM-algorithm	MC	Sep.	$\pm5\%$		1e-03	2000
GD-distance	Time & MC	Full	$\pm5\%$	0.1	1e-10	2000
EM-algorithm	Time & MC	Full	$\pm5\%$		1e-03	2000
GD-distance	Time & MC	Sep.	$\pm5\%$	0.1	1e-10	2000
EM-algorithm	Time & MC	Sep.	$\pm5\%$		1e-03	2000

Here "Full" denotes learning with parameters in both the history and body part simultaneously whereas "Sep." denotes the two-part strategy. Notice that the priors specified for the MC history is on both the *SleepStage* and *PrevStage* nodes, whereas the prior specification in the combined history, i.e. both time and MC, only is applied on the *PrevStage* node. The priors for *SleepStage* are given in Table 7.1, and the best guesses for *PrevStage* are given in Table 4.2 with $\sigma = \pm5\%$ of the values denoted in the table. Since the time history is a new concept in the sleep classification field, no prior information exists for *Time*; though, the starting value of the parameters for *Time* could be set to those given in Figure 2.6 specified as even priors.

7.3.2 Training and Evaluation of the History Expansions

The training of the history models is now performed with respect to the setups specified previously. The end-results are depicted in Table 7.9; because of zero-sum-column error the EM-algorithm cannot be performed for the simple-Bayes model with both time and Markov chain as histories and for any of the constrained models with history components, see Section 7.2.2 and Section 6.4 for further discussions on zero-sum-column error.

The end-results in Table 7.9 show that there are no significant difference between the time and the Markov chain histories, though, the end-distances show a minor tendency that time history is a little better than the Markov chain history when the learning procedure is separated, and opposite when the full learning procedure is used. It is obvious that the combination of the two histories does not yield significant better results than using only one of the history types. Again, it shows that the constrained model generally yields poorer end-results than simple-Bayes model.

The reason that the end-distance for the neural network type training method with time history for the full network is higher than then one for the separated network in both the simple-Bayes and the constrained model is because the run ended at 2000 runs and because end-difference is higher than for e.g. the same row for Markov chain history; therefore, higher precision could be obtained by letting it proceed for some more runs. When the end-

Method	History	Way	Runs	End-distance	End-log-likelihood	End-difference
GD-dist	Time	Full	2000	1.0975e-01	-6.3832e+04	-1.9104e-03
EM	Time	Full	2	1.9271e-01	-5.0799e+04	0
GD-dist	Time	Sep.	2000	6.6667e-02	-5.5956e+04	7.6360e-07
EM	Time	Sep.	2	1.4210e-01	-4.3699e+04	7.2760e-12
GD-dist	MC	Full	2000	7.0825e-02	-6.8409e+04	1.3049e-06
EM	MC	Full	2	1.4560e-01	-4.6221e+04	0
GD-dist	MC	Sep.	2000	6.8013e-02	-6.2988e+04	1.1851e-06
EM	MC	Sep.	2	1.4731e-01	-4.3736e+04	0
GD-dist	Time & MC	Full	2000	8.6929e-02	-7.0304e+04	2.4861e-06
EM	Time & MC	Full	n/a	n/a	n/a	n/a
GD-dist	Time & MC	Sep.	2000	6.6516e-02	-5.5908e+04	6.6966e-07
EM	Time & MC	Sep.	n/a	n/a	n/a	n/a

(a)

Method	History	Way	Runs	End-distance	End-log-likelihood	End-difference
GD-dist	Time	Full	2000	1.2661e-01	-8.3880e+04	3.3239e-04
EM	Time	Full	n/a	n/a	n/a	n/a
GD-dist	Time	Sep.	2000	8.4842e-02	-7.3982e+04	4.0235e-06
EM	Time	Sep.	n/a	n/a	n/a	n/a
GD-dist	MC	Full	2000	8.0001e-02	-8.2624e+04	1.8816e-06
EM	MC	Full	n/a	n/a	n/a	n/a
GD-dist	MC	Sep.	2000	8.1548e-02	-7.5360e+04	1.9918e-06
EM	MC	Sep.	n/a	n/a	n/a	n/a
GD-dist	Time & MC	Full	2000	9.1046e-02	-8.9054e+04	3.3616e-06
EM	Time & MC	Full	n/a	n/a	n/a	n/a
GD-dist	Time & MC	Sep.	2000	7.9538e-02	-7.5565e+04	1.9312e-06
EM	Time & MC	Sep.	n/a	n/a	n/a	n/a

(b)

Table 7.9: The end-results of learning (a) the simple-Bayes model sleep classification model and (b) the constrained model using the neural network type training method (GD) and the EM-algorithm for three combinations of histories. Priors of *SleepStage* and *PrevStage* are used only when the Markov chain history is used, the priors are as in Table 7.1. The EM-algorithm did not complete successfully for the simple-Bayes model with both time and Markov chain as history and in any of the cases with the constrained model because of a zero-sum-column in the network due to no observations in that/these configurations.

difference is low it means that the precision of the end-distance of the GD method (or the end-log-likelihood for the EM-algorithm) is high, otherwise if the end-difference is high, the precision is low. The latter is the problem in the case with the GD method using time history on the full model; if the method had had permissions to run further, it would probably have ended below the corresponding value when using the Markov chain history.

Compared to both the single sliced simple-Bayes model in Section 7.1 and the constrained model in Section 7.2 the use of histories does not increase (or decrease) the quality of the end-results significantly, i.e. it does not change the end-distance and the end-log-likelihood values significantly.

The results of the evaluations of the 16 successfully trained networks are summarized in Table 7.10 by taking the weighted average over the means and penalties for the comparison between the scorings produced from the trained networks and the experts' scorings. The results here are much more significant; there is a clear tendency that the time history is much better than the Markov chain history, for the gradient descent method the time history is about 15–20 percent points better than the Markov chain history. The reason for this could be that the *PrevStage* node has too much influence on the *SleepStage* node.

The constrained model is again just a little step behind. However, the pattern has changed between the full and separated networks. Now, the separated models yields significant poorer results than the full models; further investigations are necessary to investigate the reasons.

Comparing the results in Table 7.10 with the results for the simple-Bayes model and the constrained model, a minor improvement for both the training and the cross-validation data sets can be detected when using time histories, e.g. the two-phase trained network achieves a 66.17% match on the cross-validation set whereas simple-Bayes in Table 7.3 obtained a $(62.35\%+64.47\%)/2 = 63.41\%$ match. Use of Markov chain history resulted in a major depreciation combining Markov chain with time was only able to improve these poor perfomances slightly.

When the result of the fully trained simple-Bayes classifier with time history is compared with the results of the simple-Bayes model in [Nielsen *et al.*, 1997] has gained some of the distance and is now better than their model without using *KK* and *SS*.

7.4 Summary

The results in this chapter show that the neural network type training was significantly better than the EM algorithm on the sleep classification problem. The simple-Bayes model was the model that solved the problem sovereign, though the constrained model was close to the simple-Bayes model meaning our constraint assumptions are coarse but not bad. In our opinion it should be possible to construct a model performing better than simple Bayes perhaps by exploiting structural learning algorithms for classifiers as mentioned in [Friedman *et al.*, 1997].

Plots of the obtained hypnograms showed that a single slice approach resulted in oscilla-

			Complete data		Training data		×-val data	
Method	History	Way	Mean	Pen.	Mean	Pen.	Mean	Pen.
GD	Time	Full	70.88%	0.43	74.36%	0.39	64.58%	0.50
EM	Time	Full	66.16%	0.52	68.74%	0.49	61.49%	0.57
GD	Time	Sep	69.74%	0.52	71.71%	0.46	66.17%	0.49
EM	Time	Sep	66.16%	0.52	68.74%	0.49	61.49%	0.58
GD	MC	Full	47.88%	1.01	49.07%	0.95	45.72%	1.09
EM	MC	Full	60.29%	0.55	62.52%	0.50	56.24%	0.62
GD	MC	Sep	56.21%	0.79	54.15%	0.83	59.93%	0.72
EM	MC	Sep	60.29%	0.55	62.52%	0.50	56.24%	0.62
GD	Time&MC	Full	58.57%	0.73	61.16%	0.68	53.85%	0.82
GD	Time&MC	Sep	64.52%	0.57	65.65%	0.56	62.48%	0.59

(a)

			Complete data		Training data		×-val data	
Method	History	Way	Mean	Pen.	Mean	Pen.	Mean	Pen.
GD	Time	Full	70.07%	0.47	72.95%	0.42	64.84%	0.55
GD	Time	Sep	68.46%	0.50	70.46%	0.49	64.84%	0.52
GD	MC	Full	45.19%	1.15	42.77%	1.26	49.57%	0.96
GD	MC	Sep	47.39%	0.93	47.27%	1.00	47.61%	0.81
GD	Time&MC	Full	46.25%	0.83	45.33%	0.83	47.91%	0.84
GD	Time&MC	Sep	54.41%	0.95	52.51%	1.06	57.87%	0.76

(b)

Table 7.10: A summary of the results of the evaluations of the 16 successfully trained networks, in (a) the results of the simple-Bayes model is depicted, and in (b) the the results of the constrained model. The summary is made by taking the weighted average over the means and penalties for the comparison between the scorings produced from the trained networks and the experts' scorings.

tions in the hypnogram requiring some kind of smoothing, e.g. by history information. The time history extension slightly improved the results for both models, but the Markov chain extension resulted in a too heavy smoothing dramatically decreasing quality.

The simple-Bayes classifier with and without time history extension was 8-17 percent points better than the Nightingale program when evaluating both the training and the cross-validation data sets. The classifier was almost as good as the simple-Bayes classifier proposed in [Nielsen *et al.*, 1997], the two models differed a bit, their model had a more advanced method to detect KK and SS. If the training and the cross-validation data sets in [Nielsen *et al.*, 1997] are similar, the results for the simple-Bayes model with time history on our training data set was superior to the results in [Nielsen *et al.*, 1997], the reason for this could be found in the well-considered discrete intervals of the EEG frequency bands for our classifier and in our learning method.

To make this training procedure more general and more reliable, more data is required. This is also the requirement to make a more reliable evaluation of the classifier. Furthermore, if more data was present, the *PersonType* node, which was discarded in Chapter 4, could be interesting, especially with respect to the discrete intervals of the EEG frequency bands.

Chapter 8

Conclusions

In this report we constructed a system based on Bayesian networks solving the task of classifying sleep stages from measurements of brain and eye activity.

A thorough description was given of the problem domain which lead to formulation of what we called the sleep classification task. We continued by combining domain knowledge with data analysis resulting in the formulation of two networks, the simple-Bayes model and the constrained model, both classifying each epoch independently of the other epochs. These single-slice networks were extended by incorporating different types of history extracted from the previous scorings, e.g. a Markov chain property and time.

To learn the qualitative structure of the proposed networks we developed and implemented a powerful learner for time sliced Bayesian classification networks (tsl) which combines database and expert knowledge for eliciting the conditional probability table entries in the networks. The tsl program implements three learning methods, gradient descent with likelihood metric, the EM-algorithm, and the new concept of neural network type training of Bayesian networks.

For the sleep classification task the simple-Bayes model trained using neural network type training yielded the best results, simple-Bayes is despite its simplicity surprisingly effective. Our proposed constrained model performed almost as good as the simple-Bayes. Quality of both models was slightly improved by adding the proposed time extension. Encouragingly, both systems obtained noticeably higher system-to-expert agreement percentages on the cross-validation set than Nightingale.

It would be interesting to investigate whether this trend continues for larger training and cross-validation sets based on new sleep recordings. These measurements should also contain recordings of patients with sleep disorders. In our opinion it should be possible to obtain a single-slice network outperforming simple-Bayes by applying statistical methods to learn the structure of the classifier. However, in general it is hard to obtain a good classifier for a domain where the interrater agreement between the experts trained by is about 74%.

During our experience with the Bayesian network construction cycle while developing the sleep stage classifier, tsl together with Hugin-runtime became indispensable tools for the construction phase. The tsl program is therefore an important product of this project. Our hope is that some day our prototype can be extended and integrated within Hugin enabling a more integrated tool support for this phase. During development, valuable experience was gained on how crafting of realistic expert systems is done e.g. as cooperation with an expert and the overwhelming difficulty of trying to construct a model in a domain where little is known about underlying processes and relations.

Results from both the chest clinic and the sleep classification task established neural network type training as a competitive method yielding good results for classification. We recommend that future construction of classification networks for sleep stage analysis should use this method for learning.

Appendix A

Design and Implementation of the Time Sliced Learner

This chapter covers design and implementation of a tool enabling learning by three learning methods, neural network type training, gradient descent with likelihood metric and the EM-algorithm. Embossed by the nature of the sleep classification task, the learning is performed for networks representing time sliced classification tasks. Therefore, the developed program is named tsl (Time Sliced Learner).

To evaluate the quality of the learning methods, tsl needs to be able to perform classifications based on a learned classification network and observations. In some situations it is advantageous to generate the observed cases artificially by sampling. These features are thus added to the requirement specification of tsl.

The tsl program should be able to operate in changing environments and also be designed for future extensions. We therefore decide to construct it as a general purpose learner for classification scenarios. Inspired by the expressiveness of template specification for time sliced networks combined with the suitability of gradient descent methods for learning template specified networks, we design tsl for use with template specifications.

Our intention with tsl is that it should become more than a tool merely used for calculations in our project. As pointed out in [Höhle and Kristiansen, 1998], no Bayesian network software covers all the phases of the iterative Bayesian network construction cycle, which consists of construction, analyzing and adjusting the model. Combinations of sat, [Höhle and Kristiansen, 1998], and tsl should be seen as early prototypes for the construction part of assessing the quantitative structure of the networks. We hope that future investigations properties of neural network type training could take advantage of the tsl implementation. Therefore, this chapter is written at a granularity which should enable future programmers to understand and extend the structure and functionality of tsl.

First, we design the structural framework, i.e. flow, data structures, components, etc., of the program on which all future implementations of functionality will rest. Important components are discussed individually including the tsl specification language and foundation classes. Hereafter, focus is turned to functionality where we describe how the theory of the three learning methods is implemented. Throughout the chapter the example used in Section 5.4.1 is used as an illustration.

A.1 Flow and Structure of the tsl Program

We will now look at the flow of the program seen both from the user's point of view and from inside the program, followed by a component oriented structure diagram. The philosophy behind our design is to make it as flexible as possible allowing future extensions of further learning methods, e.g. adaptation, as well as freedom of choice with respect to the DBMS controlling the data.

A.1.1 The Flow of tsl

From the user's point of view tsl is called by embedding the information about the learning setup, i.e. input and output nodes, method, parameters etc. in a so called tsl specification file. Thus, activation can be assembled in the following UNIX prompt command.

```
tsl <tsl-specfile> [-c|-C]
```

Here the `-c` option means that tsl should take advantage of generated cache for frequency counting generated at an earlier point by running tsl with the `-C` option.

The flow of tsl is now controlled by the `Controller` component, see Figure A.1. It consists of two phases, first a construction phase with the goal of creating and initializing the objects needed for running the stated method, i.e. either learning, sampling or evaluation. Secondly, an execution phase, where the selected method is executed until it terminates.

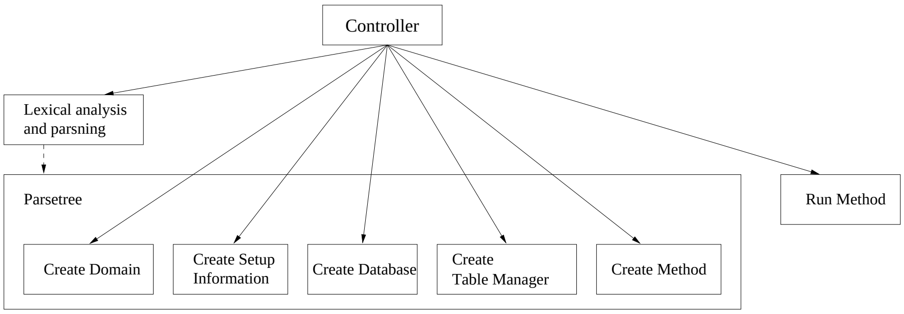

Figure A.1: The flow of the tsl program which creates the objects needed for running the method stated in the specification file.

A.1.2 The Construction Phase

To accomplish the goal of the first phase, the tsl specification file is parsed using the parse component, which results in creation of an object oriented annotated parse tree. This process happens on the basis of the tsl specification grammar, which is described in further detail in Section A.2. All future actions of the creation phase are implemented as methods traversing the parse tree. In sequence these are as follows.

1. Creation of the underlying Hugin domain based upon the template specification file. It is the responsibility of the domain to handle any expansions of the template due to time sliced structures etc.
2. Creation of a classification setup information object which turns the domain network into a classification task, i.e. information about input- and output nodes in the network and database.
3. Creation of an object handling database connectivity, extracting distinct cases, and counting frequencies.
4. Creation of a table manager which administrates and updates all parameters found in the specification file.
5. Creation of an object corresponding to the stated method.

The creation phase ends by calling the run method of `Method`, hereby we entered the second phase: the execution phase.

A.1.3 The Execution Phase

Flow in this phase is entirely controlled by the `Method` component, thus the flow of this phase differs considerably from method to method. It is therefore more interesting to consider the component-structure which forms the framework of this phase, see Figure A.2.

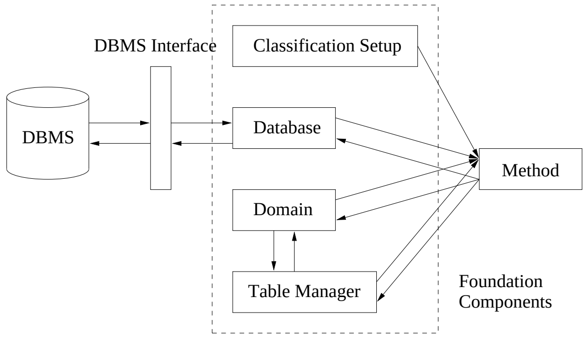

Figure A.2: The component structure of the execution phase. `Method` utilizes data structures and methods offered by the foundation components `Database`, `Domain`, `Table Manager`, and `Classification Setup`. Arrows show flow of information.

The main component is the `Method` component, which initiates all information flows. Information about the classification task presented by the network, is retrieved from the `Classification Setup` component. Cases are fetched from the `Database` component, here a layered architecture is chosen. All access to and from the DBMS is channeled through a `DBMSInterface` layer which implements the necessary fetch and write commands as DBMS specific calls. Using a different DBMS is then simply a question of exchanging this layer. For methods which rely on the existence of parameters, the `Table Manager` handles all administration of the parameters, this ranges from current values of the parameters to local computations of the gradients for each single case. These actions happen in cooperation with the `Domain` which keeps expansion of time slices transparent to the `Table Manager`.

We now proceed by describing the grammar of the tsl specification language. Hereafter, the foundation components `Domain`, `Database`, and `Table Manager` are described more explicitly. At this point all data structures and methods used by `Method` are in place for a description of `Method`.

A.2 The tsl Specification Language

This section contains a description of the tsl specification language (tslsl) used to customize and control the execution of tsl. The description is performed by describing the different productions of the LALR grammar for tslsl, see [Aho *et al.*, 1986]. Note, the complete uncommented grammar can be found in Appendix B. An example of a tslsl script is found in Figure A.3.

The language consists of two parts: settings and potentials. Here, the settings part contains information about which method to use, contains variables for establishing database connectivity, etc. Moreover, the potentials part describes which entries in the conditional probability tables are unknown, and whether we are completely ignorant about their value or have prior knowledge. Assembling the above makes the start production of the grammar as follows.

SPECFILE → SETTINGS POTENTIALS

The syntax of tslsl is inspired by an pseudo-object-oriented style, based on the syntax in e.g. Fur [Elkjær *et al.*, 1997]. Hence, we regard the two different parts as two objects, which for the settings part again consists of several smaller objects. In the following subsections, PRODUCTION denotes a production, literal a literal, and `key` a keyword as known from [Aho *et al.*, 1986].

A.2.1 The Settings Object

The settings object gives all the details for creation of the domain, classification setup, database connection, and method needed for the traversing methods of the construction phase. Variables customizing each of the foundation classes from Figure A.2 are translated into sub info-objects.

SETTINGS → `(settings` DOMAIN CP METHOD DATABASE `settings)`

```
(settings
        (domain ifile = "~/dat6/hugin/abc.net" mode = sliced)
        (cp outputvar = "B"  inputvars = "A","C")
        (gd metric    = dist
            stepsize  = 0.1
            threshold = 1E-10
            maxruns   = 2000
            logfile   = "tsl.log"
            ofile     = "~/dat6/hugin/abc-NNT.net")
        (database
                viewname  = "ABC"
                supercase = ""          subcase   = ""
                uid       = "briee"     pwd       = "*****")
settings)

(potentials
   P(B)
   {
    ( 0.5? 0.5? )
   }

   P(A | B)
   {
            (( 0.6 0.4 )   %  0
             ( 0.2 0.8 ))  %  1
   }

   P(C | B)
   {
           (( 0.9 0.1 )   %  0
            ( 0.7 0.3 ))  %  1
   }
potentials)
```

Figure A.3: An example of a tslsl file. The specification file instructs tsl to perform neural network type training in the example from Section 5.4.1, with cases fetched from the table `ABC` in the database.

The first component information object tells tsl where to find the template specification which constitutes the model of the classification problem.

DOMAIN $\rightarrow$ `(domain ifile = string mode =` MODE`)`
MODE $\rightarrow$ `sliced` | `expanded`

Here, `ifile` is the filename of the template specification constituting the network on which the method operates. The literal `string` denotes a string literal as known from standard programming languages, e.g. `"/tmp/abc.net"`. The `mode` variable tells the domain whether an expansion of the template has to be performed, see Section A.3.1 for further details.

To transform the domain into a classification problem, one has to denote which variables are input and output variables. This is done by the CP production.

```
CP          → (cp OUTPUTVAR INPUTVAR )
OUTPUTVAR   → outputvar = string | outputvar = string / string
INPUTVARS   → inputvars = NAMENOLIST
NAMENOLIST  → NAMENOITEM , NAMENOLIST | NAMENOITEM
NAMENOITEM  → string | string | float
```

Guided by our sleep classification task, we demarcate from handling general classification problems by letting OUTPUTVAR specify only one variable. It is identified by its name in the domain. In case of different names for the output variable in domain and database, we can by string/string give the name of domain and database, respectively. This feature is especially handy if each case in the database has been scored by several experts as in our sleep classification task. Here the domain variable could be *SleepStage*, where we could use either the *amd* or *gor* column in the database for training.

For input variables the domain name and database names must correspond. By use of the string|float notation one can simulate the existence of sporadic missing data in situations with fully observed cases — a feature useful for test purposes. In case of learning and evaluation the number denotes the probability with which the observation on the input variable is ignored. For the sampling method it denotes the probability that the sampled state of the variable will be replaced by a `null` value once the case is written into the database.

Choice of action for tsl is entirely determined by the METHOD production. It can be either learning, sampling, or evaluation.

```
METHOD → LEARN | SAMPLE | EVAL
LEARN  → GD | EM
```

Special attention is given to the learning methods since this is the main application of tsl. The other two methods are included for evaluating the quality of the learned network.

Specification of the two gradient descent methods, neural network type training and log-likelihood metric happens by choosing respectively `dist` or `like` as metric in the GD production. The remaining variables control the running behavior, i.e. the overall step size and the terminating condition: either the distance between two successive values of the metric is below some threshold or the number of runs is above the maximum number of runs permitted. Hence,

```
GD          → (gd metric = METRICNAME
                  stepsize = float
                  threshold = float
                  maxruns = int
                  logfile = string
                  ofile = string)
METRICNAME  → dist | likelihood
```

The `logfile` specifies the location of a logfile, where values for $\boldsymbol{\theta}^k$, $\mathrm{dist}_E(\mathbf{d})(\boldsymbol{\theta}^k)$, and $\log P(\mathbf{d})(\boldsymbol{\theta}^k)$ are written for each run k. A logfile can be imported into Splus to produce plots of the running behaviors of the learning methods. The `ofile` keyword specifies where to write the template specification file containing conditional probability tables obtained as end-result of the learning method. Specifications for the EM-algorithm follows the same pattern.

EM $\rightarrow$ `(em threshold =` float `maxruns =` int `logfile =` string `ofile =` string`)`

Sampling is included for the purpose of artificially generating cases from a known distribution. The user can control the number of cases to be sampled.

SAMPLE $\rightarrow$ `(sample cases =` int`)`

The evaluation method uses a trained network as classifier. A potential application of this feature could be to apply a classification network trained by tsl on a cross validation set. As needed by our sleep classification task, comparisons can be performed against several experts scoring, denoted by their label in the database. Results of these comparisons are combined into a report, which can be either formatted as ordinary text or LaTeX code. The latter enables direct inclusion of the results in printed reports. See Section A.3.6 for further details.

EVAL $\rightarrow$ `(eval experts =` NAMELIST `report =` string `style =` STYLE`)`
NAMELIST $\rightarrow$ string , NAMELIST | string
STYLE $\rightarrow$ `text` | `latex`

Observed cases are obtained by connecting to a database. Using the DATABASE production the user specifies how the cases are stored in the database and how to access the database. The first two variables concern the hierarchy of the data. Here `supercase` is the label of the highest abstraction of a case, e.g. for the sleep data it is the name of the patient for which we recorded an entire night's sleep. One `supercase` corresponds to observations on both input and output nodes for all time slices. Note, missing values are allowed for the input variables. The `subcase` is the label denoting one observation, i.e. one slice. In the sleep example `subcase` corresponds to an epoch.

DATABASE $\rightarrow$ `(database viewname =` string
`supercase =` string
`subcase =` string
`uid =` string
`pwd =` string`)`

The variables `uid` and `pwd` contain information on how to connect to the remote DBMS. This concludes the grammar part concerning the settings potential object.

A.2.2 The Potentials Object

The potentials object contains the conditional probability table cells found in the domain network. Here the user can denote which cells are fixed, unknown, or are unknown but with some prior information on their possible values. Our philosophy with the potentials object is that it should act on top of the conditional probability tables specified in the template specification — all tables not mentioned in the potentials object are fixed with values given by the template specification. The grammar for the potentials object is almost similar to the potential notation used in the Hugin net-specification language. Despite the small changes it is possible to simply copy paste the necessary potentials from the template specification file for use in tslsl and then perform the small changes necessary.

The potentials object consist of an optional list of tables.

```
POTENTIALS     → (potentials TABLELISTO potentials)
TABLELISTO     → ε | TABLELIST
TABLELIST      → TABLE TABLELIST | TABLE
```

Each table is specified as a simple conditional probability table. Note, if the node has any parents, they have to be in the same order as in the template specification file. The PARENTSTATE production "opens" a parent from the parentlist where as the PARENTSTATELIST production is used for producing a list of the states of the opened variable. By nesting these two productions the structure of a conditional probability table can be simulated as done in [Hugin Expert A/S, 1998]. The terminating production is STATELIST, which corresponds to a column $P(\mathcal{I}_V | i_{\mathrm{pa}(V)})$, where $i_{\mathrm{pa}(V)}$ is a specific parent configuration given by nestings of the PARENTSTATE and PARENTSTATELIST productions.

```
TABLE            → P( id | IDLIST ) { PARENTSTATE }
                   | P( id ) { PARENTSTATE }
IDLIST           → id , IDLIST | id

PARENTSTATE      → ( PARENTSTATELIST )
                   | ( STATELIST )
PARENTSTATELIST  → PARENTSTATE PARENTSTATELIST | PARENTSTATE
STATELIST        → LCELL STATELIST | LCELL
```

Here id denotes a node name as specified in [Hugin Expert A/S, 1998]. For tslsl the interesting part is specification of conditional probability learning cells, which is done by the LCELL production. Here, the user can state the nature of the cell — either fixed or parametric.

```
LCELL      → FIXED | PARAMETER
FIXED      → float | ~
PARAMETER  → NOPRIOR | PRIOR
```

For fixed cells a value has to be given; the $\sim$ keyword denotes that the value specified in the template specification should be used for this cell. For parametric cells, the user has to distinct between cells with informative and non-informative prior knowledge of their value.

NOPRIOR $\rightarrow$ float? | $\sim$?
PRIOR $\rightarrow$ float [float - float]

Thus, a parametric cell without informative prior knowledge is given by a start value and a question mark indicating that no additional knowledge about the cell exists. For cells with informative prior knowledge, the user has to specify a best guess and an interval of variation as described in Section 5.2.3. The start value of the parameter will be the best guess.

Note, for a statelist no mixture of cells with and without priors is allowed. Furthermore, if a statelist contains both parametric and fixed cells, the fixed cells are only allowed to have the value zero. This should reflect situations where the user knows for sure that for a given parent configuration some states of a variable are impossible, but is uncertain about which probabilities to assign to the remaining states. An example of this is the transition probability table from the previous sleep stage to the current. Here, it is certain that a transition from Wake to Sleep 4 would never occur.

The chosen method also puts semantic restrictions on the use of parameters. For example the EM-algorithm does not allow mixture of prior and noprior cells within the same table, similar no mixture of parametric and fixed cells is allowed for a table.

Parser implementation of tslsl

In the tsl implementation, a tslsl parser is implemented using Flex and Yacc [Levine *et al.*, 1992]. This has the advantage that the implementation becomes very flexible to future extensions of tslsl. Along the lines of [Elkjær *et al.*, 1997] an object oriented approach is taken to the parse tree construction. Each production of tslsl is converted to a corresponding class with an aggregation structure to its parts. All classes inherit from a general `parseTreeNode` class. By use of semantic actions associated with the productions a parse tree consisting of production-class instances is constructed. Traversing of the parse tree can now be performed by implementing methods on the above mentioned classes which perform the necessary local action and pass on the call to all their children.

A.3 Components

This section described the internal structures, illustrated by class diagrams, of the foundation components identified on Figure A.2. The class diagrams use the same notation as [Mathiassen *et al.*, 1993]. Furthermore, critical classes of tsl are described in additional detail by a list of relevant attributes and methods.

A.3.1 Domain

The domain component consists of two classes, the `Domain` class and the `HuginDomain` class. The `HuginDomain` class is an object-oriented version of the `h_domain_t` type in the Hugin API, and it has all the necessary methods implemented for the purpose of tsl, e.g. methods to handle nodes, their parents and children, their states, their tables, methods to set and retract evidence, methods to propagate and retrieve beliefs from the network, and methods to load and save net files, see [Hugin Expert A/S, 1998] for further information on the functionality of the Hugin API. To insert evidence retrieved from e.g. a database an appropriate structure is needed, therefore an `EvidenceItem` is defined as follows. Note, only the important attributes are included.

class `EvidenceItem`	
Attributes	
`nodeName`	The name of the node in the network.
`stateLabel`	The state label of the node corresponding to the observed state found in the evidence.
`isMissing`	Indication of whether the node is missing in the database, normally `stateLabel` will be `null` as well.

Hereby, the `EvidenceList` is a double linked list of objects of `EvidenceItem`.

The task of the `Domain` class goes a little further than this because it also maintains support for templates, which is described in Section 4.3.1. The class is as follows.

class `Domain`	
Attributes	
`domTemp`	The template Hugin network.
`dom`	The expanded Hugin network of the template network.
Methods	
`loadTemplateNet`	Loads the template network into the `Domain` class.
`expandTemplateNet`	Expands the loaded template network into the expanded network as described in Section 4.3.1.

The purpose of the class is that all functionality in `HuginDomain` should go through this class. Hence, responsibility of consulting the right node(s) in the expanded Hugin network should be placed in the methods of `Domain` before calling the methods of `HuginDomain`.

Since the Hugin API at the time of implementation did not yet contain functionality to handle templates, this feature has not been developed further. Our restriction is also motivated by the fact that when the sleep classification model does not contain a *PersonType* node, expansion of the network is unnecessary.

A.3.2 Table Manager

The `Table Manager` component is responsible for bookkeeping of the parameters placed in the conditional probability tables. This ranges from keeping track of their values, the partial derivatives, the damping values, and to perform local computations of the gradients. Figure A.4 shows the class diagram of the `Table Manager` component. The `LCell` inheritance structure stems from converting the tslsl grammar into classes. That is, the aggregation structure of a `StateList` actually contains nodes from the parse tree.

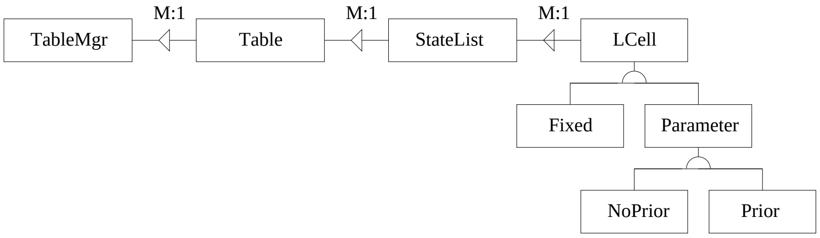

Figure A.4: Class diagram of the `Table Manager` component. Here, the numbers indicate whether the aggregation is a one- or "many"-aggregation, e.g. a `Table` instance consists of many `StateList` instances, where as each `StateList` only belongs to one `Table`.

A `StateList` corresponds to a column $P(\mathcal{I}_V|i_{\mathrm{pa}(V)})$ in the conditional probability table for a variable V. We chose the `StateList` as the level for implementing methods operating on the parameters, since e.g. calculation of priors counts and damping requires knowledge about the other variables of the statelist.

class `StateList`	
Attributes	
`params`	Array of `Parameter` node objects, i.e. all the parameters in the conditional probability table column. We do not need to keep track of any fixed values.
`noOfParams`	Number of parameters in the column.
`hasPrior`	Boolean expressing whether the state list contains parameters with non-uniform priors.
`table`	Link to the `Table` instance where the statelist is located.
`ESS`	Equivalent sample size of the Dirichlet distribution, i.e. $\alpha(i_{\mathrm{pa}(V)})$.
`alpha`	Array consisting of $\alpha(i_V, i_{\mathrm{pa}(V)}), \forall i_V \in \mathcal{I}_V$. These are the parameters of the Dirichlet prior as described Section in 5.2.3.
Methods	
`normalize`	A normalization of the current values in the statelist is performed.
`calcDamping`	Function to calculate the statelist damping based on (5.30) on page 61. The Γ-function is implemented by using code from [Press *et al.*, 1992, p. 213].
`updateGradient`	Advances each parameter in the statelist according to its damping and partial derivative.
`updateHuginTables`	Transfer the current values of all parameters in the statelist to the Hugin domain.
`localCalcCoeffs`	Perform local computations as in (5.27) and (5.26) on page 59 to compute the coefficients of $P(e)(\theta_i) = a_i\theta_i + b_i$ for each parameter θ_i present in the statelist.

Note, the `localCalcCoeffs` can, using the `table` attribute, find a clique which hosts $V \cup \mathrm{pa}(V)$. Furthermore, the calculation method could easily be adjusted to use updating of complements within the statelist, thus performing the necessary local calculations for performing a one-parameter sensitivity analysis for all variables in the statelist. See [Höhle and Kristiansen, 1998] and [van der Gaag and Kjærulff, 1999] for further details.

An instance of `Table` corresponds to a conditional probability table $P(V \mid \mathrm{pa}(V))$, here only tables containing parameters are administrated. The methods of `Table` consists mainly of call-through methods to the statelists of the table. Since the EM-algorithm operates on the table level, its functionality is implemented on `Table`. The class description of `Table` thus contains only EM specific attributes and methods.

class `Table`	
Attributes	
`node`	The Hugin API node corresponding to V.
`parents`	The parents of V as a list of node names.
`sll`	A DLList, see [George *et al.*, 1997], of `StateList` objects.
`clique`	A Hugin API clique of the junction tree, hosting $V \cup \mathrm{pa}(V)$.
`alphaCl`	A table with domain $\mathrm{cl}(V)$ holding all α parameters of the Dirichlet distributions of the individual statelists.
`expNCl`	Similar table holding the expected counts $n^*(i_{\mathrm{cl}(V)})$ as in (5.7) on page 49. These counts are increased upon propagation of each case.
Methods	
`addExpCounts`	Expected counts are added to each entry of `expNCl` based on the clique potential in `clique` after propagation of one distinct case from $\mathbf{d}$. Addition is performed by multiplying with the number of occurrences of the distinct case in $\mathbf{d}$. The method assumes that propagation of the case has already been performed before calling it.
`MStep`	The method performs the M-step of the EM-algorithm according to (5.12) on page 51. Tables are also updated in the domain.

The `TableMgr` contains a DLList of `Table` instances. Even though access to the `LCell` instances is provided through the `Table` instances, the manager contains a secondary structure of all `Parameter` objects (i.e. $\boldsymbol{\theta}$), represented as an array. This enables fast accesses in situations where an operation can be executed individually for each parameter without additional statelist knowledge, e.g. when updating the stored information in the Hugin domain. Otherwise, `TableMgr` consists mainly of call-through methods to corresponding methods in its aggregation of tables.

A.3.3 Database Connectivity

The database component has two fields of responsibility, it is responsible for the interface to the other components of tsl and communication to the local Oracle SQL database, see Figure A.2; therefore the functionality is divided into two classes, the `Database` class and the `OracleDatabase` class. Here, `OracleDatabase` is a specialization of `Database` and therefore it must implement the Oracle and the SQL specific methods, thus making it an interface to the Oracle DBMS.

The reason for choosing an external database management system as Oracle is that data are usually present in a complex structure, e.g. in a patient journal database, from which only patients with certain symptoms are extracted. Furthermore, the journals are typically already stored in a SQL database. The advantage of e.g. SQL is that it is possible to extract the needed information very easily with e.g. the view feature, i.e. creating a view of the tables with the necessary columns. Furthermore, by using this kind of database, all concurrency issues are left to the DBMS, thus enabling further learning when new cases arrive, as e.g. known from adaptation.

To access the Oracle database we use the OCI (Oracle Call Interface) libraries for C provided by Oracle. Because some of the methods that `OracleDatabase` implements are public in `Database` the choice of database is explicitly restricted to an SQL type database.

The methods of `Database` are mainly abstract methods which therefore are implemented in `OracleDatabase`, except for methods to support the creation of queries. The only relevant method left is the `calcInputs` which upon the specification of the output, the input variables, and the table name returns an `InputList`. This list constains all the distinct cases of $\mathbf{d}$ together with their count and frequencies.

The `InputList` is constructed such that each item is a distinct input configuration with respect to the specified input variables. Therefore, it contains a list of evidence which is described in Subsection A.3.1, the number of occurrences of the input configuration, and the number of occurrences of each of the output configurations given the current input configuration. Furthermore, it contains the frequency of the input configuration and an array of frequencies of each output configuration given the input configuration. This yields the following structure of an `InputItem`.

class `InputItem`	
Attributes	
`inputConfs`	The `EvidenceList` of the variables and their selected states of the input configuration of the item.
`w_count`	The number of cases in $\mathbf{d}$ with `inputConfs` as input configuration.
`w`	The frequency of `w_count` given the total count.
`Y_count`	An array containing counts of each output configuration given the input configuration, `inputConfs`.
`Y`	The frequency of `Y_count` given `w_count`.

Hereby, the `InputList` is a double linked list consisting of objects of `InputItem`.

The `OracleDatabase` contains methods to create, insert into, and drop tables. It also provides methods for performing queries in the database used by the methods in `Database`, see below. Note, the methods can also be used for general database connectivity purposes.

class `OracleDatabase`	
Methods	
`query`	Sends an SQL query to the Oracle database and returns the number of columns of the result.
`getNextQueryResultInt`	If the result of the query is one column of integers, this method returns the next integer. The user of this methods must know how many rows there will be in the result.
`getNextQueryResultString`	If the result of the query is a single column, this method returns a string containing the contents of the next row, the result must be either an integer, a numeric value, a string, or a null value.
`iterateQuery`	This method is used to process results with more than one column. For each row an `EvidenceList` containing the column name and its current value is create, then the specified callback is called with this `EvidenceList`.

The queries and results from and to the database are carried over socket connections between the Oracle server and the host running tsl. Access time to the database is very dependent of the number of users accessing the database server at the same time. Therefore, database accesses can become a bottleneck, which we try to avoid by using the database no more than necessary. For example, by keeping an internal structure of distinct input cases and creating cache for the frequency counting results minimizes the database access.

A.3.4 The `Method` component

Real functionality of tsl comes with the `Method` component which consists of an inheritance structure with an abstract super-class `Method` class enforcing the sub-classes `Learn`, `Eval`, and `Sample` to each implement a `run` method. This subsection contains a description of the three sub-classes, focusing on `Learn`.

Learner

To implement the required learning methods we design the structure of `Learn` as a flexible inheritance and callback structure as illustrated in Figure A.5.

With this structure tsl can easily be extended by incorporating different learning methods such as e.g. adaptation.

The class `Learner` contains a `learn` method which sequentially performs advancing of the parameters until a terminating condition is reached. A run of learning consists of the following steps. First, the internal structure of distinct input configurations is traversed one by one; for each configuration the callback method of the learner is called. This method entirely determines the action to be performed for each distinct data case, e.g. calculation of $\log P(\mathbf{d})$, $\mathrm{dist}_E(\mathbf{d})$, and the gradient of the metric function Secondly, once all cases are

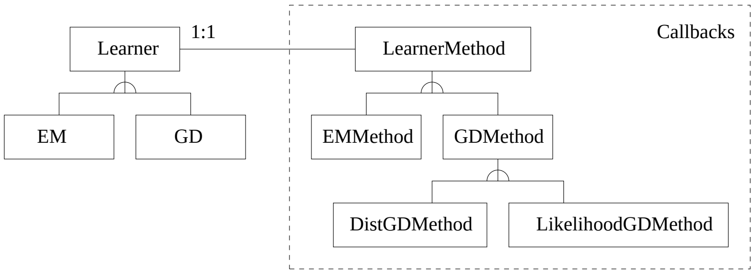

Figure A.5: The structure of the Learner `Method`.

processed we have — depending on the method — calculated the necessary gradients or tables necessary for advancing the parameters by either `MStep` or `updateGradient` on the `Table manager`. Before advancing it is checked whether the difference between the last two values of the metric are below a given `threshold` or whether the maximum number of runs is exceeded.

The core of the `LearnerMethod` callbacks is an `Inner` function as known from the Beta language [Madsen *et al.*, 1993]. Customization of the methods occurs by specialization of various hook functions as known from the Template-Hook design pattern [Gamma *et al.*, 1995].

class `LearnerMethod`	
Attributes	
`hDom`	Link to the `Domain` component.
`cp`	Link to the `Classification Setup` component.
`tableMgr`	Link to the `Table Manager` component.
`dist`	Current value of $\mathrm{dist}_E(\mathbf{d})(\boldsymbol{\theta}^k)$.
`logpE`	Value of $\log P(\mathbf{d})(\boldsymbol{\theta}^k)$.
Methods	
`Inner`	This method is called by the Learner for each distinct input configuration $i \in \mathcal{I}_I(\mathbf{d})$. By looping over all states of the output variable, calculation of both the distance and likelihood metric is performed. Additional functionality such as calculation of the gradient is obtained by specialization of various hook-functions.
Hooks	
`localIHook`	Hook function called after the evidence of i has been inserted and propagated. Useful for e.g. calculating $\partial P(i)/\partial\theta_j$.
`allVarsSetIHook`	Hook for adding functionality once we looped over all states of O. This could e.g. be calculation of the gradient.
`iorealCaseHook`	Hook for extra functionality once an (i, o) configuration is known to be a real case observed in the database. This could e.g. be counting up the number of expected cases in the EM-algorithm.

A.3.5 Sampler

The `Sampler` class is a simple class for sampling data cases, it is based on the theory and algorithm of logic sampling described in Section 5.6. Upon the specified output and input variables the `sample` method generates the specified number cases by writing them into the specified table of the database. The sampler automatically creates a new table with the given name and with columns corresponding to the output and input variables specified. If specified, the sampler also creates sporadic missing cases using the specified probability by inserting `NULL` instead of the sampled value.

A.3.6 Evaluator

The functionality of the `Evaluator` class covers methods for classifying and evaluation, the latter includes report generation for both text and LaTeX outputs. For each case (corresponding to the sub case specification) of the specified input variables in the database, the classification method, `addScorings`, uses the configuration of the case to compute the probabilities of the output variable. This is done using the specified Bayesian network and the Hugin inference engine. The state with the highest probability is chosen as output class — corresponding to a max propagation — and inserted in the database for the current configuration in the column corresponding to the output variable.

For each case, specified by the super case specification, the evaluation method, `completeEval`, creates confusion matrices for the inter-agreements of the experts specified, for the agreement between the previously performed classifications (scorings) and each of the experts, and for the agreement between the performed classifications and agreeing experts, i.e. agreements are only compared when the experts agree. The important results from these confusion matrices are in the report summarized in a table such that the results are easily read, and a similar table is included for the Nightingale scorings. The confusion matrices and the output of the evaluation is described further in Chapter 7.

Note, the use of histories requires preliminary modifications to the database in terms of inserting new columns for time and/or previous stage. Generally, the database requires preparations in term of columns containing the experts' scorings and the scorings of Nightingale used for the comparisons.

A.4 Running tsl on a Tiny Example

As a short test we now apply tsl on the scriptfile shown in Figure A.3 where the database `ABC` consists of the one case mentioned in Section 5.4.1. This yields

```
==================== TableManager =============
1: 0.526929,    0.269286,       0.1
2: 0.473071,    -0.269286,      0.1
===============================================
#1:Distance = 0.0847751  log P(d) = -1.30933
''Metric'' = 0.0847751 (diff: 998.915).
```

Here the values shown by the table manager correspond to θ_1 and θ_2. The first column contains the new values, i.e. $\boldsymbol{\theta}^1$, the second column the constrained gradient, $\boldsymbol{\nabla}_{\boldsymbol{\theta}}$, and the third column the individual dampings $\boldsymbol{\delta}$. The results are equal to the ones obtained in a more analytical fashion in Section 5.4.1. The term `distance` is the value of $\text{dist}_E(\mathbf{d}, \boldsymbol{\theta}^0)$, whereas `log P(d)` is the value of $\log P(\mathbf{d}, \boldsymbol{\theta}^0)$.

Program execution is continued until the the difference between two successive values of dist_E gets below the threshold in run 298. Here,

```
==================== TableManager ==============
1: 0.999891,    3.39193e-09,    0.1
2: 0.000109287, -3.01991e-05,   0.1
================================================
#298:Distance = 1.69583e-09  log P(d) = -0.616298
''Metric'' = 1.69583e-09 (diff: 9.50135e-11).
Stop: Below threshold (1e-10).
Iterations: 298
```

A.5 Summary

In this chapter we described the design and implementation of a powerful tool, tsl, for learning conditional probability tables for networks representing classification scenarios. Thus enabling comparisons between neural network type training and likelihood based approaches on an experimental level. Besides learning, tsl contains functionality for acting as a classifier based on classification networks, thus enabling quality for e.g. our sleep classification task.

The design of the program is made in a flexible way, enabling easy extension with new methods, e.g. sensitivity analysis as known in [Höhle and Kristiansen, 1998].

Appendix B

Complete Grammar for the tsl Specification Language

This appendix contains a LALR grammar for the tsl specification language. A specification file consists of two parts: settings and potentials, these are listed in each their section. The production SPECFILE begins the specification file.

SPECFILE → SETTINGS POTENTIALS

B.1 The Settings Object

SETTINGS	→	(settings DOMAIN CP METHOD DATABASE settings)
DOMAIN	→	(domain ifile = string mode = MODE)
MODE	→	sliced | expanded
CP	→	(cp OUTPUTVAR INPUTVAR)
OUTPUTVAR	→	outputvar = string | outputvar = string / string
INPUTVARS	→	inputvars = NAMENOLIST
NAMENOLIST	→	NAMENOITEM , NAMENOLIST | NAMENOITEM
NAMENOITEM	→	string | string | float
METHOD	→	LEARN | SAMPLE | EVAL
LEARN	→	GD | EM
GD	→	(gd metric = METRICNAME stepsize = float threshold = float maxruns = int logfile = string ofile = string)
METRICNAME	→	dist | likelihood
EM	→	(em threshold = float maxruns = int logfile = string ofile = string)

```
SAMPLE     → (sample cases = int)

EVAL       → (eval experts = NAMELIST report = string style = STYLE)
NAMELIST   → string , NAMELIST | string
STYLE      → text | latex

DATABASE   → (database viewname = string
                       supercase = string
                       subcase = string
                       uid = string
                       pwd = string)
```

B.2 The Potentials Object

```
POTENTIALS        → (potentials TABLELISTO potentials)
TABLELISTO        → ε | TABLELIST
TABLELIST         → TABLE TABLELIST | TABLE
TABLE             → P( id | IDLIST ) { PARENTSTATE }
                    | P( id ) { PARENTSTATE }
IDLIST            → id , IDLIST | id

PARENTSTATE       → ( PARENTSTATELIST )
                    | ( STATELIST )
PARENTSTATELIST   → PARENTSTATE PARENTSTATELIST | PARENTSTATE
STATELIST         → LCELL STATELIST | LCELL

LCELL             → FIXED | PARAMETER
FIXED             → float | ~
PARAMETER         → NOPRIOR | PRIOR
NOPRIOR           → float? | ~?
PRIOR             → float [ float - float ]
```

Appendix C

Reference Card for Factor Levels

This appendix contains the chest clinic network and a quick overview over the factors and their levels used in the chest clinic test, see Chapter 6.

Appendix D

Evaluation of the Simple-Bayes Model Using Priors

This appendix contains overview table and confusion matrices for the single-sliced simple-Bayes model trained with neural network type training, for comparisons the Nightingale overview table is included.

Case	Epo.	Experts	Pen.	Epo.	tsl\amd	Pen.	tsl\gor	Pen.	tsl\all	Pen.
1	728	68.82%	0.43	501	70.05%	0.45	74.04%	0.39	86.23%	0.21
3	800	81.62%	0.24	653	71.88%	0.44	67.00%	0.49	77.18%	0.36
4	577	53.55%	0.71	309	59.45%	0.64	62.91%	0.53	85.11%	0.24
6	627	75.92%	0.43	476	76.87%	0.38	77.67%	0.34	89.08%	0.16
9	718	86.77%	0.17	623	57.38%	0.61	59.61%	0.59	62.44%	0.53
10	788	73.48%	0.35	579	66.24%	0.50	68.15%	0.43	79.97%	0.29
full	4238	74.12%	0.37	3141	67.11%	0.50	68.19%	0.46	78.80%	0.32
train		70.97%	0.43		69.91%	0.47	70.46%	0.44	83.70%	0.25
×-val		79.82%	0.26		62.02%	0.55	64.08%	0.51	70.88%	0.43

Table D.1: Overview of tsl.

Case	Epo.	Experts	Pen.	Epo.	aut\amd	Pen.	aut\gor	Pen.	aut\all	Pen.
1	728	68.82%	0.43	501	69.23%	346	68.27%	360	81.04%	164
3	800	81.62%	0.24	653	72.88%	293	71.88%	293	79.02%	175
4	577	53.55%	0.71	309	52.86%	395	61.87%	277	83.82%	68
6	627	75.92%	0.43	476	70.65%	313	75.12%	213	82.56%	117
9	718	86.77%	0.17	623	50.28%	555	50.97%	545	53.29%	450
10	788	73.48%	0.35	579	61.68%	433	63.20%	367	72.19%	224
full	4238	74.12%	0.37	3141	63.28%	2335	65.22%	2055	73.99%	1198
train		70.97%	0.43		67.17%	0.47	69.55%	0.44	81.18%	0.25
×-val		79.82%	0.26		56.24%	0.55	57.37%	0.51	62.39%	0.43

Table D.2: Overview of aut.

Experts compared to each other on CONT = 1
Agreement 501 of 728 = 68.82 %

amd \ gor	4	3	2	1	R	W
4	89	0	0	0	0	0
3	64	23	3	0	0	1
2	15	48	249	0	0	3
1	0	0	1	0	0	0
R	0	0	40	17	123	24
W	1	2	5	3	0	17

Tester (tsl) compared to amd on CONT = 1
Agreement 510 of 728 = 70.05 %

tsl \ amd	4	3	2	1	R	W
4	85	67	26	0	0	0
3	3	9	18	0	0	2
2	1	11	246	0	29	3
1	0	0	0	0	1	0
R	0	1	23	1	162	15
W	0	3	2	0	12	8

Tester (tsl) compared to gor on CONT = 1
Agreement 539 of 728 = 74.04 %

tsl \ gor	4	3	2	1	R	W
4	147	25	6	0	0	0
3	11	18	2	0	0	1
2	7	28	248	1	3	3
1	0	0	1	0	0	0
R	2	1	36	16	116	31
W	2	1	5	3	4	10

Tester (tsl) compared to agreeing experts on CONT = 1
Agreement 432 of 501 = 86.23 %

tsl \ all	4	3	2	1	R	W
4	85	13	4	0	0	0
3	3	4	2	0	0	1
2	1	6	220	0	3	0
1	0	0	0	0	0	0
R	0	0	21	0	116	9
W	0	0	2	0	4	7

Table D.3: CONT=1.

Experts compared to each other on CONT = 3
Agreement 653 of 800 = 81.62 %

amd \ gor	4	3	2	1	R	W
4	105	2	0	0	0	0
3	17	35	11	0	0	0
2	1	28	347	15	7	5
1	0	0	4	21	0	2
R	0	0	3	27	133	9
W	0	0	6	10	0	12

Tester (tsl) compared to amd on CONT = 3
Agreement 575 of 800 = 71.88 %

tsl \ amd	4	3	2	1	R	W
4	104	45	26	0	1	0
3	3	14	23	0	5	0
2	0	4	320	3	42	1
1	0	0	0	0	0	0
R	0	0	33	18	119	9
W	0	0	1	6	5	18

Tester (tsl) compared to gor on CONT = 3
Agreement 536 of 800 = 67.00 %

tsl \ gor	4	3	2	1	R	W
4	117	38	20	0	1	0
3	5	16	18	0	6	0
2	1	11	297	24	32	5
1	0	0	0	0	0	0
R	0	0	29	39	97	14
W	0	0	7	10	4	9

Tester (tsl) compared to agreeing experts on CONT = 3
Agreement 504 of 653 = 77.18 %

tsl \ all	4	3	2	1	R	W
4	102	26	15	0	1	0
3	3	8	14	0	5	0
2	0	1	292	3	29	0
1	0	0	0	0	0	0
R	0	0	25	15	94	4
W	0	0	1	3	4	8

Table D.4: CONT=3.

Experts compared to each other on CONT = 4
Agreement 309 of 577 = 53.55 %

amd \ gor	4	3	2	1	R	W
4	45	62	37	0	0	0
3	0	10	34	0	0	0
2	0	0	130	21	1	0
1	0	0	10	8	0	0
R	0	0	25	6	107	1
W	0	0	9	61	1	9

Tester (tsl) compared to amd on CONT = 4
Agreement 343 of 577 = 59.45 %

tsl \ amd	4	3	2	1	R	W
4	51	5	0	0	0	2
3	33	9	0	0	0	0
2	60	29	142	8	32	9
1	0	0	0	1	3	0
R	0	1	8	8	100	29
W	0	0	2	1	4	40

Tester (tsl) compared to gor on CONT = 4
Agreement 363 of 577 = 62.91 %

tsl \ gor	4	3	2	1	R	W
4	42	12	3	0	0	1
3	0	27	15	0	0	0
2	3	33	198	32	13	1
1	0	0	0	1	3	0
R	0	0	22	31	90	3
W	0	0	7	32	3	5

Tester (tsl) compared to agreeing experts on CONT = 4
Agreement 263 of 309 = 85.11 %

tsl \ all	4	3	2	1	R	W
4	42	4	0	0	0	1
3	0	3	0	0	0	0
2	3	3	123	4	11	1
1	0	0	0	1	3	0
R	0	0	6	3	90	3
W	0	0	1	0	3	4

Table D.5: CONT=4.

Experts compared to each other on CONT = 6
Agreement 476 of 627 = 75.92 %

amd \ gor	4	3	2	1	R	W
4	90	20	1	0	0	0
3	7	24	7	0	0	0
2	0	16	233	3	6	2
1	0	0	7	1	0	0
R	0	0	38	3	115	1
W	0	1	28	9	2	13

Tester (tsl) compared to amd on CONT = 6
Agreement 482 of 627 = 76.87 %

tsl \ amd	4	3	2	1	R	W
4	95	13	2	0	0	0
3	6	12	4	0	0	1
2	10	13	245	7	38	15
1	0	0	0	0	0	0
R	0	0	7	1	114	21
W	0	0	2	0	5	16

Tester (tsl) compared to gor on CONT = 6
Agreement 487 of 627 = 77.67 %

tsl \ gor	4	3	2	1	R	W
4	88	22	0	0	0	0
3	5	13	5	0	0	0
2	4	26	275	5	14	4
1	0	0	0	0	0	0
R	0	0	24	6	106	7
W	0	0	10	5	3	5

Tester (tsl) compared to agreeing experts on CONT = 6
Agreement 424 of 476 = 89.08 %

tsl \ all	4	3	2	1	R	W
4	86	11	0	0	0	0
3	2	6	2	0	0	0
2	2	7	224	0	9	2
1	0	0	0	0	0	0
R	0	0	6	1	103	6
W	0	0	1	0	3	5

Table D.6: CONT=6.

Experts compared to each other on CONT = 9
Agreement 623 of 718 = 86.77 %

amd \ gor	4	3	2	1	R	W
4	71	8	1	0	0	0
3	1	27	32	0	0	0
2	0	8	271	7	3	0
1	0	0	5	15	5	1
R	0	0	4	1	129	6
W	0	0	3	10	0	110

Tester (tsl) compared to amd on CONT = 9
Agreement 412 of 718 = 57.38 %

tsl \ amd	4	3	2	1	R	W
4	12	2	4	0	0	0
3	18	1	4	0	0	0
2	50	57	277	15	30	22
1	0	0	0	0	0	1
R	0	0	4	11	109	87
W	0	0	0	0	1	13

Tester (tsl) compared to gor on CONT = 9
Agreement 428 of 718 = 59.61 %

tsl \ gor	4	3	2	1	R	W
4	12	1	5	0	0	0
3	18	1	4	0	0	0
2	42	41	298	15	33	22
1	0	0	0	0	0	1
R	0	0	9	17	104	81
W	0	0	0	1	0	13

Tester (tsl) compared to agreeing experts on CONT = 9
Agreement 389 of 623 = 62.44 %

tsl \ all	4	3	2	1	R	W
4	12	1	4	0	0	0
3	18	1	4	0	0	0
2	41	25	259	6	25	19
1	0	0	0	0	0	1
R	0	0	4	9	104	77
W	0	0	0	0	0	13

Table D.7: CONT=9.

Experts compared to each other on CONT = 10
Agreement 579 of 788 = 73.48 %

amd \ gor	4	3	2	1	R	W
4	19	62	18	1	0	0
3	0	29	56	0	0	0
2	0	2	353	9	12	0
1	0	0	1	9	0	1
R	0	0	10	14	148	9
W	0	1	4	9	0	21

Tester (tsl) compared to amd on CONT = 10
Agreement 522 of 788 = 66.24 %

tsl \ amd	4	3	2	1	R	W
4	24	13	6	0	0	0
3	11	8	6	0	0	0
2	65	63	335	1	15	6
1	0	0	0	0	4	0
R	0	1	28	10	152	26
W	0	0	1	0	10	3

Tester (tsl) compared to gor on CONT = 10
Agreement 537 of 788 = 68.15 %

tsl \ gor	4	3	2	1	R	W
4	5	22	16	0	0	0
3	5	9	11	0	0	0
2	9	62	388	3	18	5
1	0	0	0	0	4	0
R	0	1	26	37	131	22
W	0	0	1	2	7	4

Tester (tsl) compared to agreeing experts on CONT = 10
Agreement 463 of 579 = 79.97 %

tsl \ all	4	3	2	1	R	W
4	5	5	5	0	0	0
3	5	5	6	0	0	0
2	9	18	324	0	10	4
1	0	0	0	0	4	0
R	0	1	17	9	127	15
W	0	0	1	0	7	2

Table D.8: CONT=10.

Appendix E

Linear Modeling of Signals

This chapter contains a short introduction to the mathematical theory behind linear modeling, especially autoregressive processes, and is mainly based on [Diggle, 1992] and [Priestley, 1981]. The objective of this chapter was to investigate whether knowledge about the underlying signal processing could help improve our models. Unfortunately this was not the case.

By linear modeling we want to obtain a smooth and stable estimate for the power spectrum of the raw EEG signal $\{Z_t\}$, this is done by linear filtering. A **linear filter** is the transformation of one random sequence $\{X_t\}$ into the random sequence $\{Y_t\}$ by use of

$$Y_t = \sum_{j=-\infty}^{\infty} a_j X_{t-j}. \tag{E.1}$$

The relationship between the power spectrum of $\{X_t\}$ and $\{Y_t\}$ is given by

$$f_Y(\omega) = |a(\omega)|^2 f_X(\omega), \tag{E.2}$$

where $a(\omega) = \sum_{j=-\infty}^{\infty} a_j e^{-(j\omega)i}$ is called the **transfer function** of the linear filter.

Let $\{Z_t\}$ be a zero mean white noise sequence with variance σ^2. Note that the spectrum of such a white noise sequence is given by $f_Z(\omega) = \sigma^2$, [Diggle, 1992, p. 69].

Define the **backshift operator** B by $\mathrm{B}^l Y_t \equiv Y_{t-l}$, where $l \geq 0$. Linking this to our discussion of the EEG signal the recorded EEG values $\{Z_t\}$ are assumed to originate from a white noise sequence with unknown σ^2 and $\{Y_t\}$ are the smoothed values obtained by filtering the original values by a linear filter.

An **autoregressive process** of order p, AR(p), is defined by the difference equation

$$\phi(\mathrm{B}) Y_t = Z_t \tag{E.3}$$

where $\phi(z)$ is a polynomial with degree p with all roots outside the unit circle, the latter ensures that the process is stationary.

$$\phi(z) = 1 - \sum_{l=1}^{p} \alpha_l z^l$$

In other words, AR(p) is a finite linear filter transforming $\{Y_t\}$ into $\{Z_t\}$ by transfer function $a(\omega) = \phi(e^{-i\omega})$. Our interest is in $f_Y(\omega)$, i.e. the expression of the smoothed power spectrum. By application of (E.2) we obtain

$$f_Z(\omega) \;=\; |\phi(e^{-i\omega})|^2 f_Y(\omega) \quad\Leftrightarrow\quad f_Y(\omega) \;=\; |\phi(e^{-i\omega})|^{-2}\sigma^2, \tag{E.4}$$

where $f_Z(\omega) = \sigma^2$.

By direct calculation (E.4) can be rephrased as, [Diggle, 1992, p. 74],

$$f_Y(\omega) = \sigma^2 \left[\left\{ 1 - \sum_{l=1}^{p} \alpha_l \cos(l\omega) \right\}^2 + \left\{ \sum_{l=1}^{p} \alpha_l \sin(l\omega) \right\}^2 \right]^{-1}. \tag{E.5}$$

This yields an estimate of the spectrum of $\{Z_t\}$ by fitting an $Y_t \sim$ AR(p) process to the data. The task is now to calculate the α_l coefficients.

An alternative understanding of this fitting is obtained by considering (E.3) as

$$Y_t \;=\; \sum_{l=1}^{p} \alpha_l Y_{t-l} + Z_t, \quad t > p. \tag{E.6}$$

That is, $\{Y_t\}$ would be explicitly given if it was not for a distortion by noise, e.g. uncertainty in the measurement equipment or due to discretization in the digital measurement equipment.

Fitting a parametric model such as AR(p) to observational data involves both estimation of the parameters in the model, and selecting an appropriate order of the model. Thus, for AR(p) we need to find estimates for $\alpha_1, \ldots, \alpha_p, \mathbb{V}(Z_t) = \sigma^2$ and determine the order p.

Assuming p is known it can be shown that the estimation problem is similar to classic linear regression which can be solved by least square approximation [Priestley, 1981, pp. 346–348]. This is the same as finding estimates for the autocorrelation function of the AR(p) process, cite.

The *autocorrelation function* of a stationary random sequence $\{X_t\}$ can be regarded as important but not complete summary of the serial dependence within $\{X_t\}$ and is defined as

$$\rho_k = \frac{\operatorname{cov}(X_t, X_{t-k})}{\operatorname{cov}(X_t, X_0)} \tag{E.7}$$

Note, $\rho_k = \rho_{-k}$ and $(X_t \perp\!\!\!\perp X_{t-k} \;\Rightarrow\; \rho_k = 0)$.

Multiplication by $\mathrm{B}^k\, Y_t = Y_{t-k}$, $k \geq 0$, on both sides of (E.3) yields

$$Y_{t-k}(\phi(B)Y_t) \;=\; Y_{t-k}Z_t \tag{E.8}$$

For $k > 0$, Z_t and Y_{t-k} are independent as Y_{t-k} only depends on the current and past values of Z_t. Hence, $\operatorname{cov}(Y_{t-k}, Z_t) = 0 \Leftrightarrow \mathbb{E}[Y_{t-k}Z_t] = \mathbb{E}[Y_{t-k}]\mathbb{E}[Z_t]$. Since we assumed $\{Z_t\}$ to be zero mean we have $\mathbb{E}[Y_{t-k}Z_t] = 0$.

For $k = 0$ we have $\mathbb{E}[Y_{t-k}Z_t] = \mathbb{E}[Y_t Z_t]$. Inserting the expression for Y_t from (E.6) and applying the same independence considerations as before yields $\mathbb{E}[Y_t Z_t] = \mathbb{E}[Z_t^2] = \sigma^2$, because $\mathbb{V}[Z_t] = \mathbb{E}[Z_t^2] - (\mathbb{E}[Z_t])^2$.

Thus, taking expectation on both sides of (E.8) and dividing by $\mathrm{cov}(Y_t, Y_0)$ yields

$$
\begin{aligned}
\mathbb{E}[Y_{t-k}Y_t] - \sum_{l=1}^{p} \alpha_l \mathbb{E}[Y_{t-k}Y_{t-l}] &= \mathbb{E}[Y_{t-k}Z_t] \\
\rho_k - \sum_{l=1}^{p} \alpha_l \rho_{k-l} &= \begin{cases} \sigma^2 & \text{if } k = 0 \\ 0 & \text{if } k > 0 \end{cases}
\end{aligned}
$$

By re-arranging, an expression for the autocorrelation coefficients is obtained by

$$
\rho_k = \begin{cases} \sum_{l=1}^{p} \alpha_l \rho_{k-l} + \sigma^2 & \text{if } k = 0 \\ \sum_{l=1}^{p} \alpha_l \rho_{k-l} & \text{if } k > 0 \end{cases} \tag{E.9}
$$

This equation system is known as the Yule-Walker equations, [Priestley, 1981, Section 5.4.1]. Given know<ledge of all the auto-correlation coefficients the equation system consists of p unknowns and p equations and can be solved with respect to α_i. Rephrasing (E.9) in matrix notation for $k = 1 \ldots p$ and remembering $\rho_k = \rho_{-k}$ gives

$$
\begin{bmatrix} \rho_0 & \rho_1 & \cdots & \rho_{p-1} \\ \rho_1 & \rho_0 & \cdots & \rho_{p-2} \\ \vdots & \vdots & & \vdots \\ \rho_{p-1} & \rho_{p-2} & \cdots & \rho_0 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_p \end{bmatrix} = \begin{bmatrix} \rho_1 \\ \rho_2 \\ \vdots \\ \rho_p \end{bmatrix} \tag{E.10}
$$

Least square approximation now gives that the unknown autocorrelation coefficients can be estimated by the sample autocorrelation coefficients $\hat{\rho}_k$, [Priestley, 1981, p.351]. Here

$$
\hat{\rho}_k = \frac{1}{n} \sum_{t=1}^{n-|k|} (z_t - \bar{z})(z_{t+|k|} - \bar{z})
$$

and $\bar{z} = 1/n \cdot \sum_{i=1}^{n} z_i$ is the sample mean of the series $\{z_t : t = 1, \ldots, n\}$, in our case the recorded values of the EEG signal.

Using these estimators for the ρ_i in (E.10) we are able to calculate estimates for α_i by

$$
[\hat{\rho}_{ij}] \cdot [\hat{\alpha}_i] = [\hat{\rho}_i]
$$

Since $\rho_k = \rho_{-k}$ this ensures that $[\hat{\rho}_{ij}]$ is symmetric and thus invertible. Hence, the equation system has a solution and can be solved with respect to $\hat{\alpha}$ by e.g. the Gauss-Jordan method [Fraleigh and Beauregard, 1990].

$$
[\hat{\alpha}_i] = [\hat{\rho}_{ij}]^{-1} \cdot [\hat{\rho}_i]
$$

Using $k = 0$ an estimate of σ^2 by least-square approximation is obtained by

$$\hat{\sigma}^2 \quad = \quad \hat{\rho}_0 - \sum_{l=1}^{p} \alpha_l \hat{\rho}_l$$

Now, we have the estimates $\hat{\alpha}_1, \dots, \hat{\alpha}_p$, and $\hat{\sigma}^2$ necessary for computing $f_Y(\omega)$ by (E.5). With these results at hand we are ready to perform AR(p) modeling of signals.

Bibliography

[Aho *et al.*, 1986] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers — Principles, Techniques, and Tools.* Addison-Wesley, Reading, MA, USA, 1986.

[Bangsø and Wuillemin, 1999] Olav Bangsø and Pierre-Henri Wuillemin. Top-down specification of Bayesian networks and compact representation of repetitive structures in Bayesian networks. Technical report, Aalborg University, Department of Computer Science, 1999.

[Binder *et al.*, 1997] John Binder, Daphne Koller, Stuart Russell, and Keiji Kanazawa. Adaptive probabilistic networks with hidden variables. *Machine Learning*, 29:213–244, 1997.

[Brink *et al.*, 1994] Rune Sørdal Brink, Allan Junge Hyldal, Peder Greve Jensen, Andy Juhl Jensen, Jan Rønlev Nielsen, and Esben Østergård. Sleep classification using a causal probabilistic network. Internal paper produced by project group S721, Aalborg University, Institute of Electronic Systems, 1994.

[Castillo *et al.*, 1997] Enrique Castillo, José Manuel Gutiérrez, and Ali S. Hadi. Sensitivity analysis in discrete Bayesian networks. *IEEE Transactions on Systems, Man, and Cypbernetics — Part A: Systems and Humans*, 27(4), july 1997.

[Coupé and van der Gaag, 1998a] Veerle M. H. Coupé and Linda C. van der Gaag. Practicable sensitivity analysis of Bayesian belief networks. Technical Report UU-CS-1998-10, Utrecht University, Department of Computer Science, 1998. ftp://ftp.cs.uu.nl/pub/RUU/CS/techreps/CS-1998/1998-10.ps.gz.

[Coupé and van der Gaag, 1998b] Veerle M. H. Coupé and Linda C. van der Gaag. Properties of sensitivity analyses of Bayesian belief networks. , 1998.

[DeGroot, 1985] Morris H. DeGroot. *Probability and Statistics.* Addison-Wesley, second edition, 1985. ISBN 0-201-11366-X.

[Diggle, 1992] P. Diggle. *Time Series Analysis: A Biostatistical Approach.* Oxford University Press, Oxford, 1992.

[Elkjær *et al.*, 1997] Anders Strandløv Elkjær, Michael Höhle, Kasper Overgård, and Kristian Lykkegaard Sørensen. Fur — endnu en ø i havet. Unpublished DAT2 project report, Spring 1997.

[Fraleigh and Beauregard, 1990] John B. Fraleigh and Raymond A. Beauregard. *Linear Algebra.* Addison-Wesley, Reading, MA, USA, second edition, 1990. ISBN 0-201-51418-4.

[Friedman *et al.*, 1997] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian networks classifiers. *Machine Learning*, 29:131–163, 1997.

[Gamma *et al.*, 1995] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley, 1st edition, 1995. ISBN 0-201-63361-2.

[George *et al.*, 1997] Christian D. George, Jeanette F. Jensen, Jan Justensen, Brian Kristiansen, Jesper Krogh, Daniel K. Nielsen, and Morten Post. Edith — en syntaksunderstøttende editor. Unpublished DAT2 project report, Spring 1997.

[Heckerman *et al.*, 1994] D. Heckerman, D. Geiger, and M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. Technical Report MSR-TR-94-09, Microsoft Research, Redmond, Washington, 1994.

[Höhle and Kristiansen, 1998] Michael Höhle and Brian Kristiansen. Sensitivity analysis in Bayesian networks and influence diagrams. Unpublished DAT5 project report, Fall 1998.

[Hugin Expert A/S, 1998] Hugin Expert A/S. *Hugin API 4.0 Reference Manual*, 1998. http://www.hugin.dk/ftp/doc/api-manual.ps.

[Jensen, 1996] Finn Verner Jensen. *An introduction to Bayesian Networks*. UCL Press, 1996. ISBN 1-85728-332-5.

[Jensen, 1999] Finn V. Jensen. Gradient descent training of Bayesian networks. In *Symbolic and Quantitative Approaches to Uncertainty, European Conference ECSQARU'99, London*. Lecture Notes in Computer Science, no. 1638, Springer, 1999. To appear.

[Jie Cheng, 1999] Jie Cheng. *Belief Network PowerConstructor 2.0*, 1999. http://www.cs.ualberta.ca/~jcheng/bnpchlp/index.html.

[Kemp *et al.*, 1992] Bob Kemp, Alpo Värri, Agostinho C. Rosa, Kim D. Nielsen, and John Gade. Full Specification of the EDF format. http://www.medfac.leidenuniv.nl/neurology/knf/kemp/edf/edf_spec.htm, 1992.

[Kemp, 1987] Bob Kemp. *Model-Based Monitoring of Human Sleep Stages*. PhD thesis, Leiden University, 1987.

[Kjærulf, 1995] Uffe Kjærulf. dHugin: a computational system for dynamic time-sliced Bayesian networks. *International Journal of Forecasting*, 11:89–111, 1995.

[Lauritzen and Spiegelhalter, 1988] Steffen L. Lauritzen and David J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B*, 50(2):157–224, 1988.

[Lauritzen, 1991] Steffen L. Lauritzen. The EM algorithm for graphical association models with missing data. Research Report R-91-05, Department of Computer Science, Aalborg University, Denmark, January 1991.

[Lauritzen, 1996] Stefen L. Lauritzen. *Graphical Models*. Oxford University Press, 1996. ISBN 0-19-852219-3.

[Levine *et al.*, 1992] John R. Levine, Tony Mason, and Doug Brown. *Lex and Yacc*. Nutshell Handbook. O'Reilly and Associates, Sebastopol, California, U.S.A., 2nd edition, 1992.

[Madsen *et al.*, 1993] Ole Lehrmann Madsen, Birger Møller-Pedersen, and Kristen Nygaard. *Object Oriented Programming in the BETA Programming Language*. Addison-Wesley Publishing Company, 1993.

[Mathiassen *et al.*, 1993] Lars Mathiassen, Andreas Munk-Madsen, Peter Axel Nielsen, and Jan Stage. *Objectorienteret Analyse*. Forlaget Marko, 1st edition, 1993. ISBN 87-7754-089-5.

[McLachlan and Krishnan, 1997] G. J. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. Wiley, 1997.

[Mendelson, 1989] Wallace B. Mendelson. *Human Sleep - Research and Clinical Care*. Plenum Publishing Corporation, 2nd edition, June 1989.

[Mortensen *et al.*, 1997] Birgit Mortensen, Mette Steffensen, and Niels Zeithen. Multi-pocess dynamic linear models for longitudinal data — illustrated by eeg-signals for sleep. Master's thesis, Aalborg University, Institute of Electronic Systems, Dept. of Mathematics, 1997.

[Nielsen *et al.*, 1997] K. D. Nielsen, A. Kjær, W. Jensen, T. Dyrby, L. Andreasen, J. Andersen, and S. Andreassen. Causal probabilistic network and power spectral estimation used in sleep stage classification. *Meth Inform Med*, 36:345–348, 1997.

[Nielsen, 1992] Kim Dremstrup Nielsen. *Computer Assisted Sleep Analysis*. PhD thesis, Aalborg University, Institute of Electronic Systems, Dept. of Medical Informatics & Image Analysis, 1992. ISBN 87-984421-0-4.

[Olesen *et al.*, 1992] Kristian G. Olesen, Steffen L. Lauritzen, and Finn V. Jensen. aHUGIN: A system creating adaptive causal probabilistic networks. In *Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence*, pages 223–229, San Francisco, California, 1992. Morgan Kaufmann Publishers.

[Oppenheim and Schafer, 1989] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*. Prentice Hall Signal Processing Series. Prentice Hall, Englewood Cliffs, NJ, USA, 1989.

[Parkes, 1985] J. D. Parkes. *Sleep and its Disorders*. W. B. Saunders Company, 1985. ISBN 0-7216-1858-8.

[Peter *et al.*, 1991] J.H Peter, T. Penzel, T. Podszus, and P. von Wichert. *Sleep and Health Risk*. Springer Verlag, 1991.

[Press *et al.*, 1992] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes in C.* Cambridge University Press, New York, NY, 1992.

[Priestley, 1981] M. B. Priestley. *Spectral Analysis and Time Series.* Academic Press, Inc., London, 1981.

[Rechtschaffen and Kales, 1968] Allan Rechtschaffen and Anthony Kales. *A Manual of Standardized Terminology, Techniques and Scoring System for Sleep Stages of Human Subjects.* Brain Information Service/Brain Research Institute, UCLA, 1968.

[Russel and Norvig, 1995] Stuart J. Russel and Peter Norvig. *Artificial Interlligence — A Modern Approach.* Prentice-Hall, 1995.

[Russel *et al.*, 1995] Stuart Russel, John Binder, Daphne Koller, and Keiji Kanazawa. Local learning in probabilistic networks with hidden variables. In *Proc. of the 14th IJCAI*, pages 1146–1152, Montreal, Canada, 1995.

[Shachter and Peot, 1990] Ross D. Shachter and Mark A. Peot. Simulation approaches to general probabilistic inference on belief networks. In M. Henrion, R. D. Shachter, L. N. Kanal, and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence 5*, pages 221–231, Amsterdam, 1990. North-Holland.

[Shachter, 1998] Ross D. Shachter. Bayes-ball: The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams). In Gregory F. Cooper and Serafín Moral, editors, *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 480–487, San Francisco, July 24–26 1998. Morgan Kaufmann.

[Thiesson, 1991] Bo Thiesson. (G)EM algorithms for recursive graphical association models with missing data. Master's thesis, Aalborg University, Institute for Electronic Systems, Department of Mathematics and Computer Science, 1991.

[Thiesson, 1996] Bo Thiesson. *Some Statistical Methods for Learning Probabilistic Networks.* PhD thesis, Aalborg University, Institute for Electronic Systems, Department of Mathematics and Computer Science, 1996.

[Tobaksskaderådet, 1999] Tobaksskaderådet. Fakta om danskernes rygevaner. http://www.tobaksskaderaadet.dk/fakta/default.html, June 1999.

[van der Gaag and Kjærulff, 1999] Linda C. van der Gaag and Uffe Kjærulff. Making sensitivity analysis computationally efficient. Technical report, Aalborg University, Department of Computer Science, 1999.

[West and Harrison, 1989] M. West and P. J. Harrison. *Bayesian Forecasting and Dynamic Models.* Springer, New York, 1989.

Index

abstract node, **30**
actioculogram, **6**
AOG, *see* actioculogram
autocorrelation function, 140
autoregressive process, **139**

backshift operator, **139**
basic structure, **31**
Body movement, **8**

cell, **48**
chest clinic, 71
confusion matrix, **91**
conjugate gradient method, **53**
conjugate prior, **50**

damping, **54**, 60, **60**
data acquisition factor, 73
directed link, **31**
Dirichlet distribution, **50**

E-step, **47**
EEG, *see* electroencephalogram
EEG pattern
  alpha activity, **6**
  beta activity, **6**
  delta activity, **6**
  sigma activity, **6**
  theta activity, **6**
electroencephalogram, **5**
electrooculogram, **6**
EM-algorithm, **46**
EOG, *see* electrooculogram
Euclidean distance, 54
export node, **30**

forward sampling, 67
fully observable, **46**
functional data set, **63**

generalized parameters, 62
global node, **62**
goodness-of-fit, **68**

hidden node, **30**
hidden variable, **46**, 63

incomplete observation, **46**
initial state, **33**
inport node, **30**
input configuration, **63**
input variable, **63**
inverse filtering, **15**

k-complex, **6**
Kullback-Leibler distance, 54

linear filter, **139**
linear modeling, 14
locally stationary signals, **13**
Logic Sampling, 67

M-step, **47**
Markov chain, **33**
maximum a posteriori estimate, **47**
maximum likelihood estimate, **46**
mean error, **68**
method factor, 75
Movement arousal, **8**

neural network type training, **63**
non-informative prior, **51**
number of unknowns factor, 75

observation scheme factor, 73
output variable, **63**

parameter knowledge factor, 74
penalty, **92**
pre-emphasizing slope constant, **15**

rapid eye movement, *see* sleep stage, REM

real node, **31**
reference link, **31**
reference node, **31**
referenced node, **31**
REM, *see* sleep stage, REM
REM period, **10**
REM sleep latency, **11**

sample size, **46**
self-reference link, **31**
simple nodes, **30**
simulation, 67
sleep classification problem, **12**
sleep cycle, **11**
sleep latency, **10**
sleep spindle, **6**
sleep stage
  1, 4, **8**
  2, 4, **8**
  3, 4, **9**
  4, 4, **9**
  MT, **8**
  REM, 4, **9**
  W, 4, **8**
Sporadic missing data, 73
step size, **54**
stochastic data set, **63**
strictly proper scoring rules, **54**

template, **30**
transfer function, **139**
transition probability, **33**
true distribution, 72

underlying Bayesian network, **30**

virtual sample size, **50**

zero-sum-column, **81**