Hidden Variable Models for Knowledge Discovery

Een wetenschappelijke proeve op het gebied van de Natuurwetenschappen, Wiskunde en Informatica

Proefschrift

ter verkrijging van de graad van doctor aan de Katholieke Universiteit Nijmegen, volgens besluit van het College van Decanen in het openbaar te verdedigen op maandag 3 december 2001 des namiddags om 3.30 uur precies
door

Machiel Jan Dirk Westerdijk

geboren op 8 februari 1969 te Vlaardingen
Promotor: prof. dr. C.C.A.M. Gielen
Co-promotor: dr. W.A.J.J. Wiegerinck
Manuscript committee:
  prof. dr. M. Boasson (University of Amsterdam (UvA))
  prof. dr. W. Daelemans (University of Antwerp)
  prof. dr. ir. Th.P. van der Weide (University of Nijmegen)
Contents

1 Introduction and Summary 1
1.1 Hidden Variable Models and Learning ................................ 4
  1.1.1 Hidden Variable Models .................................. 4
  1.1.2 Learning ................................................. 7
1.2 Using Hidden Variable Models for Classification Tasks ........... 9
1.3 Rule Extraction with Hidden Variable Models ..................... 11
1.4 Selecting Data Attributes for Fast and Accurate Classification .... 13

2 Deterministic Generative Models for Fast Feature Discovery 17
2.1 Introduction .................................................... 18
2.2 Standard Vector Quantisation .................................. 20
2.3 Generative Vector Quantisation ................................ 20
  2.3.1 The GVQ learning algorithm .............................. 21
2.4 Approximate association ....................................... 22
  2.4.1 Undirected graph methods ................................ 23
  2.4.2 Approximation in the Directed Graph ..................... 25
  2.4.3 Experimental comparison .................................. 26
2.5 Imposing constraints on the basic model .......................... 29
  2.5.1 Multiple feature sets and multi-valued features ........... 29
  2.5.2 Penalty constraints ....................................... 30
  2.5.3 Binary features for binary data ............................ 30
2.6 Results on Real-World Data ................................... 31
  2.6.1 Handwritten digits ....................................... 31
  2.6.2 Image Compression ....................................... 33
2.7 Relation of GVQ to other models ................................. 35
  2.7.1 The deterministic vs the noisy approach ................... 36
  2.7.2 Alternative choices for the prior distribution ........... 37
  2.7.3 Binary Lattice Vector Quantisation ......................... 37
2.8 Conclusion ...................................................... 38
2.A Belief Propagation ............................................. 38
2.B Belief Revision .................................................. 40
2.C Mean Field ..................................................... 41
3 Improving Classification with Hidden Variable Models by Sequential Constraint Optimization
3.1 Introduction .............................................. 48
3.2 Hidden Variable Models .................................. 49
  3.2.1 Linear Gaussian hidden variable models ............... 50
3.3 Sequential Constraint Optimization ....................... 51
  3.3.1 The Standard Perceptron Learning Rule ............... 54
3.4 Application to linear models and results ................. 54
  3.4.1 Application to Generative Vector Quantization ....... 54
  3.4.2 Application to Principal Component Analysis ...... 61
3.5 Discussion and Conclusions .............................. 63
3.A Derivation of the objective function for the Lagrange multipliers ................................................................................................................................. 65
3.B Solving the Quadratic Programming Problem ............. 66

4 Inducing Classification Rules with Binary Hidden Variable Models
4.1 Introduction .................................................. 68
4.2 Deterministic Binary Hidden Variable Models ............ 69
4.3 Rule Induction with Binary Hidden Variable Models .... 71
  4.3.1 Maximizing the Likelihood of the Input Distribution under Classification Constraints ...................... 72
  4.3.2 Maximizing the Likelihood of the Joint Distribution .... 75
4.4 Results ....................................................... 76
  4.4.1 Description of the data sets ............................ 76
  4.4.2 Classification Results .................................. 77
  4.4.3 Rule Extraction ......................................... 79
4.5 Discussion and Conclusions ................................. 83

5 Comparing mixture models with decision trees for case specific attribute selection
5.1 Introduction .................................................. 88
5.2 Case specific attribute selection based on information .......... 89
5.3 Estimating probabilities for CSAS .......................... 90
  5.3.1 Mixture models ......................................... 91
  5.3.2 Naïve Bayes ........................................... 94
  5.3.3 Decision trees ......................................... 94
5.4 Experimental Results ...................................... 96
  5.4.1 Comparison of the methods on UCI and Statlog databases .......................................................... 97
5.5 Discussion ................................................... 102
  5.5.1 Deterministic Annealing and CSAS .................... 102
  5.5.2 Dealing with missing values ............................ 103
Chapter 1

Introduction and Summary

Together with the increasing use of computer systems in our society, every day immense amounts of data are gathered and stored automatically. Whether at the bank, at the super-market, browsing the Internet or making a telephone call, each transaction that we make is monitored and stored in some data base. At the same time digital sensors keep track of the heart beats of a patient in a hospital, monitor the status of a pressure tank in a chemical plant, observe the explosion of a star in a distant galaxy or witness the activities at the counter of a gas station.

While more and more data is becoming available, the power to process a fixed amount of data is becoming cheaper. Nowadays, a desktop computer, with a price of a home stereo set, can process billions of data bits in a few seconds. The two developments of increasing amounts of data-sets and the reduced costs for processing these data-sets create a large potential for techniques which can reveal patterns and relationships that are hidden in the data. The extracted knowledge can be used, for example, to predict consumer spending, to classify radar images or electrocardiograms, to verify a signature, etc.

The automatic discovery of knowledge from large data-sets is the subject of this thesis. The subject will be treated using concepts and ideas from different scientific areas. Although having a large overlap, each area has its own language and focuses on different applications. To start with, the field of statistical pattern recognition applies ideas from mathematical statistics and probability theory to describe and characterize distributions of patterns. Throughout the thesis, we shall adopt the language of probability theory to formulate algorithms and models as in statistical pattern recognition. A more algorithmic or procedural approach to discover knowledge in large data sets is taken in the field of machine learning which has its roots in computer science. This approach will often be used in this thesis to solve discrete optimization problems. As a third area of research we can identify the field of neural networks. Here information processing systems are developed which are inspired by systems found in nature such as in the nervous system and in the brain. The models that are proposed in this thesis can all be interpreted as neural networks. In some cases we
shall explicitly explain models in terms of communicating neuronal units in order to understand the workings of certain learning algorithms.

Before going into more detail let us first give an idea about the basic tasks that we wish to perform. Throughout this thesis, a data set is considered to be a set of points (patterns, records) each of which is characterized by a number of attributes (fields, variables, features). The attributes represent measurable properties of an event or an object. An example of a small data set is shown in Figure 1.1. The set consists of 400 points, each of which is characterized by 2 numerical attributes $X_1$ and $X_2$. By visual inspection, we see that there is some structure in the distribution of points, namely the points are grouped into four overlapping ‘clouds’ or clusters. In a real world data base, with many more attributes, such a cluster could, for example, correspond to a group of customers with specific buying habits, or a group of images of specific handwritten digits. In the example, we used a simple clustering technique (Vector Quantisation [27]) to identify the centers of the clusters. These centers are indicated by the crosses. A second property that characterizes the data is the ‘shape’ of the clusters which is given by their covariance matrices. Describing and characterizing the distribution of data with properties as in the example is a basic data analysis task. Techniques with which such representations are found are called unsupervised learning methods. We shall use unsupervised methods to find representations in the form of hidden variable models which play a central role in the thesis. These models will be described later in the introduction.

In addition to unsupervised methods there are supervised learning methods. These can be explained as follows: Suppose that together with position attributes, each data point has a class label. Supervized methods use these labeled examples to construct a function to correctly classify new unlabeled data points. For example, in handwritten digit recognition such a function can be used to construct a function for automatic classification of handwritten ZIP codes. As an example, a set of labeled data is plot-
Chapter 1

Figure 1.2: An example of data points in two dimensions \((X_1,X_2)\) belonging to two classes shown by crosses and circles. The solid curve shows the boundary which optimally separates the two classes.

...ted in Figure 1.2. The positions of these points are the same as in Figure 1.1. Based on these examples we fitted a classification boundary between the points so that it optimally separates the two classes. A new unlabeled point can then be classified by looking at which side of the boundary the point is located. Note, that the separation cannot be perfect due to the overlap of the clusters \(i.e.,\) some examples are positioned at the wrong side of the curve. This happens in most practical situations: Because of errors or because of missing information different classes overlap so that it is not possible to construct a 100% accurate classifier. A basic problem in the design of a classifier is to find the proper level of detail of the separating curve so that it will generalize well in spite of confusing noise.

In the example of Figure 1.2, we see that the label of a point is related to the cluster to which it belongs. Hence, in this example the distribution of points is related to the shape of the classification boundary. This relation exists in many real world problems. Therefore, modeling the distribution of the data as is done with unsupervised methods may aid in the construction of a well performing classification function.

Essentially, each chapter in this thesis is concerned with one of the two problems that are sketched in the examples above, namely

1. to find compact descriptions of complicated data distributions, and
2. to use these descriptions to construct well performing classification functions.

An important additional motivation, which can be found in all chapters, has been to create methods that provide valuable insight to human experts. By this we mean that the models are designed not to operate as black boxes with a lot of parameters with complex dependencies. Rather, they are designed to be easy to understand and to interpret.

The next four sections will introduce the main topics of the thesis and will give...
a basic idea of the central problems that are involved. The following scheme gives a brief description of the topics that will be discussed in each section:

Section 1.1 Here one of the fundamental models of the thesis will be introduced, namely the *Generative Vector Quantiser*. The purpose of the model is to describe data sets in a compact manner using a small number of characteristic features. The model is a member of the class of *hidden variable models*. An essential problem here is to find a model that is optimal for a particular data-set. Section 1.1 introduces the concept of *learning* and gives an idea of the solution to the problems involved with hidden variable models.

Section 1.2 As discussed above, the second basic data analysis task is *classification*. Section 1.2 introduces the techniques that shall be used to find hidden variable models that, in addition to finding an accurate data representation, are also accurate for classification.

Section 1.3 Many complex real world classification problems can be expressed much simpler by a small set of underlying *classification rules*. Based on the ideas that are introduced in section 1.1 and section 1.2, we developed an algorithm that finds such hidden classification rules. This algorithm will be introduced in section 1.3.

Section 1.4 This section introduces the concept of *Case Specific Attribute Selection*. This concept plays a role in problems where we are not primarily interested in generic rules as in section 1.3 but in rules that are optimal for a specific individual case. For example, in medical diagnosis such a rule would correspond to a series of tests that are optimal for a specific patient. In that case, a rule should be generated on-line and will be different for different, new patients having different complaints and symptoms. For the purpose of speed, another type of hidden variable models will be used, namely Gaussian Mixture models. How these models and alternative models are applied for Case Specific Feature Selection will be explained in section 1.4.

1.1 Hidden Variable Models and Learning

This section introduces the concept of hidden variable models. These models play a central role throughout the thesis. The second part of the section introduces the problem of ‘learning’ which is the main topic of Chapter 2.

1.1.1 Hidden Variable Models

In many situations the data are given in terms of a large number of attributes. In addition, due to noise, inconsistencies, and missing information it is hard to see the relations between data points. It is, however, very well possible that a seemingly
Figure 1.3: The bar data set. Each of the 12 images consists of $6 \times 6$ pixels which can have value ‘black’ or ‘white’.

Figure 1.4: The bar features. By making combinations of these images one can form any of the images in Figure 1.3.

A noisy unstructured data set can be explained by a much simpler process which has fewer degrees of freedom. The purpose of a hidden variable model is to represent the data more efficiently in terms of a new, small set of attributes: the hidden variables.

An example of a data set that can be represented more efficiently with hidden variables is shown in Figure 1.3. The figure shows 12 images (the data points) each consisting of 36 pixels (the attributes) with a value ‘black’ or ‘white’. Clearly, the values of the pixels are not assigned randomly: If a pixel is ‘black’ then either all the pixels in the same column or the pixels in the same row are black. More careful inspection shows that it is not the pixels but the horizontal and vertical bars of Figure 1.4 which are randomly combined. Hence, the underlying process has 12 attributes (6 horizontal and 6 vertical bars) instead of 36.

The functional relationship between the data attributes and hidden variables is displayed graphically in Figure 1.5. For example, the arrows directed towards at-
Introduction

Figure 1.5: The relations between hidden variables h1, h2 and h3 and observed (data) variables X1, X2, X3, and X4 in a hidden variable model.

tribut X1 indicate that X1 is a function of the hidden variables h1, h2 and h3. Note, that a graphical model that fits the bars example would consist of 36 ‘X’ nodes and 12 ‘h’ nodes.

Many techniques for data analysis seek for a description of data in terms of more elementary variables. An advantage of such a representation is that it reduces redundancy in the input patterns [3]. In addition, it can be helpful in understanding the processes that generated the data. Examples, where the observed data can be reduced to more elementary hidden variables, can be found in biological modeling (genes instead of the nucleotides A, C, G and T), image processing (bars instead of pixels) and data mining (customer profiles instead of contents of market baskets). In the bar example, a bar is either present or absent. In contrast, in most techniques it is assumed that the hidden variables are continuous. Examples are Principal Component Analysis [30] and Factor Analysis [5]. Effectively, these methods project the data onto a lower dimensional subspace such that most of the variance of the data is retained. A drawback is that with continuously valued hidden variables one can not identify a discrete number of clusters in the data.

More recently, hidden variable models have been proposed with which cluster structures can be identified [23, 60]. In these models the hidden variables have integer or binary values. The model that we will study in Chapter 2, the Generative Vector Quantiser (GVQ), also falls in this category: It has binary hidden variables and numeric data variables. Hence, GVQ represents a cluster as a binary composition of elementary feature vectors like the bars in Figure 1.4. To provide an easy data interpretation, GVQ associates only a single feature composition to each data pattern. This is in contrast to probabilistic models which associate a data pattern to a distribution over compositions. In this sense, one can view the GVQ model as a special deterministic variant within the larger class of noisy, probabilistic models.

An example of a set of clusters identified with 3 hidden variables is shown in Figure 1.6. As before, each data point has 2 attributes so that the data set can be plotted in a 2 dimensional space. As can be seen the data are grouped into 8 clusters. The centers of these clusters, indicated by bold circles, can be found by adding the vectors, shown as arrows, in different combinations. Next to each cluster is shown a
Chapter 1

Figure 1.6: In GVQ the cluster centers (circles) are generated by a small set of basic vectors (arrows), which correspond to the hidden states (100), (010) and (001), respectively. The cluster center corresponding to the state (011), for example, is given by the sum of the vectors corresponding to the two states (001) and 010 (see broken lines).

binary string which contains the values of the hidden variables. The meaning of the values is inclusion ‘1’ or exclusion ‘0’ of a particular vector.

1.1.2 Learning

For a given data set, the main problem within this context is to find a hidden variable model. The quality of the model that we find depends on the search procedure that we use. For large data sets such a procedure will in general need many iterative steps before it arrives at a good solution. In analogy with biological systems, the iterative procedure of finding a good model of the observed data is referred to as learning, see Figure 1.7.

The best known learning algorithm for hidden variable models is the Expectation Maximization (EM) algorithm [16]. The EM-algorithm iterates between an Expectation step (E-step) and a Minimization step (M-step). The E-step determines which values of the hidden variables are optimal for a given data pattern. In the bars example, this corresponds to the problem of determining which combination of bars forms an image like that in Figure 1.3. If, for all patterns in the data set, these combinations are known then it is computationally straightforward to optimize the feature values (these are ‘black’ or ‘white’ in the bar example of Figure 1.4 and in the 2-dimensional example in Figure 1.6 they correspond to the coordinates of the vectors) in the M-step. In mathematical notation, the EM-algorithm can be explained as follows: Let us represent the data-points by \(d\)-dimensional vectors \(\mathbf{x}_\mu = (x_{\mu 1}, \ldots, x_{\mu d})^T\), where \(\mu \in \{1, \ldots, P\}\) indexes the data point in the data set of \(P\) points. Furthermore,
Introduction

Figure 1.7: Learning is the (iterative) process of finding the model which best characterizes the distribution of patterns using a finite data-base.

Let us represent the features (the images in Figure 1.4 or the arrows in Figure 1.6) by vectors \( \mathbf{f}_i = (f_{i1}, \ldots, f_{id})^T \). Representative clusters are formed by making binary combinations of the features. Each binary combination can be represented by a binary state vector \( \mathbf{h} = (h_1, \ldots, h_n)^T \), where \( h_i \in \{0, 1\} \). Using these binary vectors, we can form a cluster center by summation: \( \sum_{i=1}^{n} h_i \mathbf{f}_i \). To each data-point \( \mathbf{x}^\mu \) there is a cluster center which is closest. If we take the Euclidian metric to define distances, this closest cluster center has a binary state

\[
\mathbf{h}^\mu = \arg\min_{\mathbf{h}} \| \mathbf{x}^\mu - \sum_i h_i \mathbf{f}_i \|^2.
\]  

(1)

The goal of learning is to find feature vectors such that the distances between data-points and their closest cluster centers becomes as small as possible. In other words, we want to minimize

\[
\sum_{\mu=1}^{P} \| \mathbf{x}^\mu - \sum_i \mathbf{f}_i h_i^\mu \|^2,
\]  

(2)

w.r.t. the vectors \( \mathbf{f}_i \). Since the states \( \mathbf{h}^\mu \) depend on the orientation of the feature vectors \( \mathbf{f}_i \), the minimization is done iteratively using two steps in each iteration: 1) The E-step which finds binary vectors \( \mathbf{h}^\mu \) with (1) for fixed \( \mathbf{f}_i \); and 2) The M-step, which finds feature vectors \( \mathbf{f}_i \) by minimizing (2) for fixed \( \mathbf{h}^\mu \).

The basic problem in the application of the EM-algorithm is the E-step (1) i.e., there is no efficient way to determine which of the, exponentially many, hidden variable combinations is optimal for a given data point. This makes the E-step computationally intractable. A major issue in developing a learning method for the GVQ model is to find accurate and tractable approximate solutions for the E-step. In data-mining applications, where data-bases are often large and high dimensional, this issue...
becomes particularly important. One of the central issues in Chapter 2 is to develop and to experimentally compare different approximating algorithms that can do the E-step in a short time.

Summary and main conclusions of Chapter 2

In Chapter 2 we propose a model, the Generative Vector Quantiser (GVQ), which is constructed to give an efficient representation of complex data sets in terms of a small number of hidden variables. One aspect that GVQ has in common with the popular standard Vector Quantiser (VQ) [27] is that GVQ represents the data in terms of a discrete number clusters. Standard VQ does, however, not have a hidden variable representation. Other popular methods to find hidden variable representations such as Principal Component Analysis (PCA) [30] and Factor Analysis [5] are restricted to representing linear structures in the data. No such restriction holds for GVQ.

Like in PCA, GVQ is deterministic in the sense that a data point is associated to only a single hidden variable state. In contrast to non-deterministic models ([60, 31, 2]), the deterministic approach makes the application of a special class of learning algorithms particularly interesting, namely the combination of Expectation Maximization [16] with Belief Revision [51]. Chapter 2 extensively compares this algorithm with other methods such as ‘mean-field’ and ‘belief propagation’. It is found that under different circumstances different algorithms perform better.

Finally, some practical applications of GVQ will be presented, namely Handwritten Digit Analysis and Image Compression. In handwritten digit analysis the GVQ model is used to find the features or basic building blocks of handwritten digits. It will be demonstrated that with a small number of features a GVQ model can reconstruct a large variation of realistic handwritten digits. In image compression GVQ is compared with standard Vector Quantisation. It is demonstrated that the GVQ method can compress an image with a much smaller loss than the standard Vector Quantisation method.

1.2 Using Hidden Variable Models for Classification Tasks

Hidden variable models are perfectly suited to describe complicated data distributions. It would be of great practical use if these descriptions could assist in discriminating between data of different classes. A standard way to do this is to split up a training set of labeled examples into subsets such that the data in each subset all have the same class label. One can then train a separate hidden variable model on each of these subsets. Afterwards, a novel unlabeled data pattern can be classified by determining how well each model can reconstruct the pattern. This approach is often taken in complicated classification tasks such as speech recognition [56] or protein classification [38].

This procedure works well if each model represents the ‘true’ data distribution
of each class. With realistic data sets, however, the models will not be optimal for classification. The main reason is that the number of available data examples is limited. One therefore has an insufficient number of examples to extrapolate and to accurately predict the distribution of future data. Hence, the estimated classification function will not be optimal for classifying new data. To improve classification it may be better to use an alternative optimization criterion which directly punishes for misclassified examples. This is done in the learning algorithms for models such as support vector machines [72] and multi-layer perceptrons [9] which are famous for their high classification accuracy. On the other hand, these models, that are only optimized for classification, lack insight into underlying data generating processes. For example, in handwritten character recognition, these techniques do not find a representation of the basic shapes of characters nor of their typical features. It is therefore desirable to have models with an optimal classification performance which at the same time give a faithful representation of the distribution of the data. Section 3 investigates techniques which combine unsupervised and supervised learning to find such hidden variable models. Another advantage of this approach is that the combination of unsupervised and supervised learning allows to find a well performing classifier even if only a fraction of the training examples has a class label. The idea is that the unsupervised component finds the clusters and the supervised component labels and adjusts the clusters using only a small number of labeled examples.

**Summary and main conclusions of Chapter 3**

The topic of Chapter 3 is the application of hidden variable models in classification tasks. Hidden variable models are used often in classification tasks. Examples can be found in speech recognition [56] and protein structure classification [38]. A schematic representation of the topic of section 3 is presented in Figure 1.8.

The standard learning objective, to find a model for each class, is maximum likelihood estimation. The algorithm that is proposed in Chapter 3, Sequential Constraint Optimization (SCO), improves the standard maximum likelihood algorithm by incorporating classification constraints in the learning procedure.

Two variants of the methodology will be compared, namely 1) a learning rule which uses carefully adjusted Lagrange multipliers in which case the algorithm is similar to the Support Vector learning algorithm [72] and 2) a learning rule where the Lagrange multipliers are either 0 or 1 in which case the algorithm is equivalent to the Batch Perceptron algorithm [17].

It is shown that the first approach needs fewer steps to converge to an optimal solution. To do this each step needs more computational time than the second approach. In order to get a converging learning curve, the second approach requires a careful adjustment of a learning parameter. The first approach is less sensitive in this respect.

Finally, it will be shown that application of these approaches on a number of
real world data sets results in an increased classification performance compared to standard maximum likelihood learning.

1.3 Rule Extraction with Hidden Variable Models

An important application for automatic data analysis techniques is to extract knowledge that provides insight to human experts. An important class of these techniques presents knowledge in the form of induced classification rules. These rules relate values of data attributes to classes in the form of logic if-then rules. For example, with two data attributes $X_1$ and $X_2$ a classification rule can have the form: "IF $X_1 > \theta_1$ AND $X_2 < \theta_2$ THEN CLASS=A". The basic objective of a rule induction algorithm is to divide the attribute space into regions which contain examples of one class. In order to get rules which provide insight, the additional objective is to divide the space in as few regions (=few rules) as possible and such that the regions are described with as few conditions as possible. Well known rule induction algorithms are decision tree learning algorithms [55] and genetic rule induction algorithms [71].

As explained in section 1.1.1, there are many situations where the given data attributes are not well suited to identify groups in the data. In these situations rule induction algorithms produce poor results: either the rules are inaccurate but short and simple, or the rules are accurate but complicated with large numbers of conditions which make them incomprehensible. In contrast, hidden variable models are constructed such that the hidden variables will be optimal to identify groups. It is
therefore more sensible to represent the conditions of classification rules with the hidden variables than with the given data attributes. This is the topic of Chapter 4. In this chapter we will present techniques which can induce accurate classification rules that relate hidden variable values to classes.

Chapter 4 compares two rule induction algorithms. The objective of the first algorithm corresponds naturally to what we desire intuitively i.e., finding a model which gives an accurate description of the data distribution (identifying groups) in such a manner that all examples are classified correctly. In the algorithm this desideratum is formulated mathematically as a constraint optimization problem. The other approach is more standard. Here the class labels are simply treated as values of an additional data attribute. This can be explained shortly in the following way: As in section 1.1.2, a data point having \( d \) attributes is represented by a vector \( \mathbf{x}^\mu = (x_1^\mu, \ldots, x_d^\mu) \).

If the data point has a class label \( y^\mu \) then we can treat this label simply as an additional attribute by defining augmented data points \( \mathbf{x}^{\mu*} = (x_1^\mu, \ldots, x_d^\mu, y^\mu) \). If we do this for all data-points then we can find a GVQ model by the EM-algorithm on the augmented data set. In probabilistic modeling the distribution of the augmented data points, which include the original attributes and the class labels, is known as the \textit{joint probability distribution}.

The two approaches for rule induction are experimentally compared on four real world data sets.

\textbf{Summary and main conclusions of Chapter 4}

The topic of Chapter 4 is to use the GVQ model of Chapter 2 to extract compact classification rules from data sets. A schematic representation is given in Figure 1.9.

The learning algorithm that is proposed in this chapter is a variant of the SCO

![Figure 1.9: Chrefchrules investigates algorithms which learn hidden variable models for rule extraction. The objective is to find a small set of short rules with which most patterns can be accurately classified.](image)
algorithm of Chapter 3. In that chapter a separate model was trained for each class so that \emph{each model} is associated to a specific class. In contrast, in Chapter 4, we apply the learning algorithm to one single GVQ model. In this way \emph{each hidden state} is associated to a specific class. The algorithm finds faithful symbolic representations of the data (by which we mean that combinations of symbols have a one-to-one correspondence with separate clusters in the data) which at the same time discriminate between different classes. The rule extraction method is demonstrated on four real world tasks, namely handwritten digit recognition, satellite image recognition, plant species recognition and emotion recognition from speech. On these data-sets the SCO algorithm is compared with the more standard approaches of unsupervised learning and of joint likelihood estimation. The results show that the SCO algorithm greatly improves the classification accuracy of rules found by unsupervised learning. In addition, the results show that the SCO rules are more accurate than the rules found by joint likelihood estimation.

1.4 Selecting Data Attributes for Fast and Accurate Classification

In many applications one wishes to classify an object based on as few attribute values as possible. For example, in the medical domain, diagnosing a disease may require a series of lab tests. In such a situation, it would be beneficial if the doctor had a system that predicts which of the tests are most relevant for diagnosing the patient under examination. Other, similar applications can be found in machine fault diagnosis or in automatic help desk systems. This problem, of finding the most relevant attributes in a specific situation, shall be referred to as Case Specific Attribute Selection (CSAS).

The task of predicting class membership using the least possible amount of resources basically consists of two parts. First, the system must be able to predict class labels using incomplete information. Second, the system needs to determine which attribute should be measured next to get maximum information gain.

There are various ways to address this issue. A well known classification method which has been extensively studied within the machine learning community over the last decade is to use induced decision tree classifiers. It is also well known that decision trees have limitations in terms of classification ability. For example, the decision boundaries in a tree are generally restricted to be axis parallel (for example, parallel to the axis \(X_1\) or \(X_2\) in Figure 1.1) which may not be optimally suited for particular classification problems [46], [13]. On the other hand, there exist classifiers which are more flexible than decision trees. A 2-dimensional impression of both types of classifiers is shown in Figure 1.10. Chapter 5 formulates the use of these more flexible methods for attribute selection, and compares and contrasts their performance with decision trees.

For the task of attribute selection we are explicitly concerned with the data attributes. Finding a more efficient hidden variable representation than in the previous
Introduction

sections is therefore not relevant here. The main objective is to reduce uncertainty about the class of a given pattern. For this we need to be able to quantitatively specify degrees of uncertainty based on the available information. This can be done if we have a probability model.  

As was stated above, we want a flexible model which avoids the limitations of decision trees. However, for CSAS we have to put restrictions on the computational complexity of such a model which is related to the time it takes to compute a probability value. The reason for this is that after each measurement of an attribute value, the expected relevance of a possibly large number of attributes needs to be evaluated. This means that, for practical use, the classification system has to be able to infer even larger numbers of probability values in just a fraction of time. Therefore we use the class of factorized mixture models for the task of Case Specific Attribute Selection. In a mixture model, like in the models described in the previous sections, it is assumed that the data contain different groups or clusters. These groups are modeled with separate ‘sub’ models. The sub models are the mixture components. To acquire the necessary computational speed, it is assumed that the attributes within a mixture component are independent. This avoids the exponential scaling problem present in other models where one needs to sum over all possible value combinations of the unknown attributes. Although the attributes within a cluster are independent, attributes can be related to each other based on their cluster membership. To get an idea about this effect one can look at Figure 1.1. If the X1 value of a data point is very small then the point is likely to be part of the most left hand cluster. We see that this cluster is positioned high on the vertical axis. Hence, even though within the clusters there seems to be no strong correlation, we see that if X1 is small then X2 will be large. By adding more mixture components, a mixture model can approximate any possible probability function with arbitrary precision.

Figure 1.10: a) Decision boundaries in a decision tree have a restricted shape: axis parallel. b) With a more flexible model, where this restriction no longer applies, it is possible to model a much richer class of decision boundaries. The numbers are the indices of the most relevant attributes within each region.
Chapter 1

Figure 1.11: A decision is directly induced from the data-base. At each node in the tree a ‘yes’ or ‘no’ question needs to be answered. This question corresponds to a decision boundary in the data space. Depending on the answer one arrives at the next node until one arrives in a leaf with the class label. The positions of the decision boundaries are found by using information theory.

In Chapter 5 we shall use the framework of information theory [8] for CSAS. In this framework, attributes which have not yet been measured are ranked according to how much the entropy in the probability distribution of the classes is expected to decrease after measurement of the attribute. The attribute with largest expected decrease of entropy will be selected. In decision trees the entropies are estimated simply by counting the numbers of data points on different sides of the decision boundaries. Using a mixture model, the entropies are computed from a probability density function which has been fitted to the data. The basic differences between these approaches i.e., decision trees and mixture models, are illustrated graphically in Figure 1.11 and Figure 1.12.

In Chapter 5 we shall compare these fundamentally different methodologies on 10 realistic data-bases. These data-bases include medical problems (diagnosis of diabetes and of heart diseases), biological classification (Iris plants, families of yeasts, and ecoli bacteria), and problems such as credit card fraud.

Summary and main conclusions of Chapter 5

Chapter 5 investigates and compares different approaches to the problem of Case Specific Attribute Selection (CSAS).

Different studies [21, 32, 64, 26] have mentioned the practical relevance of techniques for CSAS. The usefulness and performance of such techniques has, however, never been evaluated nor compared. The purpose of Chapter 5 is to compare and to contrast two fundamentally different approaches to CSAS, namely decision trees [10, 55] and mixture models [68]. Decision trees are an obvious candidate for CSAS, since their basic mechanism is to sequentially select attributes to optimally divide
The data into regions of different classes. Mixture models, on the other hand, do not divide the data into different regions but rather fit a probability density function to the data. Both approaches are compared within the common theoretical framework of information theory [63].

In order to optimize a mixture model for CSAS we found that a variant of the Deterministic Annealing algorithm [58] performs very well. This algorithm adjusts the model to the optimal level of detail that is needed for a specific data set.

The methods, decision trees and mixture models, are extensively compared on a large number of real-world data sets. The results show that mixture models perform better on average.
Chapter 2

Deterministic Generative Models for Fast Feature Discovery

Abstract. We propose a vector quantisation method which does not only provide a compact description of data vectors in terms codebook vectors, but also gives an explanation of codebook vectors as binary combinations of elementary features. This corresponds to the intuitive notion that, in the real world, patterns can be usefully thought of as being constructed by compositions from simpler features. The model can be understood as a generative model, in which the codebook vector is generated by a hidden binary state vector. The model is non-probabilistic in the sense that it assigns each data vector to a single codebook vector. We describe exact and approximate learning algorithms for learning deterministic feature representations. In contrast to probabilistic models, the deterministic approach allows the use of message propagation algorithms within the learning scheme. These are compared with standard mean-field/Gibbs sampling learning. We show that Generative Vector Quantisation gives a good performance in large scale real world tasks like image compression and handwritten digit analysis with up to 400 data dimensions.

2.1 Introduction

Many techniques for data analysis can be regarded as seeking for a description of data in terms of elementary features. An advantage of a feature representation is that it reduces redundancy in the input patterns [3]. Furthermore, a description in terms of features can provide a lucid explanation of objects (input patterns), which can in addition be helpful in understanding the hidden data generating process. Areas in which feature representations are particularly relevant can be found in biological modelling, image processing and data mining.

Currently, the most widely applied techniques for feature extraction are linear. Well known examples are principal component and factor analysis. Both these techniques give a meaningful representation of the data only if the data are Gaussian distributed around some low dimensional linear subspace. More recent non-Gaussian linear methods include independent component analysis [7] and the sparse coding approach by [50]. A significant advantage of linear methods is their speed. In addition, linear models provide an easily interpretable feature representation of the data, often in terms of the basis spanning the linear subspace. One important drawback of linear models is that they can not describe multi-modal distributions.

The most well known and simplest method for finding multi-modal structure in the data is vector quantisation (VQ) [24]. The drawback of vector quantisation, however, is its lack of a feature representation. To overcome this problem, more advanced non-linear probability models have recently been promoted by several authors in the context of feature extraction [59, 23, 60, 2]. In contrast to standard vector quantisation, where a data point is explained in terms of a single codevector, these models explain a data point in terms of a combination of elementary features. Each such combination is formed or generated by the state of a set of hidden or latent variables.

The model that we propose in this chapter, the Generative Vector Quantizer (GVQ), is exactly such a generative model, with a binary hidden layer and a continuous visible layer representing the codebook vectors. Hence, in GVQ a codebook vector is considered to be composed of a binary combination of features in which a given feature is either fully present or fully absent. To provide an easy data interpretation, GVQ associates only a single codebook vector and therefore a single feature composition to each data pattern. This is in contrast with probabilistic models which associate a data pattern with a distribution over compositions where, in principle, each possible composition has a contribution. In this sense, one can view the GVQ model as a special deterministic variant within the larger class of noisy, probabilistic models.

In addition to interpretability, there is another important advantage of using a deterministic model. To learn a generative model from a given data set there exists an accurate and rapidly converging algorithm. This EM-algorithm [16] iterates between an Expectation step (E-step) and a Minimisation step (M-step). The E-step determines which hidden states (which combinations of features) are responsible for generating a given data pattern. If these states are known then it is computationally...
Chapter 2

straightforward to optimize the feature values in the M-step. The basic problem in
the application of the EM-algorithm is that there is no efficient way to determine
which of the, exponentially many, states generated the data point. This makes the
E-step computationally intractable. A major issue in developing learning methods
for Generative models is to find accurate and tractable approximate solutions for the
E-step. In data-mining applications, where databases are often large and high dimen-
sional, this issue becomes particularly important. The deterministic property of the
GVQ model makes the use of a special class of message passing algorithms within
the learning scheme directly relevant. These algorithms [51] are used within graphi-
cal models to infer marginal probabilities given some evidence. In the deterministic
approach the distribution of a multi-dimensional state space is given simply by the
product of the marginals of this distribution. In other words, in the deterministic limit,
algorithms which estimate marginal probabilities well, will necessarily estimate the
full distribution equally well.

Message passing algorithms are interesting alternatives to the methods for prob-
abilistic models such as the mean-field approximation [22, 79, 60]. In this chapter
we will describe how message passing algorithms can be used for learning feature
representations in the form of a GVQ model. In addition, we will present an exten-
sive comparison between these algorithms and the mean-field method for learning
deterministic GVQ models. We will indicate under which circumstances a specific
algorithm should be preferred over others.

In section 2.2 and section 2.3 we present the basic idea of GVQ and its relation-
ship to standard vector quantisation, along with the GVQ learning algorithm. The
crucial issue of the tractable implementation of this algorithm is discussed at some
length in section 2.4.

In order to tune the representation for a particular application there are some
useful types of constraints one can impose on the model. In some applications one
knows that are certain distinct classes present in the data. For example in handwritten
digit analysis the features for constructing 2’s are not used for constructing 4’s which
have their own distinct set of features. Furthermore, to learn a multiple feature set
model it is desirable that the sets compete in a winner-take-all fashion, so that the
sets force each other to specialize on different structures in the data. We will show in
section 2.5 that in the GVQ model such a multiple set representation can be built in
in a natural way.

High dimensional real world problems, namely handwritten digits and image
compression, are studied in sections 2.6.1 and 2.6.2. The relation of this work to
other models is discussed in section 2.7, along with potential benefits to discrete op-
timisation using message passing schemes.
2.2 Standard Vector Quantisation

The aim of vector quantisation (VQ) is to represent a dataset by a smaller set of representative vectors. This set can be used to code a data pattern with a small number of bits. The code of a data pattern is given by the index of the closest representative vector. For this reason the collection of representative vectors is called a codebook and the vectors themselves are the codebook vectors. An example of a data set with its representative codebook vectors is shown in Figure 2.1 (a).

An important advantage of standard vector quantisation methods is that, for a given data set, they can quickly construct a set of representative codebook vectors. For this reason these techniques are widely used in many application domains for compression or for clustering of data. Standard vector quantisation does not, however, represent objects as a collection of features. To overcome this deficiency, we introduce generative vector quantisation, as described in the following section.

2.3 Generative Vector Quantisation

In Generative Vector Quantisation (GVQ) the objective is similar to that of VQ, namely to find a codebook representation of the data. In contrast to standard vector quantisation, GVQ reduces the number of representative vectors by using a smaller set of basic feature vectors \[ f_1, \ldots, f_n \] which exist in the same \( N \)-dimensional space as the data. Each codebook vector is then formed by some binary combination of these feature vectors,

\[
\sum_{i=1}^{n} s_i f_i \equiv F s,
\]

where the feature matrix \( F = [f_1 f_2 \ldots f_n] \), and the state vector \( s \in \{0,1\}^n \). There are therefore \( M = 2^n \) possible codebook vectors \( F s^1, \ldots, F s^M \).

An example of a set of codebook vectors generated by 3 features in a 2-dimensional space is shown in Figure 2.1 (b). To contrast this approach with the standard approach, the data used for GVQ is the same as in Figure 2.1 (a).

In Figure 2.1(b), the circles represent the generated codebook vectors which correspond to the 8 states (000), (100), \ldots, (111). The features are given by the codebook vectors corresponding to the unary state vectors (100), (010) and (001) etc. The zero state vector is the origin of the representation. The remaining codebook vectors are then ‘generated’ by combinations of these basic codebook vectors, or features. For example, the codebook vector corresponding to state (011) is given by adding the features corresponding to state (010) and (001), see Figure 2.1 (b).

Note that in GVQ the number of features \( n \) is not related to the dimensionality of the data space. Hence, there may be more or less basic feature vectors than there are dimensions in the data space.
2.3.1 The GVQ learning algorithm

In GVQ, as in standard VQ, each data point $x^\mu$ is associated with a particular codebook vector, indexed by $c_\mu$. Typically, this association is made such that $x^\mu$ is assigned to the closest codebook vector, in the Euclidian sense. The squared Euclidian distance between the whole data set $\{x^\mu | \mu = 1, \ldots, P\}$ and its codebook representation, $\{Fs^\mu | \mu = 1, \ldots, P\}$, is

$$E = \sum_{\mu=1}^{P} \|x^\mu - Fs^\mu\|^2.$$  

(1)

The task is, therefore, to find both the optimal associations of data points to codebook vectors, and the best feature vectors in order to minimize $E$. Since the associations between data points and codebook vectors will change if the feature matrix $F$ is changed, minimising (1) directly with respect to $F$ and the associations is not practical. For this reason we make use of a two-step iteration procedure.

After initialisation of the features $F$ the GVQ learning algorithm iterates between an association step 1 which finds, for each data-point, the most nearby codebook vector and a minimisation step 2 which finds the optimal feature configuration for the given association:

1. For $\mu = 1, \ldots, P$

$$c_\mu \leftarrow \arg\min_{j} \|x^\mu - Fs^j\|^2,$$  

(2)
2. \[ F \leftarrow \arg\min_F \sum_{\mu} \| x^{\mu} - Fs^{\mu} \|^2 \]  

(3)

The second step only involves the mean \( \langle x \rangle_k \) of each group of data associated with a single codebook vector \( k \) since,

\[
\arg\min_F \sum_k \sum_{\mu \in C_k} \| x^{\mu} - Fs^{\mu} \|^2 = \sum_k \sum_{\mu \in C_k} [\| Fs^{\mu} \|^2 - 2x^{\mu}Fs^{\mu}] = \\
\sum_k \sum_{\mu \in C_k} [\| Fs^{\mu} \|^2 - 2x^{\mu}Fs^{\mu}] + N_k \langle x \rangle_k^2 = \\
\sum_k N_k \| \langle x \rangle_k - Fs^{\mu} \|^2,
\]

(4)

where \( \langle s \rangle \) represents the set of all data points associated with the state \( s^i \) and where \( N_k \) is the number of data points in cluster \( k \). The expectation value \( \langle x \rangle_k \) is taken over the data in cluster \( \langle s \rangle \). The additive constant does not depend on \( F \). The objective function (4) can be minimized efficiently by use of Singular Value Decomposition (see for example Press, 1992, chapter 14.3).

The first step, (2), is computationally more difficult since, in principle, it involves a search through all \( 2^n \) binary states \( s \). In section 2.4 we will discuss and compare different approximate algorithms which can reduce this computational overhead.

2.4 Approximate association

In the association step (2) we want, for a given fixed data point \( x \), to minimize the error function

\[ E(s, x) = \| x - \sum f_j s_j \|^2 \]  

(5)

with respect to \( s \). Since there is an exponential number of binary states \( s \), an exhaustive search over all these states rapidly becomes computationally intractable for even a moderate number of features.

In this section we will compare two types of approach for finding the optimal state \( s^* \) which minimizes (5)\(^2\), drawing heavily on the terminology of graphical models.

---

1In this section we refer, for notational convenience, directly to a specific binary state \( s \) and omit the upper indices used in section 2.3.1. Furthermore, we do the association for a single data point \( x \). It is clear that the problem is the same if we instead use the cluster means \( \langle x \rangle \) of (4).

2Note, that the minimizing state \( s^* \) need not be unique. Here we do not specify a prior preference i.e., we regard each solution to be equally valid.
Chapter 2

[51, 48]. In doing so we make a distinction between two types of methods. The first class of methods only considers relations, implicitly given by (5), between the binary variables $S_i$. As will be explained the relationships between the binary variables can, in that case, be represented by an undirected graph. In section 2.4.1 we will present a number of specialized approximating algorithms for undirected graphical structures.

The second class of methods considers explicitly the relations between binary variables $S_i$ and visible variables $X_i$. The corresponding graphical dependency structure is then directed. Section 2.4.2 discusses an algorithm which exploits this graphical structure.

In section 2.4.3 we shall give an experimental comparison between these methods for a range of GVQ architectures.

2.4.1 Undirected graph methods

Since we are interested in finding the state $s$ which minimizes the error function $E(s, x)$ for a given $x$, we can define a new error function which contains only dependencies on $s$. First note that,

$$E(x, s) = x^2 - 2x \sum_i f_i s_i + \sum_{ij} f_{ij} s_i s_j$$

$$= x^2 - 2x \sum_i f_i s_i + \sum_i f_i^2 s_i^2 + 2 \sum_{i<j} f_{ij} s_i s_j,$$  \hspace{1cm} (6)

and that $s_i = s_i^2$. The new $s$-dependent error $E_x(s)$ is defined as

$$E_x(s) = \sum_i \left\{ h_i s_i + \sum_{j>i} w_{ij} s_i s_j \right\},$$  \hspace{1cm} (7)

where $w_{ij} = 2f_{ij} \cdot f_i$, $h_i = -2f_i \cdot x + f_i^2$. The $s$ that minimizes $E_x(s)$ is equal to the $s$ that minimizes $E(s, x)$. Note that this error function contains only pairwise and symmetric dependencies between the variables $s_i$. This dependency structure, given by the weight matrix $w_{ij}$, can be represented as an undirected graph. An example of a fully connected graph, i.e. all weights $w_{ij}$ are non-zero, is shown in Figure 2.2 a).

The following subsections discuss three different algorithms which make use of this undirected graph structure.

Belief Propagation (BP)

The error function (7) can be used to define a probability distribution $p_x(s)$ on the set of binary states $s$,

$$p_x(s) = \frac{1}{Z_x} \exp \left( -\frac{1}{2\sigma^2} E_x(s) \right),$$  \hspace{1cm} (8)

\(^3\)Note, that we use capitals to refer to variables and lowercase letters to refer to their values.
where $Z_\lambda$ is a normalization constant. In this formulation the state $s' = \arg\min_s E_\lambda(s)$ with the smallest error in (7) now corresponds to the state with the largest probability in the distribution $p_\lambda(\cdot)$. In the case that the noise $\sigma$ in (8) is decreased, $s'$ will start to dominate the distribution. In the limit $\sigma \to 0$ the corresponding probability $p_\lambda(s')$ saturates to the value 1. It is easy to see that the marginal probabilities $p_\lambda(s_j)$, i.e. probabilities of individual units, also saturate to the values 0 or 1. A useful property of the zero noise limit is that the single unit states $s_i$ for which $p_\lambda(s_i) = 1$ together form the global objective state $s = (s_1, s_2, \ldots, s_n)$ of the whole graph. Hence, by computing the marginals $p_\lambda(s_j)$ from (8) we can, by reducing $\sigma$ in (8), find the minimising state $s'$. However, computation of the marginals $p_\lambda(s_j)$ has the same computational complexity as the original minimisation problem (7) since it involves a summation over all combinations of the states of all the other units $j \neq i$.

The belief propagation algorithm [51] provides a computationally inexpensive approximate method to compute the marginals $p_\lambda(s_j)$. The basic idea of this method is to decompose the summation into a sequence of local operations which take place at the individual nodes. In appendix 2.A we will present this method and give an example for a simple network. The computational complexity of this algorithm scales quadratically $O(n^2)$ with the number of binary nodes $n$.

Our implementation of belief propagation starts with a large noise value $\sigma$. While running Belief Propagation, in an attempt to avoid local minima, the noise $\sigma$ is slowly reduced to zero. The corresponding state $s'$ for which all $p_\lambda(s'_i) = 1$ is then taken as the solution for our minimisation problem.

**Belief Revision (BR)**

Belief Revision [51] is an algorithm for directly tackling the minimisation problem $\min_s E_\lambda(s)$. Again, the trick is to carefully exploit the graphical structure of the problem, given by the weights $w_{ij}$, in order to decompose the minimisation problem into local operations. In fact, it can be shown that the belief propagation algorithm is equivalent to Belief revision by taking the limit $\sigma \to 0$. We refer to appendix 2.B for a derivation of the Belief Revision algorithm and a description of our implementation of the algorithm for the GVQ model. The computational complexity of this algorithm is $O(n^2)$.

**Mean-Field (MF)**

The basic idea of variational algorithms (of which the mean-field method is a special case) is to replace the intractable objective function with a tractable approximation to it, so that the optimization of the approximate objective function can be carried out efficiently, see for example [60]. Based on this principle we derive a mean-field variational algorithm in appendix 2.C to find the minimising $s'$ state of the objective function (7). The complexity of the mean-field algorithm is $O(n^2)$. 

---

*Generative Vector Quantisation*
Chapter 2

2.4.2 Approximation in the Directed Graph

Instead of representing only the relation in each pair $ij$ of binary variables $S_i$ and $S_j$ as in the previous section, we can also form a graphical representation of the relation in each pair of all the variables i.e., binary variables $S_i$ as well as continuous variables $X_j$. The most efficient way to do this is to represent the relations with a directed graph.

To explain this we represent our GVQ model as a joint probability model $p(x,s)$ of binary states $s$ and visible states $x$. The prior distribution of the binary units $p(s)$ is constant, i.e. $p(s) = \frac{1}{2}$ and $p(s) = \frac{1}{2\pi}$. The joint probability distribution can be constructed as follows:

$$p(x,s) = p(s)p(x|s) = 2^{-n} (2\pi\sigma^2)^{-\frac{n}{2}} \exp \left\{ -\frac{1}{2\sigma^2} E(s,x) \right\}, \quad (9)$$

where $E(s,x)$ is our original objective function (5). An example, of the graphical representation of (9) is shown in Figure 2.2 b). As can be seen, there are no direct links between binary hidden units reflecting the fact that the prior distribution $p(s)$ is factorized, i.e. $p(s) = \prod p(s_i)$. The arrows reflect the relation between hidden states $s$ and visible states $x$ given by $p(x|s) \propto \exp \left\{ -\frac{1}{2\sigma^2} \sum f_i s_i \right\}$ so that the set of arrows expanding from unit $S_i$ correspond to feature $f_i$.

Belief Propagation in the Directed Graph (DBP)

In appendix 2.D we present a belief propagation algorithm which explicitly takes the directed graphical structure into account by passing messages from hidden units to visible units and vice versa. Using the same noise reduction process $\sigma \rightarrow 0$ as in section 2.4.1, the desired state $s'$ can be inferred by computing the conditional probabilities $p(s|s|x)$ from (9). Again, in the limit $\sigma \rightarrow 0$ the marginal probabilities $p(s'|x) \rightarrow 1$ together form our desired objective state $s' = (s'_1, s'_2, \ldots, s'_n)$. For general probability distributions with bi-partite structures as in Figure 2.2 b) the computational complexity of this algorithm scales exponentially with the number of connections between a visible unit $X_i$ and hidden units $S$, that is, the algorithm is exponentially complex in
the number of parents of the visible units. However, for the special case of our GVQ model, which has a quadratic dependence between binary states \( s \) and visible states \( x \), we can reduce this complexity to \( \mathcal{O}(n^2 \times d) \), by introducing an integral transform. However, this is potentially at the cost of decreased accuracy\(^4\).

**Belief Revision in the Directed Graph (DBR)**

In section 2.4.1 we transformed the probabilistic belief propagation algorithm into a noiseless algorithm by taking the limit \( \sigma \to 0 \). The same operation can be applied on the DBP algorithm for the directed graph of section 2.4.2. The resulting belief revision algorithm still takes the directed graphical structure into account. However, in contrast to the probabilistic algorithm, the complexity of the algorithm can no longer be reduced to polynomial. It remains exponential in the number \( k \) of connections that each visible unit has with the binary units \( i.e. \), the overall complexity is \( \mathcal{O}(n \times d \times 2^k) \). We refer to appendix 2.E for a more technical discussion of this approach.

**2.4.3 Experimental comparison**

So far we have not characterized the type of problems for which we can find a sensible GVQ representation. In practice we can expect a large range of situations where we want to find a feature representation. For example, some situations require a large number of features in a low dimensional data space (over-complete basis) or, in the opposite case, they require a few nearly orthogonal features in a high dimensional space. The purpose of this section is to determine under which circumstances the approximating algorithms are most suitable.

**Influence of connectivity structure**

In this sub-section we monitor the performance of the algorithms if we gradually increase the complexity of the graphical structure of the GVQ model. To do this, we generated a number of artificial problems. In each experiment we sampled a fixed number \( n = 12 \) hidden units of \( d = 4 \) dimensional features (visible units). The features \( f_i \) are chosen such that each visible node \( X_i \) has at most \( k \) connections with the binary layer, see Figure 2.3. The connections are selected randomly. The feature values \( f_{ij} \) that determine the strength of these connections are sampled according to \( f_{ij} \sim \mathcal{N}(2,1) \), a Gaussian distribution with mean 2 and variance 1. We chose a non-zero mean to avoid non-realistic symmetries in the generated data. Together with each feature set we randomly chose a binary state \( s' \), according to \( p(s') = 0.5 \). Then for fixed \( f_i \) and \( s' \) we generated an input pattern \( x \) using \( x = \sum_i f_i s'_i + \epsilon \), where \( \epsilon \) is adding a small amount of random noise. The components of \( \epsilon \) are sampled from \( \mathcal{N}(0, \frac{1}{10}) \). Given the input pattern \( x \), each method was used to recover the generating

\(^4\)As explained in appendix 2.D.
Figure 2.3: GVQ networks with $n = 12$ features and $d = 4$ visible units. a) Each visible unit $X_i$ is connected with only two binary parent units ($k = 2$). b) Fully connected network ($k = n$).

Figure 2.4: Comparison of the Belief Propagation (BP, dotted lines), the Belief Revision (BR, dashed lines), Mean-Field (MF, solid lines) and the Belief Revision in the Directed graph (DBR, dot-dash lines) algorithm. a) The error $E(s, x)$ as a function of the number of parent connections $k$ (given as a fraction $k/n$ of the total number of parents $n$). b) The error $E(s, x)$ as a function of the number of input dimensions $d$ (given as a fraction $d/n$ of the total number of parents $n$).

We then computed the error $E$ which is defined here as the average absolute error per input\textsuperscript{5} dimension i.e., $E = \frac{1}{d} \sum_i |x_i - \sum_j f_{ij} s_j|$ which is directly related to (5). Note that the minimum error is $\varepsilon$. This procedure was repeated 100 times for each connectivity number $k$. Figure 2.4 (a) shows the average of these results as a function of the number of connections $k$. For each method the error clearly increases with

\textsuperscript{5}We look at the input space and not at errors in the binary latent space since we are interested in the reconstruction errors of data examples. Two codebook vectors with the same distance to data point $x$ but with a large distance to each other in the binary latent space are considered equally valid.
increasing number of connections $k$. If the number of connections is small ($k/n < 0.6$) the error of the directed belief revision algorithm is close to the minimum error $\varepsilon$ \textit{i.e.}, it is close to the exact solution. In this region DBR outperforms all undirected algorithms. At a certain point ($k/n \approx 0.65$) the error of DBR starts to rise quickly. This is expected since the loops are shorter in denser networks. The transition point is somewhat dependent on the imposed noise $\varepsilon$. For smaller values of $\varepsilon$ the point shifts to the right. In the extreme case $\varepsilon = 0$ DBR does not make errors anymore, \textit{i.e.}, $E(s, x) = 0$ for $\varepsilon = 0$ (which is not true for the other methods). However, for the value of $\varepsilon$ we used here the DBR algorithm performs poorly for fully connected networks. In that case we should use the mean-field algorithm.

In Figure 2.4 we do not include the performance of DBP. To obtain comparable performance to DBR, we found that we needed to anneal $\sigma$ to such a small level that retaining accuracy of the integral transform became computationally burdensome.

As can be seen from Figure 2.4, the error of BP is in all cases larger than the BR error. To get a BP error closer to BR we need to anneal to even smaller values of $\sigma$. The anticipated positive effect of avoiding local minima was not present.

In Figure 2.4 (b) we show the result for fully connected networks ($n = 15$, $f_{ij} \sim \mathcal{N}(0, 1)$) where we increase the number of input dimensions $d$. For the fully connected case we omitted the DBR and DBP methods since we know, from the previous experiment, that their performances will be poor. We see that for these fully connected GVQ networks mean-field outperforms the other methods over the whole region. For large values of $d/n$ the thresholds $h_i$ will dominate the contributions given by the interactions $w_{ij}$ in the error function (7). Hence, effectively the units $S_i$ will become more independent in which case all methods perform better explaining the decreasing errors in Figure 2.4 (a).

In the experiments with the BR method we made an interesting observation. In all the trials where the messages converged to a stationary value the final error was equal to the minimum error $\varepsilon$. The BR error in Figure 2.4 comes from the remaining non-converging trials. This indicates that by looking at the convergence behaviour of BR we are able to determine whether the final answer is correct. If it does not converge we can always do the association with MF instead.

Performance as a function of the similarity between the features

In the experiments above the features were sampled from zero mean normal distributions. This may not be particularly representative of features in real-world problems. A crude attempt to address this issue is given by generating features which have a degree of similarity. Here we do this by drawing the features from a Gaussian with non-zero mean. The feature values $f_{ij}$ are sampled from a normal distribution with mean $\gamma$, that is $f_{ij} \sim \mathcal{N}(\gamma, 1)$. Hence, the larger $\gamma$ the more the features $f_i$ ‘point in the same direction’. The result is shown in Figure 2.5. We see that if $\gamma > 4.5$ the message passing methods BR and DBP both outperform MF. In the previous experiments, the mean-field method performed better for fully connected architectures.
However, the results in Figure 2.5 indicate that under certain circumstances message passing algorithms may outperform mean-field also for fully connected networks.

2.5 Imposing constraints on the basic model

In some applications one has a priori knowledge about the probability distribution which generated the data. In that case it can be of great help to impose constraints on the basic GVQ model which incorporate this knowledge. In this section we propose three constraining methods which we believe to be useful for a large class of applications. The first two of these methods impose constraints on the distribution of the hidden states $s$. The third represents a constraint on the distribution of values of the input patterns $x$.

2.5.1 Multiple feature sets and multi-valued features

In many real world problems we can expect that there are different unrelated groups of patterns. In each of these groups the patterns are built up out of features from a set which is specific for the group. It is easy to adjust the GVQ learning algorithm for learning multiple feature sets by simply constraining the set of allowable states.

Figure 2.6 shows the result of learning two sets of 3 features from a data set of 300 samples. Features corresponding to one set can only be combined with features from the same set. The origins of the different sets, indicated with the dashed lines in Figure 2.6, which are considered as constant ‘on’ features for that set, are also determined by the optimization process. For a given data point there is only one
feature set responsible for generating the closest code vector. In this sense we can interpret the multiple set model as a winner-take-all configuration of multiple GVQ’s.

Allowing multiple sets makes it possible to find different groups of objects, i.e. each GVQ within the winner-take-all configuration learns to represent a certain class of objects in an un-supervised manner. An example of this is given in section 2.6.1 in which handwritten 3’s and 5’s are separated in an un-supervised manner using multiple feature sets. Note that in the extreme case of using one feature per set, GVQ is equivalent to standard vector quantisation.

2.5.2 Penalty constraints

Another way of biasing the solution to those consistent with prior beliefs is given by adding an extra penalty term to the energy function (1). For example, one can bias the final representation to be sparse, i.e. each object is composed of a small number of features from a large set, by adding for example the term $\lambda \mathbf{s}^T \mathbf{s}$ to the energy function.

2.5.3 Binary features for binary data

In the case that the elements of the data are binary, it is desirable that the feature combinations result in values close to 1 and 0. Experiments with the original error function (1) on this type of data, however, result in feature combinations with smoothly varying values, i.e. the reconstructed data patterns are not binary patterns but have values between the binary states.

Better results are obtained by using sigmoid squashing functions so that the codebook vectors are forced to have values close to 0 and 1. For this reason we use a sig-
moid function to ‘squash’ the combination of features. For ease of interpretation, it is also advantageous that the features themselves are constrained to be binary. This can be implemented in a ‘soft’ manner by defining the constrained features as \( F \equiv \sigma(\hat{F}) \) such that we can minimize the error function with respect to the unconstrained matrix \( \hat{F} \). Our error function for binary data is thus

\[
E = \sum_{\mu=1}^{p} \| x^{\mu} - \sigma_\beta \left( \sigma(\hat{F}) s^{\mu} \right) \|^2
\]

in which

\[
\sigma(x) = (1 + e^{-x})^{-1},
\]

and

\[
\sigma_\beta(y) = \sigma(\beta (y - 1/2)),
\]

where \( \beta \) is a parameter which controls the steepness of the squashing function. In our implementation (10) is minimized with respect to \( \hat{F} \) using the scaled conjugate gradient method [53].

After incorporating the squashing functions, the model becomes more closely related to other models such as sigmoid belief networks [47] and the “multiple cause mixture” representation proposed by [61]. Note, however, that the deterministic approach, see section 2.7.1, and the learning procedure, discussed in section 2.4, of GVQ sets it apart from these methods.

### 2.6 Results on Real-World Data

In this section we demonstrate the application of GVQ in two practical situations. First we extract features of handwritten digits. Using a small number of basic features GVQ can find nice reconstructions of the original digits. Finding a feature representation can for example be useful as a pre-processing step in a classifier. In the second application we demonstrate the advantage of GVQ over standard vector quantization in image compression. We show that when using a feature representation images can be compressed into an even smaller number of data bits.

#### 2.6.1 Handwritten digits

We randomly selected 400 training images of handwritten ‘threes’ and ‘fives’ from the CEDAR CDROM 1 database [27]. Since the original images contain different numbers of pixels, we rescaled all images to \( 20 \times 20 \) pixels. A typical sample of these images is shown in Figure 2.7.

We decided to fit a GVQ model consisting of 4 mutually exclusive sets of 5 features (including in each set an origin feature), see section 2.5.1. For this application
we made use of the binary feature binary data version of GVQ as discussed in section 2.5.3 with $\beta = 4.5$. The features, which were obtained, are shown in Figure 2.8. Each row in the figure corresponds to a feature set and the last feature on the right hand side in each row corresponds to the origin feature of the set. By inspection it is clear that the first two sets (top two rows) specialize on ‘threes’ whereas the last set (last row) specializes on constructing ‘fives’. The third set can construct both ‘fives’ and ‘threes’. These properties become more clear if we look at Figure 2.9 where 37 of the most representative feature combinations (codebook vectors) are shown. By most representative we mean those feature combinations (codebook vectors) which account for most of the data. The codebook vectors in the left sub-figure of Figure 2.9 are combinations of features from the first set in Figure 2.8, which clearly are all ‘threes’. As we see from the second sub-figure in Figure 2.9, the second feature set, although primarily concerned with modeling ‘threes’, is nevertheless able to
construct a ‘five’. This is even more apparent in the third sub-figure in Figure 2.9 containing both ‘threes’ and ‘fives’. The reconstructions in this set show that there is a class of handwritten digits containing ‘threes’ and ‘fives’ which share at least one feature, namely the origin feature. This origin feature can be supplemented with an additional feature to become either a ‘three’ or a ‘five’.

2.6.2 Image Compression

A well known application of vector quantisation is in image compression. In this section we demonstrate the additional advantage for image compression gained by describing codebook vectors in terms of a small number of features.

As an example, we used GVQ to compress the image in Figure 2.10 a) and compared the result with standard vector quantisation. The original image consists of 768 × 704 pixels with 256 possible gray levels for each pixel which corresponds to 865 kbits of information. The image was split into $P = 2112$ segments of 16 × 16 pixels. We used standard VQ to construct 16 codebook vectors to represent this set of $P$ segments. We then reconstructed the image using the closest codebook vector to each image segment. The result is shown in Figure 2.10 b). We also applied standard GVQ using $n = 8$ features (plus an additional origin feature) to construct a representative set of codebook vectors for the image segments. Figure 2.10 c) shows the reconstructed image. The superior performance of GVQ over VQ in representational accuracy is given by the codebook flexibility. In standard VQ only 16 codebook vectors can be used, compared to $2^n = 256$ codebook vectors in GVQ. Despite there being more codebook vectors available in GVQ, the information required to define the compressed image using GVQ is less than that in VQ, as we show in the following subsection.

The features, which were learned to construct codebook vectors representing the segments of the Vermeer image, are shown in Figure 2.11. Interestingly, the final features can be seen to be slightly biased to modeling variation around the vertical...
Generative Vector Quantisation

Figure 2.10: a) The Vermeer image prior to compression consists of 865 kbits. After compression: b) With standard vector quantisation $I_v = 74$ kbits, c) with GVQ using 1 set of 8 features (see Figure 2.11) $I_{GVQ} = 56$ kbits.

Figure 2.11: Features which were learned for representing the 2112-16 × 16 bit segments constituting the Vermeer image. The lower-right feature is the origin feature, which is always on.

direction, which is plausible given the large number of almost vertical shadows in the original image.

Information Requirements

Since a codebook vector in GVQ is constructed out of a set of $n$ features, we need at most $n$ bits to specify a feature combination vector. If the data have a clustered structure, the number of bits needed will be smaller than $n$ since some combinations will never be used. Therefore, if the number of used feature combinations is $N_{gvq}$ we also need $\log N_{gvq} \leq n$ bits to specify a codebook vector in GVQ. Similarly, in standard vector quantisation we need $\log N_{vc}$ bits to specify a codebook vector if $N_{vc}$ is the number of learned codebook vectors.

Consider the case that the image to be compressed is unique, in the sense that
we can not use features (or codebook vectors) which were used to encode previously encountered images. In order to compare the compression efficiencies, we need to take into account the information to describe the codebook vectors in VQ and the information in the features in GVQ. This information is proportional to the number of pixels $n_p$ used in an image segment, and the information required to determine the gray value of a pixel $I_g$. Hence, if the original image is split into $P$ segments, each made up of $n_p$ pixels, the information $I_{VQ}$ in the compressed image using VQ is

$$I_{VQ} = I_g n_p N_v + P \log N_v$$

and the information $I_{GVQ}$ in the compressed image using GVQ is

$$I_{GVQ} = I_g n_p n + P \log N_{gvq}$$

If there is a moderately number of segments $P$, which is the case if we compress a single specific image, then for a given compression quality\textsuperscript{7} the difference between $I_{VQ}$ and $I_{GVQ}$ is determined mainly by the difference between $N_v$ and $n$. Since we can construct a large number of codebook vectors with a small number of features it is expected that $n \ll N_v$, especially if the distribution of the image segments has a structured multi-modal form.

The image in Figure 2.10 b), obtained after compression with standard VQ, consists of $I_{VQ} = 74$ kbits. In contrast, if we apply our GVQ algorithm using $n = 8$ features the compressed image, Figure 2.10 c), consists of $I_{GVQ} = 56$ kbits. While containing 18 kbits less of information, the GVQ compressed image gives without doubt a superior representation of the original image.

### 2.7 Relation of GVQ to other models

A large number of data modeling techniques can be seen as special cases of using Gaussian mixture models. Although not necessary for the motivation for GVQ, in this section we describe how GVQ can be seen as part of an ongoing tradition by relating it to the framework of Gaussian mixture models. This will enable us to clarify the relation of our model to other recently proposed techniques and approximations to them.

In the present context, a Gaussian mixture model can be conveniently considered as a layer of hidden or latent variables $s = (s_1 \ldots s_n)$ connected to a layer of visible variables $x = (x_1 \ldots x_N)$. Each data point then corresponds to an instantiation of the visible units. The distribution on the visible units is obtained from the marginal of the joint distribution over the hidden and visible units,

$$p(x) = \sum_s p(x|s)p(s), \quad (11)$$

\textsuperscript{7}By compression quality we mean the similarity between the compressed image and the original image.
where the likelihood term is given by
\[
p(x|s) = (2\pi\sigma^2)^{-d/2} e^{-\frac{1}{2\sigma^2}||x-g(s)||^2}.
\] (12)

For convenience, we write the prior distribution of the hidden states \(p(s)\) in terms of an energy function \(\phi(s)\),
\[
p(s) = \frac{1}{Z} e^{-\frac{1}{\sigma^2} \phi(s)},
\] (13)

where \(Z\) is the normalizing constant for the prior. Within this model, each hidden state \(s\) corresponds to a uniquely located Gaussian distribution in the visible layer with mean \(g(s)\) with a prior belief that each Gaussian is responsible for the visible variable given by the prior \(p(s)\).

In this section we discuss how various aspects of the GVQ model can be related to different choices of the mean function \(g(s)\) and prior \(p(s)\) in the limit that \(\sigma\) goes to zero.

### 2.7.1 The deterministic vs the noisy approach

Taking the zero \(\sigma\) limit in the Gaussian mixture model provides a hard assignment of data points \(x\) to closest Gaussian centers. The resulting \(\sigma \to 0\) model corresponds to standard vector quantisation in which the codebook vectors are the cluster centers \(g(s)\). In GVQ the additional constraint is that the codebook vectors are generated by binary combinations of vectors. This corresponds to the choice \(g(s) = \sum_i f_i s_i\) for the mean function, where the hidden states are now coded as binary vectors \(s \in \{0,1\}^n\).

As discussed in section 2.4.2 the finite noise model corresponding to GVQ is given by
\[
p(s,x) = p(s) \left(2\pi\sigma^2\right)^{-d/2} \exp\left\{-\frac{1}{2\sigma^2}||x - \sum_i f_i s_i||^2\right\},
\] (14)

where, in the basic approach of section 2.3.1, the prior probability of a feature combination is the same for each combination \(i.e., p(s)\) in (14) is constant. This finite noise model is closely related to the Cooperative Vector Quantizer proposed in [79] which was further investigated in the context of mean-field learning in [22]. GVQ corresponds to (14) with \(\sigma \to 0\). This implies that the posterior probabilities \(p(s|x)\) reduce to \(p(s|x) = \delta(s' - s)\) in which \(s'\) is the unique binary state that is associated with \(x\). Since \(\delta(s' - s) = \prod_i \delta(s'_i - s_i)\), in the zero noise limit the multi-dimensional distribution of the hidden binary states \(s\) is given by the product of its marginals. For this reason the message passing algorithms, described in section 2.4, which infer marginal probabilities \(p(s_i)\) become directly relevant in the zero noise limit. As we saw in section 2.4.3 there are indeed situations where these message passing algorithms outperform the variational algorithms for finite noise models.
Chapter 2

In summary, GVQ forces a solution in which each data point is explained by a single process - that is, there is a unique explanation for each data point, found by a competitive process subject to this requirement. On the other hand, in a finite $\sigma$ model, each data point is associated with each Gaussian center with a certain probability. If one attempts to interpret each data point in terms of features one would then need to evaluate the contribution of each Gaussian to the explanation for the data point in some manner. In applications where clusters have strong overlap, these contributions are important. Otherwise, a finite $\sigma$ model would make interpretation unnecessary complicated. Another motivation for considering the zero noise limit is that taking the most probable explanation (nearest Gaussian center) in the finite $\sigma$ models may not give rise to a satisfactory interpretation since the competition between Gaussian centers for the single best explanation for each data point has not been optimized during learning.

GVQ is most appropriate in cases in which one believes that any data point is well explained by a single process (codebook vector).

2.7.2 Alternative choices for the prior distribution

Within the probabilistic framework, some interesting connections can be made to the sparse coding work of [50] and the independent factor analysis (IFA) work of [2]. The main difference between these models and our model is in the assumption for the hidden state distribution $p(s)$. Both authors consider hidden state variables with continuous values, that is $s \in \mathbb{R}^n$. In IFA Attias [2] considers a product of Gaussian mixture distributions (independent factors) for the hidden states $s$. If, within IFA, the number of mixtures for each hidden state variable $s_i$ is set to 2 (bi-modal distribution), the distribution of the visible patterns can be regarded as a noisy version of the binary feature combinations in GVQ.

In contrast to the multi-modal assumption of IFA and the binary assumption of GVQ, Olshausen and Field [50] consider a sharply peaked unimodal distribution for continuous hidden variables $s$. This choice encourages a sparse representation of the data patterns since the hidden variables $s_i$ will be in the ‘off’ state most of the time. In this case an individual pattern will be constructed as a combination of only a small number of features out of a large, typically over-complete, set of features. As discussed in section 2.5.1, a similar property can be incorporated in our method by replacing the basic constant GVQ prior in (13) with a soft prior $\phi(s) = s^T s$. Note, however, that when this penalty term becomes too large, only a single feature will be used to represent a pattern. In other words GVQ will tend to standard vector quantisation as the solution is strongly encouraged to be sparse.

2.7.3 Binary Lattice Vector Quantisation

In another context, research has been done on Binary Lattice Vector Quantizers or Direct Sum Quantizers. In fact, the binary codebook representation of GVQ model
Generative Vector Quantisation

is formally equivalent to the representation of a Binary Lattice Vector Quantizer. These representations have been studied for the purpose of data transmission across noisy channels. The main objective there is to transmit coded data such that the reconstruction error is minimal. An important sub-problem within that objective is the ‘Index Assignment Problem’ which is to find an optimal binary index assignment to codebook vectors as to minimize the mean-squared error caused by channel errors, see [41] and [33].

For ‘direct sum quantizers’, an alternative method for the association step is studied in [4]. This method is a heuristic compromise between component wise optimization and exhaustive search where the association is done in multiple stages bringing the codebook closer to the data-point at each stage. Whether this method is more accurate and efficient than the methods studied in section 2.4 remains to be investigated.

To our knowledge binary lattice vector quantizers have, however, not been studied for the purpose of feature extraction and clustering of non-homogeneous data, which has been the purpose of the present chapter.

2.8 Conclusion

Generative Vector Quantisation is a method, which performs salient feature extraction at modest computational expense. The simplicity of GVQ, which searches for descriptions in terms of binary feature combinations, may lead to a lucid data representation, which is important in many data exploration tasks. A central thesis of the GVQ model is that data points are explained by a single generating process. Unlike a probabilistic model, GVQ constructs a competition between alternative explanations for a data point, in which there can be only one winning explanation. This winner-take-all process provides the basis for a clear feature representation. The deterministic nature of GVQ allows the use of a larger class of (approximate) association methods, such as Belief Revision, within the learning scheme. However, in the case that the data cannot be expected to be explained by a winner-take-all process, a probabilistic approach may be a more appropriate.

GVQ is potentially a powerful tool for exploring and representing data in a deterministic manner. Ultimately, the strength of GVQ lies in it’s transparent simplicity, being based on the intuitive notion that, although data may appear complex, it’s construction may be well understood in terms of a small number of elementary building blocks.

Appendix 2.A  Belief Propagation

An efficient technique to find an approximate solution to the marginal $f_k(s_j)$ of (8) is to decompose the global summation operation into distributed local operations, reducing the exponential summation to quadratic time.
For expositional clarity, consider an energy function $E(x)$, with weights $w_{ij}$ defined to give a chain structure as shown in figure 2.12(a) (technically, the weight matrix has zeros everywhere except for entries along the first diagonals adjacent to the main diagonal). For the chain structure in Figure 2.12(b) the marginal probability that unit 1 is in state $s_1$ is

$$p(s_1) \propto \sum_{s_2,s_3,s_4} e^{(h_1 s_1 + s_1 w_{12} s_2 + h_2 s_2 + s_2 w_{23} s_3 + h_3 s_3 + s_3 w_{34} s_4 + h_4 s_4)}$$

(15)

which can be decomposed into local operations as follows

$$p(s_1) \propto e^{\nu [h_1 s_1]} \sum_{s_2} \left( e^{\nu [h_2 s_2 + s_2 w_{12} s_1]} \sum_{s_3} \left( e^{\nu [h_3 s_3 + s_3 w_{23} s_2]} \sum_{s_4} e^{\nu [h_4 s_4 + s_4 w_{34} s_3]} \right) \right)$$

(16)

where $\nu = -\frac{1}{\sigma}$. Distributing the marginalization in this manner results in a summation over a number of states that scales only linearly with the network size instead of over an exponentially scaling number of states in (15). To write this in a more general form we define the message that node $S_j$ sends to $S_i$ as

$$\lambda_{ij}(s_i) = \alpha \sum_{s_j} e^{\nu [h_j s_j + s_j w_{ij} s_i]} \prod_{k \in \mathcal{N}_j \setminus i} \lambda_{jk}(s_j),$$

(17)

where $\mathcal{N}_j$ is the set of all nodes connected to node $j$. Combining the incoming messages $\lambda_{ij}$ into node $S_i$ gives the marginal probability distribution of that node

$$p(s_i) \propto e^{\nu h_{s_i}} \prod_j \lambda_{ij}(s_i).$$

(18)

The recurrent marginalization procedure defined by (17) and (18) will give an exact solution for all connection weights $w_{ij}$ that define singly connected graphs i.e., graphs without loops.
For graphs with loops, for example Figure 2.12(b), which corresponds to the energy function
\[ E_x(s) = h_1s_1 + s_1w_{12}s_2 + h_2s_2 + s_2w_{23}s_3 + \\
+ h_3s_3 + s_3w_{34}s_4 + h_4s_4 + s_4w_{14}s_4 + s_2w_{24}s_4 \] (19)
the method is still applicable although no longer guaranteed to find the optimal solution. Nevertheless, there is experimental evidence [40, 73] that for large classes of graphs with loops the belief propagation algorithm gives good solutions.

In our implementation the messages are initialized as
\[ \lambda_{ij}(s_i) = \alpha \sum_{s_j} e^{\nu[h_i + s_is_j]} \]
with \( \nu = -1/(2(0.1)^2) = -50 \). After initialisation an iteration in the procedure is as follows:

- A random ordering of the nodes is chosen, which are then sequentially visited in that order.
- For each node, all messages coming into the node are updated according to the rule (17) and the state of node \( i \) is updated to
\[ s_i^t = \arg\max_{s_i} \left( e^{\nu h_i} \prod \lambda_{ij}(s_i) \right). \]
- The variance \( \sigma^2 \) is halved in \( \nu = 1/(2\sigma^2) \).

This iterative process is repeated until \( \sigma = 10^{-4} \). We then choose that state \( s \) which in the iterations had the lowest energy \( E_x(s) \).

### Appendix 2.B Belief Revision

The inference problem that we need to solve is to find a single hypothesis or explanation \( s \) for each observed state \( x \).

The minimisation problem for the chain in Figure 2.12(a) is
\[ E^* = \min_{s_1, s_2, s_3, s_4} \left( h_1s_1 + s_1w_{12}s_2 + h_2s_2 + s_2w_{23}s_3 + h_3s_3 + s_3w_{34}s_4 + h_4s_4 \right) \] (20)
which can in analogy with (16) be decomposed into local operations as follows
\[ E^* = \min_{s_1} \left( h_1s_1 + \min_{s_2} \left( s_1w_{12}s_2 + h_2s_2 + \min_{s_3} \left( s_2w_{23}s_3 + h_3s_3 + \min_{s_4} \left( s_3w_{34}s_4 + h_4s_4 \right) \right) \right) \right). \] (21)

Note that (21) has the same de-componential structure as (16) except that the summation operator is changed into a minimisation operator and the messages are combined as a summation instead of as a product.
In a more general form, we define the message $\lambda_{ij}(s_i)$ that node $S_j$ sends to $S_i$ as

$$\lambda_{ij}(s_i) \equiv \min_{s_j} \left( s_j w_{ij} s_j + h_j s_j + \sum_{k \in \mathcal{E}_j \setminus i} \lambda_{jk}(s_j) \right),$$

(22)

where $\mathcal{E}_j$ is the set of all nodes connected to node $j$. With this definition we see from (21) that the minimisation problem for the network in Figure 2.12 can be rewritten in the following recurrent form

$$E_i^t = \min_{s_i} \left( \sum_{j \in \mathcal{E}_j \setminus i} \lambda_{ij}(s_i) + h_i s_i \right).$$

(23)

The recurrent minimisation procedure defined by (22) and (23) will give an exact solution for all connection weights $w_{ij}$ that define singly connected graphs \textit{i.e.}, graphs without loops. Nevertheless, there is experimental evidence [40, 73] that for large classes of graphs with loops the belief propagation algorithm gives good solutions.

In our implementation, the messages are initialized as $\lambda_{ij}(s_i) = \min_{s_j} \left( s_j w_{ij} s_j + h_j s_j \right)$. After initialisation an iteration in the procedure is as follows:

- A random ordering of the nodes is chosen, which are then sequentially visited in that order.

- For each node, all messages coming into the node are updated according to the rule (22) and the state of node $i$ is updated to $s_i^t = \arg\min_{s_i} \left\{ h_i s_i + \sum_{j \in \mathcal{E}_j \setminus i} \lambda_{ij}(s_i) \right\}$.

This iterative process is repeated until the messages converge. If they do not converge, the iterations are stopped after a predefined maximum number of iterations. We then choose that state $s$ which in the iterations had the lowest energy.

The computational complexity of this algorithm is quadratic in the number of nodes since, for each of the $n$ nodes, there are $n$ messages, in the fully connected case.

\textbf{Appendix 2.C \quad Mean Field}

The basic idea of variational algorithms (of which the mean-field method is a special case) is to replace the intractable objective function with a tractable approximation to it, so that the optimization of the approximate objective function can be carried out efficiently.
Generative Vector Quantisation

To explain the mean-field approximation for the association step in GVQ learning we first formulate the model as a probability distribution with finite noise $\sigma$.

$$p_s(s) \propto \exp\left\{-\frac{1}{2\sigma^2} E_s(s)\right\}.$$  \hspace{1cm} (24)

Finding the most probable state $s$ of $p_s(s)$ is equivalent to minimising $E_s(s)$, and in the limit $\sigma \to 0$, the distribution $p_s(s)$ becomes deterministic. That is, the mean state $s$ is equal to the most probable state. We can therefore use an algorithm that attempts to approximate the mean of $p_s(s)$ for finite $\sigma$ and, in the limit that $\sigma \to 0$, this will become an approximation for the most probable state.

One way to find an approximation to the mean of the variables of an intractable distribution is to use a simpler, tractable approximating distribution. Specifically, in the variational method the objective is to find an approximating $Q_s(s)$ distribution to the state distribution $P_s(s)$ with which the associations can be tractably computed. The optimal approximating $Q_s(s)$ is found by minimising the Kullback-Leibler divergence between the two distributions

$$KL = \sum_s Q_s(s) \log \frac{Q_s(s)}{P_s(s)} \geq 0$$  \hspace{1cm} (25)

with respect to the parameters of $Q_s(s)$. Note that the Kullback-Leibler divergence is a positive measure of the difference between two distributions.

In its most basic form, the variational approximating distribution $Q_s(s)$ is factorial. This is known as the mean-field assumption for $Q_s(s)$

$$Q(s|x) = \prod_i q_i(s_i) = \prod_i \mu_i^{s_i} (1 - \mu_i)^{1-s_i},$$  \hspace{1cm} (26)

where $\mu_i \in [0, 1]$ are called the mean-field parameters. Substitution of $Q_s(s)$ into $KL$ gives (up to a constant):

$$KL' = -\sum_i \mu_i \log \mu_i + (1 - \mu_i) \log (1 - \mu_i) - \frac{1}{2\sigma^2} \left( \sum_i \sum_{j \neq i} \mu_i \mu_j w_{ij} + \sum_i \mu_i h_i \right).$$  \hspace{1cm} (27)

To find a solution for the $\mu_i$, we set the derivatives w.r.t. to mean-field parameters $\mu_i$ equal to zero, which leads to the following mean-field fixed point equations,

$$\mu_i = \text{sig} \left( \frac{1}{2\sigma^2} \left\{ \sum_{j \neq i} \mu_i w_{ij} + h_i \right\} \right),$$  \hspace{1cm} (28)

where $\text{sig}(x) = (1 - \exp(-x))^{-1}$. In the limit $\sigma \to 0$ these equations become

$$\mu_i = \Theta \left( \sum_{j \neq i} \mu_i w_{ij} + h_i \right),$$  \hspace{1cm} (29)
Chapter 2

where \( \Theta(x) = 0 \) for \( x \leq 0 \) and \( \Theta(x) = 1 \) for \( x > 0 \). Hence, the solutions for \( \mu \) become binary and there no longer exists a distinction between state values \( s \) and state probabilities \( \mu \). For a given input \( x \) (29) defines an iterative procedure to find an associated state \( s \).

In our implementation, the mean-field parameters are initialized as \( \mu = \frac{1}{\xi} + \xi \), where \( \xi \) is small random noise and \( \sigma \) is initialized as \( \sigma = 100 \). After initialisation an iteration in the procedure is as follows:

- A random ordering of the nodes is chosen, which are then sequentially visited in that order.
- Each node, is then updated according to equation (28).
- The noise \( \sigma \) is then reduced according to \( \sigma \leftarrow \sigma / \alpha \).

The noise reduction parameter \( \alpha \) is chosen such that \( \sigma = 10^{-4} \) at the final iteration.

Gibbs Sampling

Another well known optimization technique for stochastic models is Gibbs sampling. In Gibbs sampling the state of a unit is updated according to the probability \( p(s_i | s_{\neg i}) \), where \( s_{\neg i} \) contains the values of all units besides unit \( i \). This conditional probability for the model (28) is

\[
p(s_i | s_{\neg i}) = \text{sig} \left( \frac{1}{2\sigma^2} \left\{ \sum_{j \neq i} s_j w_{ji} + \theta_i \right\} 2 \left( s_i - \frac{1}{2} \right) \right),
\]

(30)

In the GVQ limit \( \sigma \to 0 \) this equation defines an iterative scheme which is the same as that for the mean-field method. Hence, in the limit \( \sigma \to 0 \) the mean-field method and the Gibbs sampling method are equivalent.

Appendix 2.D  Belief Propagation in the Directed Graph

Our goal is to compute marginal probabilities \( p(s | x) \) in a directed graphical model with a structure as shown in Figure 2.13 a). For graphs with tree like structures one can, analogous to the undirected case, decompose the summation into local operations. For a complete treatment of how this is done we refer to [51], here we simply state the results. For a directed network there are two types of messages, namely \( \rho \)-messages that are send in the direction of the arrows from parent nodes (binary units \( S_j \)) to child nodes (visible units \( X_i \)) and \( \lambda \)-messages that are sent in the opposite direction. The following recursive procedure is guaranteed to give the exact solution for directed graphs without cycles such as shown in Figure 2.13 b):
Generative Vector Quantisation

Figure 2.13: Graphical structure of GVQ. a) Fully connected network. b) A network structure without loops.

- The message that visible unit $X_i$ sends to hidden unit $S_j$ is given by
  \[
  \lambda_{X_iS_j}(s_j) = \sum_{s' \in \{\text{Pa}(X_i) \setminus S_j\}} p(x_i, s_j, s') \prod_{s_k \in \text{Pa}(X_i) \setminus S_j} \rho_{S_kX_i}(s'_k),
  \]  
  (31)
  where $\text{Pa}(X_i) \setminus S_j$ is the set of parent units of unit $X_i$ excluding unit $S_j$. The set of states $s'$ of this set is notated as $\{\text{Pa}(X_i) \setminus S_j\}$.

- Message from hidden unit $S_k$ to visible unit $X_j$
  \[
  \rho_{S_kX_j}(s_k) \propto \pi(s_k) \prod_{X_j \in \text{Ch}(S_k) \setminus X_j} \lambda_{X_jS_k}(s_k),
  \]  
  (32)
  where $\text{Ch}(S_k)$ is the set of child units of binary node $S_k$.

The marginal probabilities $p(s_i)$ are given by
  \[
  p(s_i) \propto \pi(s_i) \prod_{x_i \in \text{Ch}(S_i)} \lambda_{x_iS_i}(s_i),
  \]  
  (33)

As for the undirected methods it is not guaranteed that this method gives the exact result for $p(s_i|x)$ if the network contains cycles. There is evidence [73], however, that for certain structures, even with loops, the message passing scheme presented above may give good results. In contrast to the undirected algorithm, the computation of $\lambda$ messages, (31), involves a summation over an exponentially large set of states. Hence, straightforward application of the algorithm results in a method which scales exponentially with the with the number of parents of single visible nodes $X_i$. For the case of the GVQ model, the special form of the conditional probabilities, $p(x|s) = e^{-\frac{1}{2\sigma_2}[x;\Sigma_x f_{ik} s_k]^2}$, allows us to use a ‘trick’ with which the summation (31) can be computed tractably. The trick is to remove the quadratic interactions in the exponent using the identity
  \[
  \sqrt{\frac{n}{a}} e^{b^2/(4a)} = \int_{-\infty}^{\infty} dy e^{-ay^2 + by}
  \]  
  (34)
where \( i = \sqrt{-1} \) is the unit imaginary number. Application of (34) in (31) results in the following expression for the \( \lambda \) messages

\[
\lambda_{X,S_j}(s_j) \propto \int_{-\infty}^{\infty} dy \prod_k \left\{ \rho_{S_k X_i}(s_k = 0) + \rho_{S_k X_i}(s_k = 1)e^{ikf_{k i}/\sigma} \right\} e^{-y^2 + iy(s_j - f_{j i})}/\sigma. \tag{35}
\]

The integration can be done efficiently with Gaussian quadratures and its complexity scales (only) linearly with the number of parents connected to node \( X_i \).

Our implementation starts with the initialisation of \( \lambda \) and \( \rho \) messages. Then a single iteration of the algorithm consists of the following steps:

1. For each parent node \( S_j \) compute the incoming \( \lambda_{X,S_j} \) messages from all visible nodes \( X_i \) with (35);
2. For each child node \( X_i \), compute the incoming \( \rho_{X,S_i} \) messages from all the connected parent nodes \( S_k \) with (32);
3. Reduce the noise according to \( \sigma \rightarrow \sigma / \alpha \).

In our experiments we use \( \alpha = 2 \).

**Appendix 2.E  Belief Revision in the Directed Graph**

Instead of slowly reducing the noise \( \sigma \) while running the belief propagation algorithm of appendix 2.D we now formulate the algorithm directly for infinitesimal \( \sigma \). In the limit \( \sigma \rightarrow 0 \) a single state will start to dominate the summation (31). Hence, in the limit \( \sigma \rightarrow 0 \) the expression for the \( \lambda \) messages are

\[
\lambda_{X,S_j}(s_j) \propto \max_{s' \in \{Pa(X_i) \setminus S_j\}} p(x_i|s_j, s') \prod_{s_k \in Pa(X_i) \setminus S_j} \rho_{S_k X_i}(s_k). \tag{36}
\]

In this case we can no longer apply the integral ‘trick’ as in (35) since maximization can not be interchanged with integration. Hence, the computational complexity of (36) scales exponentially with number of parents in \( Pa(X_i) \).

The update equations (32) for the \( \rho \)-messages do not change in the limit \( \sigma \rightarrow 0 \). Finally, the belief revision solution for the minimising state \( s' \) of (5) is

\[
s'^*_i = \arg \max_{s_i} \prod_{X_j \in Ch(s_i)} \lambda_{X_j S_i}(s_j) \tag{37}
\]

Note that this solution does not depend on the parameter \( \sigma \) in \( p(x_i|s_j, s') \). In the implementation we use \( \sigma = 1 \).
Chapter 3

Improving Classification with Hidden Variable Models by Sequential Constraint Optimization

Abstract. In this chapter we propose a method to use multiple generative models with hidden variables for classification tasks. The standard approach to use generative models for classification is to train a separate model for each class. A novel data point is then classified by the model that attributes the highest probability. The algorithm we propose modifies the parameters of the models to improve the classification accuracy. Our approach is made computationally tractable by assuming that each of the models is deterministic, by which we mean that a data-point is associated to only a single hidden state. The resulting algorithm is a variant of the support vector machine learning algorithm and in a limiting case the method is similar to the standard perceptron learning algorithm. We apply the method to two types of hidden variable models. The first has a discrete hidden state space and the second, PCA, has a continuous hidden state space. We compare the effectiveness of both approaches on an OCR problem and on a Satellite Image recognition problem.

Adapted from: M.J.D. Westerdijk and W.A.J.J. Wiegerinck. Improving Classification with Latent Variable Models by Sequential Constraint Optimization Neurocomputing. (Accepted)
3.1 Introduction

Probabilistic graphical models, such as hidden Markov models [6], sigmoid belief networks [47] and hierarchical mixtures of experts [31] are excellently suited to discover hidden structures in complex data. In particular if prior knowledge is available about the sources that generate the observed data, such *generative models* can provide a compact representation of a data distribution that reflects its underlying structure.

In addition to this explanatory task generative models can also be used for classification. In their standard use this is done by training separate models on each class. A data pattern is then classified with Bayes decision rule which compares the likelihoods of the models for each class. This will work well if each model is an accurate representation of the ‘true’ probability distribution. In practice, however, there can be several reasons why models optimized for probability estimation will not be optimal for classification: 1) The amount of available data is not sufficient to make accurate probability models of the input distribution. The corresponding classification boundary will also be inaccurate. To improve classification it may be better to use an objective function which directly depends on the classification rate; 2) Model selection algorithms consider a limited class of structures. In practice, it is unlikely that this class contains the ‘true’ structure of the data generating process.

On the other hand, models that are directly optimized for classification *e.g.*, support vector machines [72] or multi-layer perceptrons, often operate like black-boxes and lack insight into underlying generative mechanisms. In addition, since these techniques only consider information in the data which is relevant for classification, generalization might be improved by incorporating information about the input distribution of the classes. Also, in many applications one is better able to a-priori specify the structure of the input distribution than to describe the shape of the classification boundary. For example, it is easier to give a separate a-priori description of handwritten 2’s and 9’s than to describe their differences.

Recently, Jaakkola *et al.* (1999) [28] proposed a general framework in which they tune a given set of probabilistic models such that the classification error on training data is minimized. Out of all sets of models with the same classification performance, it finds the set which is closest to the initial (prior) set of models. A natural choice for this prior set is the set of models obtained from maximum likelihood fitting. This Maximum Entropy Discrimination (MED) method is based on the maximum entropy principle. As a special case, this framework includes support vector machines. However, application of this framework on a set of generative models with a layer of hidden variables is computationally intractable. Variational techniques which can be used to optimize the log-likelihood of such models are not applicable, since in this scheme they do not provide a lower bound of the objective function any more. In this chapter we deal with this problem by using *deterministic* generative models, which have been introduced in [74, 75]. In these models, the layer of hidden variables is represented by a single state, which simplifies the framework considerably, while the principle of hidden explanatory features of the data is retained. Similar to MED our
method finds a new set of models by solving a constrained minimization problem. The method minimizes the distance to initial models found, e.g., by maximum likelihood estimation, under constraints which are given by the classification objective. In doing this, we need to make a linearity assumption. As a result the solution is in general not exact. To improve the solution, the procedure is repeated a number of times until the model parameters converge. An alternative procedure is to apply the standard perceptron learning rule [17] to the generative models. Techniques, which are based on the latter approach, are not new. They have been developed, for example, for vector quantization [36].

In section 3.3 we derive a new learning rule from the constraint minimization objective. In a recent paper [77] we have derived a similar learning rule within the MED framework for one specific type of hidden variable model. The learning rule derived here provides a more general method to optimize generative models for classification. We show that the perceptron learning rule can be seen as a limit of this technique.

We will apply the method to two types of linear hidden variable models, namely Principal Component Analysis (PCA) and Generative Vector Quantization (GVQ) and show results on a Satellite Image recognition problem and a handwritten digits recognition problem in section 3.4.

In the next section we start with our definition of hidden variable models and its special case of Linear Gaussian hidden variable models.

### 3.2 Hidden Variable Models

In generative models, it is assumed that data vectors are generated by an underlying process. In hidden variable models, this process is assumed to depend on the state $s$ of a hidden variable. The hidden variable itself is generated by an independent process. Generally, the state space of the hidden variables is (much) smaller than the state space of the data vectors, such that the hidden states can provide a compact description of the data. In this chapter, we consider generative models in which the data are modeled by a $D$-dimensional continuous variable $\mathbf{x} = (x_1, \ldots, x_D) \in \mathbb{R}^D$. The components $x_i$ are called visible units, which are organized in a visible layer. The hidden variables are represented by an $n$-dimensional vector with states $s = (s_1, \ldots, s_n)^T$. Depending on the type of model the hidden unit states $s_i$ can be discrete or continuous.

In a probabilistic framework, a generative model with model parameters $\theta$, generates visible states $\mathbf{x}$ with probability

$$p(\mathbf{x}|\theta) = \sum_s p(\mathbf{x}|s, \theta)p(s|\theta).$$  \hspace{1cm} (1)
A common approach to find the model parameters $\theta$ given a training set $D = \{x_\mu\}_{\mu=1}^P$ is to maximize the log-likelihood $L(\theta|D)$ of the model, which is defined as

$$L(\theta|D) = \sum_\mu \log p(x_\mu|\theta).$$

In this chapter we also consider deterministic generative models. In a deterministic model, only a single hidden state $s_x$ is responsible for generating a visible state $x$. We will consider such models as limiting cases of probabilistic generative models, i.e., the probability distribution of $p(s|x, \theta)$ is sharply peaked around $s_x$. In this limit, the probability of hidden states other than $s_x$ are infinitesimally small. However we can still do likelihood maximization for given data, or compute the hidden state that maximizes the probability of a data point to be generated, and we ignore the fact that the actual likelihoods and probabilities vanish in the limit.

### 3.2.1 Linear Gaussian hidden variable models

In linear Gaussian hidden variable models the distribution $p(x|s, \theta)$ is a Gaussian distribution in which the means are linear in $s$, i.e.

$$p(x|s, \theta) = ||2\pi \Sigma||^{-\frac{1}{2}} \exp \left( -\frac{1}{2} (x - Fs)^T \Sigma^{-1} (x - Fs) \right)$$

in which $\theta = (F, \Sigma)$ are model parameters. $F$ is a real valued $D \times n$ matrix with columns $f_1, \ldots, f_n$, and $Fs \equiv \sum_{i=1}^n f_is_i$. The column vectors $f_i$ of $F$ will be referred to as features.

A large number of models and methods for data analysis fit into this form. Well known examples are principal component analysis (PCA, [30]), factor analysis (FA,[5]) and vector quantization (VQ, [27]):

- **Standard PCA** is a deterministic model. The starting point is a linear Gaussian model with continuous hidden states $s_i$ which have a prior distribution $p(s)$ given by the (improper) constant distribution. The covariance matrix is proportional to the identity matrix $(\Sigma)_{ij} = \sigma^2 \delta_{ij}$. The model then corresponds to a linear subspace spanned by the features $f_i$ around which the data are uniformly scattered with variance $\sigma^2$. The deterministic model can be viewed as a $\sigma \to 0$ limit. One can show, see [66], that in this limit, the maximum likelihood solution for the features $f_i$ are the principal components of the covariance matrix of the data $\langle (x_i - \langle x_i \rangle) (x_j - \langle x_j \rangle) \rangle$ where the averages are taken over the data set . In the finite $\sigma$ case with $p(s) \sim \mathcal{N}(0, 1)$ the model is known as Probabilistic PCA (PPCA) [66].

- **Factor Analysis** is a proper probabilistic linear Gaussian model. In this model, the continuous hidden states $s_i$ have a Gaussian prior $p(s) \sim \mathcal{N}(0, 1)$. The covariance matrix is related to the features $F$ in the following manner $\Sigma =$
Chapter 3

\[ FF^T + R, \] where \( R \) is a diagonal matrix. The parameters \( F \) and \( R \) are to be determined by likelihood maximization.

- **Mixture Models** and **Vector Quantization**. The starting point is a linear Gaussian model in which the hidden states \( s \) are binary (0/1) vectors with exactly one component being 1. So the state space consists of states \( s_1 = (1, 0, 0, \ldots, 0), s_2 = (0, 1, 0, \ldots, 0), \ldots, s_n = (0, 0, 0, \ldots, 1) \). Each of these states corresponds to a unique vector \( f_i = Fs_i \). In Gaussian mixture models the covariance matrix \( \Sigma \) may depend on \( s_i \) and is found by maximum likelihood estimation through an EM-procedure.

Now consider a Gaussian mixture model with a covariance matrix proportional to the identity matrix \( \Sigma = \sigma^2 I \). In the deterministic limit \( \sigma \to 0 \), the distribution of the hidden states given a certain input \( p(s|x) \) will then peak at a single state \( s \). In that case, if we take the priors \( p(s) \) to be constant, the Gaussian mixture model reduces to vector quantization. The corresponding centers \( f_i \) are known as codebook vectors and represent cluster centers in the data. Finally, if more components of the states \( s \) are allowed to have value 1 e.g., \( s = (1, 0, 1, 0, 1, 1) \) such that the codebook vectors are additive compositions of the basic features we have **generative vector quantization** [74, 77]. We will discuss this model in more detail in section 3.4. The probabilistic counterpart of the model is known as the Cooperative Vector Quantization model [79].

### 3.3 Sequential Constraint Optimization

Suppose we have a set of training data \( \{x_\mu\} \) with corresponding class labels \( \{y_\mu\} \). We consider two-class classification problems with classes \( y_\mu = 1 \) and \( y_\mu = -1 \). A common approach for using multiple input models in classification tasks is to compare the likelihoods of an input pattern \( x_\mu \). The model that gives the pattern the highest likelihood then classifies the pattern. Equivalently, using log-likelihoods, patterns are classified with the log-odds discriminant function

\[
\mathcal{L}(x|\theta) = \log \frac{p(x|\theta^1)}{p(x|\theta^{-1})},
\]

where \( p(x|\theta^1) \) is the model trained on the class 1 data and \( p(x|\theta^{-1}) \) is trained on the class \(-1\) data. The log-likelihood function (2) which is constructed to find a good input density model is not optimally tuned for finding the best classification boundary. Our aim is to use (4) for tuning the parameters \( \theta \) of each model to improve the classification boundary.

The first assumption that we make is that the pattern \( x \) is generated by a single hidden state \( s_k \) namely that state \( s_k \) with the maximum a-posteriori (MAP) probability

\[
s_k = \arg\max_s p(s|x)p(s).
\]
Hence, we approximate the distribution over patterns \( p(x) \) with

\[
p_M (x|\theta^c) = p (x|s, F^c) p(s).
\]

In the Hidden Markov Model literature [56] the search for the MAP state \( s \) is also known as Viterbi search. Note, that in deterministic models, such as PCA, VQ and GVQ, the assumption that a pattern is generated by a single state \( s \) is already implicitly made.

The (soft) classification constraint that each training point \( x\mu \) should satisfy is

\[
y\mu L (x\mu | \theta) \geq \gamma - \xi \mu,
\]

where \( \gamma \) is a parameter which has to be fixed in advance and \( \xi \mu \) is a variable which should be positive i.e., it is subject to the constraint \( \xi \mu \geq 0 \). Suppose that both \( \gamma \) and \( \xi \mu \) in (7) are taken zero. In that case the constraint implies that \( y\mu \) and \( L (x\mu | \theta) \) have the same sign which means that all patterns must be situated at the correct side of the classification boundary. In general, with larger values, \( \gamma \) specifies a margin between the classification boundary and the data point \( x\mu \). In many practical situations it will not be possible to find a parameter setting \( \theta \) such that all training patterns are outside the margin. For this reason the slack variables \( \xi \mu \) are introduced, which allow for violations of the strict margin constraint.

While satisfying the constraints (7) and \( \xi \mu \geq 0 \) we want the new model \( \theta_0 + \Delta \theta \) to be as close to the initial model \( \theta_0 \) as possible. Furthermore, we want to minimize the training errors \( \xi \mu \). The new model that meets these criteria is found by minimizing

\[
E (\theta, \xi) = \sum_c (\theta^c - \theta^*_0)^T S (\theta^c - \theta^*_0) + C \sum_\mu \xi \mu
\]

with respect to \( \theta \) and \( \xi \mu \) subject to the constraints (7) and \( \xi \mu \geq 0 \). The local metric structure in \( \theta \)-space close to \( \theta_0 \) is parameterized by the symmetric positive definite Fisher matrix \( S = \langle \nabla \theta \log p(x|\theta)^T \nabla \theta \log p(x|\theta) \rangle \). The distance \( (\theta^c - \theta^*_0)^T S (\theta^c - \theta^*_0) \) corresponds to a second order expansion of the Kullback-Leibler divergence at \( \theta_0 \).

The Kullback-Leibler divergence is a proper measure to determine differences between probability distributions since it is invariant to arbitrary re-parameterizations of the parameters \( \theta \).

The classification constraints can be dealt with by introducing Lagrange multipliers \( \lambda_\mu > 0 \). The resulting Lagrangian

\[
L (\theta, \lambda) = \sum_c (\theta^c - \theta^*_0)^T S (\theta^c - \theta^*_0) - \sum_\mu \lambda_\mu \left\{ (y\mu L (x\mu | \theta) - \gamma) \right\}.
\]

has the same minimum for \( \theta \) as (8) in the constraint region given by (7) and \( \xi \mu \geq 0 \) if \( L (\theta, \lambda) \) is at a maximum for the Lagrange multipliers \( \lambda \) subject to the constraints \( 0 \leq \lambda_\mu \leq C \).
By setting the derivatives of $L(\theta, \lambda)$ w.r.t. to the model parameters $\theta$ equal to zero one arrives at the dual objective function $J(\lambda)$ which has to be maximized with respect to $\lambda$. In general, it is not possible to find a suitable expression for this dual objective function. In appendix 3.A we show that if the discriminant function $\mathcal{L}(x|\theta)$ is linearly dependent on the parameters $\theta$ in a region around $\theta_0$, then the $\lambda_\mu$ are given by the solution of the following quadratic form

$$J(\lambda) = L^T \lambda - \lambda^T Q \lambda,$$

(10)

with constraints $0 \leq \lambda_\mu \leq C$. The matrix $Q$ is a $P \times P$ matrix with elements $Q_{\mu,\mu'}$, formed by the inner products of the model derivatives $H(x|\theta_\mu,c) \equiv \nabla_{\theta^c} \mathcal{L}(x|\theta^c)|_{\theta^c=\theta_0}$ at points $x_\mu$ and $x_{\mu'}$.

$$Q_{\mu,\mu'} = \frac{1}{4} \gamma_\mu \gamma_{\mu'} \sum_c H(x_\mu|\theta,c)^T S^{-1} H(x_{\mu'}|\theta,c).$$

(11)

The matrix elements $Q_{\mu,\mu'}$ measure the similarity between data points $x_\mu$ and $x_{\mu'}$ according to their effect on the discriminant function. The linear contribution in (10) is given by the $P$-dimensional vector $L$ with components,

$$L_\mu = \gamma - y_\mu \mathcal{L}(x_\mu|\theta).$$

(12)

which specifies how much each pattern $x_\mu$ is in- or outside the margin (specified by $\gamma$). Recently, due to the interest in Support Vector Machines, many efficient techniques [52, 29, 11] have been developed with which the quadratic programming problem can be solved. A brief description of the method we used for this chapter is given in appendix 3.B.

After obtaining the Lagrange multipliers $\lambda$ that maximize (10) we can construct the new, improved, model parameters $\theta^c \leftarrow \theta_0^c + \Delta \theta^c$. The shift $\Delta \theta$ is given by a weighted combination of the derivatives $H$ of the discrimination error $\mathcal{L}$,

$$\Delta \theta^c = \frac{1}{2} S^{-1} \sum_\mu \lambda_\mu \gamma_\mu H(x|\theta,c)$$

(13)

Due to the nonlinearity of $\mathcal{L}$, the resulting parameters $\theta^c$ are only an approximation of the desired parameters that minimize (9). The assumption that the obtained $\theta^c$ are close to $\theta_0$, motivates us to do the optimization a second time, but now with the obtained $\theta^c$ as initial parameters. This leads us to the proposition of an iterative procedure, which we call Sequential Constraint Optimization (SCO). In SCO, the discriminant function $\mathcal{L}$ is linearized around the initial parameters $\theta_0$. For the linearized discriminant function, the optimal parameters $\theta^c$ are obtained by the above described optimization procedure. Subsequently, these parameters are used as the initialization for the next iteration. Thus the parameters are iteratively updated $\theta \leftarrow \theta^c + \Delta \theta^c$ until some convergence criterion is reached (see section 3.4).
3.3.1 The Standard Perceptron Learning Rule

The Lagrange multipliers in the Sequential Constraint Optimization algorithm are found by maximizing the quadratic form (10). The maximization problem would be greatly simplified if the quadratic terms given by the $Q$ matrix could be neglected, i.e., if we could ignore the similarities between data points according to their effect on the discrimination function. The solution for the Lagrange multipliers is then simply found by comparing the value of the discriminant function at each data-point with the margin parameter:

$$
\lambda_\mu = \begin{cases} 
C & y_\mu \cdot L(x_\mu | \theta_1) < \gamma, \\
0 & y_\mu \cdot L(x_\mu | \theta_1) \geq \gamma
\end{cases}
$$

(14)

Hence, only those patterns that are close (within a region of size $\gamma$) or at the wrong side of the classification boundary contribute with a constant weight to the change in the model parameters $\Delta \theta$. If we apply this solution to the update rule (13) we arrive at a simpler learning algorithm. In fact this version of the algorithm is closely related to a standard learning algorithm which can be found in many text books on machine learning (see for example [18]), namely the batch Perceptron algorithm (BPA). In its standard form BPA is applied to adjust the orientation of a linear classification boundary to maximize the classification score. Adapted versions of BPA have been applied to more advanced models. For example, the application of BPA to a Vector Quantization model is closely related to the Learning Vector Quantization algorithm [35]. Other choices for the loss function $\lambda(x)$ have also been proposed, such as in the LVQ2 learning algorithm [36].

A more direct correspondence between BPA and the SCO algorithm can be made by considering the limit $C \rightarrow 0$. If we make the parameter $C$ in the SCO algorithm small enough then (10) is maximized within such a small region that the quadratic term can be neglected. In that case the two algorithms have identical behavior.

One of the points we will investigate in the next section is to see what the practical differences are between the two approaches, for large $C$ and for small $C$.

3.4 Application to linear models and results

3.4.1 Application to Generative Vector Quantization

In this section we apply the SCO technique to Generative Vector Quantization (GVQ) models discussed in [74]. In a GVQ model, data points are explained in terms of binary combinations of feature vectors. We first review the basic idea of this technique and then we apply SCO to GVQ and show results on handwritten digit recognition.

Consider a generative model (1) with one hidden layer of $n$ binary units with states $s \in \{0, 1\}^n$ and a continuous visible layer (the layer corresponding to the data) with values $x = (x_1, \ldots, x_d, \ldots, x_D) \in \mathbb{R}^D$. The prior hidden state distribution $p(s)$
Figure 3.1: The cluster (circles) are generated by a small set of basic features \( f_1, f_2, \) and \( f_3 \), which correspond to the states \( (100), (010) \) and \( (001) \), respectively. The cluster corresponding to the state \( (011) \), for example, is given by the sum of the features corresponding to the two states \( (001) \) and \( 010 \) (see broken lines).

is constant, so that it does not play a role in GVQ. An example of a set of clusters generated by 3 features in a 2-dimensional space is shown in Figure 3.1. The data points for which this clustering is found are plotted with small dots. Note that the number of features \( n \) is not related to the dimensionality of the data space. Hence, considered as a basis, the feature set may be under or over complete.

Finding the features \( F_0 \) that maximize the log-likelihood (2) proceeds as follows: After initialization of the features \( F \), the GVQ learning algorithm iterates between two steps. Step 1 is finding the MAP state which associates to each data point \( x \) the single most nearby cluster center \( s_k \) (5). Step 2 is finding the optimal feature configuration \( F \) for the given MAP state. The first step is computationally difficult since, in principle, it involves a search through all \( 2^n \) binary states \( s \). In a recent paper [75] we compared different search methods, such as variational mean-field and Bayesian inference methods, for GVQ on a large variety of problems. In most situations it is possible to find a good solution within reasonable time even for large numbers of hidden variables.

Application of the Sequential Constraint Optimization algorithm to GVQ models is quite straightforward. For these linear deterministic models, the log-likelihood at a data point \( x^d \) is proportional to the ‘distance’ of the data point to the model:

\[
\log p(x^d | \theta^c) \propto - \| x^d - F^c s^c \|^2,
\]

where \( s^c \) is the solution of (5). The derivative \( H \) and the Fisher matrix \( S \) are computed
as follows: Let $z_{i\mu} = (z_{1i\mu}, z_{2i\mu}, \ldots, z_{ni\mu})$ denote the vectorial distance between data-point $x_{i\mu}$ and the model:

$$z_{i\mu} \equiv x_{i\mu} - \sum_l f_{il} s_{il},$$

where $s_{il}$ is the $l$th component of $s_x$, and $f_{il}$ are the matrix entries of $F$. The derivative of the discriminant function w.r.t. to feature component $f_{dn}$ is then

$$H(x_{i\mu}|\theta^c)_{dn} = -2 z_{d\mu} s_{n\mu},$$

and the (empirical) Fisher matrix $S$ for each class $c$ is:

$$S_{imjn} = 4/N_c \sum_{\mu} z_{i\mu} z_{j\mu} s_{m\mu} s_{n\mu},$$

where $N_c$ is the number of training patterns of class $c$.

**Results on Handwritten Digit Recognition**

The data set we used to test our method consisted of 11000 handwritten digits compiled by the U.S. Postal Service Office of Advanced Technology. We used the same preprocessed data as Saul et al. (1996) [60] and Sallans et al. (1998) [59]. Each processed image is built up out of $8 \times 8$ black and white pixels. A data sample of each digit class is shown in Figure 3.2.

In the experiments discussed below we address the following questions:

1. Is there a difference in the learning behavior between SCO and BPA in terms of convergence behavior and training time?
2. What is the influence of the initialization on the final results after training? Is it useful to start with the maximum likelihood solution instead of a random initialization?

3. Final experiments: does post-training help improving the test score over the maximum likelihood models?

First, we will discuss results of the learning behavior on the training data. We compared the behavior of BPA with SCO. For both methods we chose $\gamma_{BP} = 1$ for the margin parameter. This value gave the best convergence behavior i.e., the training score increases with the least amount of oscillations, for all values of the other parameter $C$. In contrast, the parameter $C$ has a strong impact on the convergence behavior of BPA. Taking $C$ too small results in an extremely slow decrease of the training error; if it is taken too large we get wild oscillations and there is no convergence at all. Hence, in each experiment we needed to fine tune $C$ to get convergence within reasonable time. The values found in this manner varied within the range $10^{-4} < C < 10^{-3}$. This sensitivity to $C$ is not present if we use SCO. If we take small values $C < 10^{-3}$ in SCO convergence is slow and the learning curve is, as expected, almost identical to the learning curve found with BPA. However, if we increase $C$ then SCO needs fewer and fewer iterations to converge. For values $C > 20$ the learning curve no longer depends on $C$ and convergence is reached after about 4 or 5 iterations.

To see the effect of the initialization, we did experiments starting with a random initialization and starting with model parameters obtained from maximum likelihood fitting. These experiments were performed for both SCO and BPA, so there are 4 types of experiments which we will abbreviated as follows: BPA with random and maximum likelihood initialization are abbreviated as BPA-R and BPA-ML, respectively; SCO with random and maximum likelihood initialization are abbreviated as SCO-R and SCO-ML, respectively.

Figure 3.3 shows typical learning curves on a 2-class classification problem: separating digits 3 and 5. We chose these classes for this example since these were found to be more difficult to separate than other pairs of classes. The data samples on which the models were trained consisted of 200 patterns per class. The two GVQ models each have 4 feature vectors $f_i$. Later we present a more extensive experiment where the number of features is determined with a cross-validation procedure. The three figures show learning curves plotted on a different scale. In the left subplot we see that, compared with the other procedures, BPA-R needs many iterations to converge. At the end it also reaches a smaller maximum training score than all the other procedures. In the middle subplot we rescaled the vertical axis. The lower, staircase-shaped learning curve corresponds to BPA-ML, which converges after about 80 iterations. We see that the initial maximum likelihood models (# of iterations = 0) already classify 0.9775 of the training patterns correct. In the right subplot we see the learning curves of SCO. Both SCO-R and SCO-ML converge to 1 after only 3 iterations. Although, SCO needs fewer iterations to converge, the training time needed per iter-
Sequential Constraint Optimization

Figure 3.3: Learning curves on two classes (3 and 5) of the digit data set. The plots only differ by the range of the horizontal and vertical axis. **Left:** Black solid curve: BPA -ML initialized with the maximum likelihood solution. Dotted curve: BPA -R. In both cases $C = 2 \times 10^{-4}$. **Middle:** Rescaled vertical axis. In this range, we again see two curves. The lower staircase-shaped curve is perceptron learning with maximum likelihood initialization (BPA -ML). The other curve is SCO -ML. **Right:** Here we also changed the range of the horizontal axis. The solid line is SCO with random initialization and the dashed line is SCO with maximum likelihood initialization.

Slight differences for SCO and BPA. In Figure 3.4 we plotted the CPU time as a function of the number of iterations. Clearly, in the first 2 iterations SCO requires a lot of CPU time to compute the Lagrange multipliers by solving the quadratic programming problem. The time needed is not constant but depends on the current shape of the decision boundary: if it better separates the classes, more Lagrange multipliers will be zero and the easier it is to solve the quadratic programming problem.

One important point of our investigation is to see how much the trained models differ from the original maximum likelihood solutions. For this purpose we plotted the (negative) log-likelihood against the training set score after training. To get an idea how large the effects of potential local minima and the choice of the specific training set are we repeated the experiments several times with different training samples. The results are plotted in Figure 3.5. Again, we made plots at different scales. In the left subplot we see that randomly initialized procedures end up far away from the maximum likelihood models. The SCO -R procedure is still able to maximally separate all the training data. The BPA -R procedure seems to get stuck in local minima of the classification error. In the right subplot we focused on the procedures starting with the maximum likelihood initializations, SCO -ML and BPA -ML. For comparison: the original maximum likelihood solutions had an average negative log-likelihood $-\langle \log p(x) \rangle = 1649$. We see that the solutions after applying SCO and BPA are close to this solution.

Each time we selected a training set, we also selected a (disjoint) validation set. The error rates on these data-sets are summarized in table(3.1). The first column
Table 3.1: Test error rates on discriminating between 3’s and 5’s using GVQ models with 4 hidden variables.

<table>
<thead>
<tr>
<th></th>
<th>ML</th>
<th>SCO -R</th>
<th>BPA -R</th>
<th>SCO -ML</th>
<th>BPA -ML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>6 ± 1%</td>
<td>11 ± 1%</td>
<td>24 ± 5%</td>
<td>5 ± 1%</td>
<td>5 ± 1%</td>
</tr>
</tbody>
</table>

presents the validation set score of the maximum likelihood solution without applying either SCO or BPA. Clearly, the randomly initialized procedure performs poorly. The test set score hardly improves if use SCO-ML or BPA-ML. In the next (final) experiment, we show that the test set performance does improve greatly if we use all the training data and use larger GVQ models.

In the final experiment, we used the same partitioning of the data into a training set and a test set as Saul et al. (1996) [60] and Hinton et al. (1995) [25]. The training set consisted of 700 examples of each digit and the test set consisted of 400 examples of each digit.

The multi-class classification problem was split up into 10 binary classification problems for which we trained 20 models in total. For each digit, we constructed a classification boundary between the training examples of that digit and the examples of the other digits. To do this we used the SCO procedure to train two GVQ models for each digit class, one for the positive training examples (the 700 examples corresponding to the digit) and one for the negative training examples (the $9 \times 700 = 6300$
Sequential Constraint Optimization

Figure 3.5: Fraction of mis-classification versus the log-likelihood on the training set after learning. The labeling is chosen as follows: ‘x’ = BPA -R, ‘Δ’ = SCO - R, ‘o’ = SCO -ML and ‘*’ = BPA -ML. All experiments were repeated 10 times with different randomly selected training sets. **Left:** Large ranges of the horizontal and vertical axis. **Right:** By taking a smaller range for the horizontal and vertical axis we see the difference between SCO and BPA if we initialize them both with the maximum likelihood solution.

examples corresponding to the remaining digits\(^1\). Cross-validation within the training set, where we varied the number of hidden states from 1 to 8, revealed that we had to choose at least 8 hidden units for each model. To restrict the computational overhead we did not investigate the performance of larger numbers of units. The reason for this is that, for the purpose of this chapter, we did not incorporate any approximating techniques to find the Viterbi solutions \(\mathbf{s}_i\) for each pattern \(\mathbf{x}\). As explained before, the exact algorithm scales exponentially with the number of features. There exist, however, different accurate algorithms (such as mean-field) to speed up the association step [77]. With the 8 hidden unit configuration the CPU time needed for a 500 MHz Pentium III to fit a positive and negative model for one digit was 1.4 hours.

To classify a test example \(\mathbf{x}\) we computed its distance \(E_i^c(\mathbf{x}) = \min_{\mathbf{s}} \|\mathbf{x} - M^c_\mathbf{i}(\mathbf{s})\|\) to both the positive \(c = 1\) and negative \(c = -1\) models of each class \(i\). We then computed, for each digit \(i\), the difference \(\Delta E_i\) between the distances to the positive and negative model, i.e., \(\Delta E_i = E_i^1(\mathbf{x}) - E_i^{-1}(\mathbf{x})\). The class \(i\) for which \(\Delta E_i\) was minimal was then chosen to label the test example \(\mathbf{x}\). This procedure corresponds to the intu-

\(^1\)As in Support Vector Machines, the SCO framework can also be extended to multi-class classification problems in a direct way. On the other hand, there are indications [78] that, for practical purposes, the multi-class formulation does not have a great advantage over the multiple two-class approach.
Chapter 3

Table 3.2: Test error rates on the full 10-class digit recognition problem. Each GVQ model had 8 hidden variables.

<table>
<thead>
<tr>
<th>GVQ max.likelih.</th>
<th>nearest neighbor</th>
<th>back prop.</th>
<th>wake-sleep</th>
<th>GVO-SOC / BPA</th>
<th>mean field</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.8%</td>
<td>6.7%</td>
<td>5.6%</td>
<td>4.8%</td>
<td>4.8%</td>
<td>4.6%</td>
</tr>
</tbody>
</table>

The initial GVQ models, trained by maximum likelihood fitting, misclassified 7.8% of the test patterns. After application of the SCO procedure the test error was reduced from 7.8% to 4.8%. The difference with BPA was less than 0.1%; we did not include this in the table.

Using exactly the same partitioning of the data set into train- and test-set, Hinton et al. (1995) reported test error rates of 6.7%, 5.6% and 4.8% obtained with nearest neighbor classification, a back-propagation multi-layer perceptron and generative models trained with the wake-sleep algorithm, respectively. Using the same data partitioning, Saul et al. (1996) obtained a slightly smaller error rate of 4.6% with sigmoid belief networks. In that case a single network was trained for each digit using standard (unconstrained) maximum likelihood optimization. Each network was very large compared with our GVQ models. It consisted of an $8 \times 8$ grid of visible units, a middle layer of 24 binary hidden units and a top layer of 8 binary units.

An overview of the test error results is presented in table(3.2).

3.4.2 Application to Principal Component Analysis

Principle Component Analysis reduces a high-dimensional data space to a low-dimensional linear sub-space. The components, which span the low-dimensional sub-space, are chosen such that they capture as much of the variance of the data as possible. Although, PCA is not considered to perform excellently in classification tasks it is used often for this purpose because of its transparency and clear feature-wise data representation. For this reason PCA has been used to classify, for example, images of faces [70]. In this section we apply SCO to Principle Component Analysis models and investigate whether the accuracy of such models can be improved without moving too far away from the original models.

As explained in section 12, PCA can be seen as the zero noise limit of a probabilistic generative model, namely a probabilistic principle components analyzer [66]. The hidden states $s_i$ are continuous in PCA. The Viterbi solution $s_x$ corresponding to data pattern $x$ is simply given by the projection of $x$ onto the columns $f_i$ of $F$, where the vectors $f_i$ now represent the principal components. In other words,
in PCA the components of the states $s_i$ are given by $s_i = f_i \cdot x / \| f_i \|^2$. A data pattern is classified by comparing the distances of the pattern to the projections: $\mathcal{L}(x, \mu) = -\sum c_i \| x_i - F s_i \|^2$.

Application of SCO to PCA is the same as for GVQ: the derivative $H$ and the Fisher matrix are also given by (15) and (16).

**Results**

We show results of SCO applied to PCA on the Landsat Satellite Image data set obtained from the Statlog data repository [43]. The spatial resolution of a pixel in a Landsat Image is about 80m x 80m. Each pattern in the data set corresponds to a 3x3 square neighborhood of pixels. Such a pattern contains the pixel values in the four spectral bands of each of the 9 pixels in the 3x3 neighborhood and a number indicating the classification label of the central pixel. Thus, the total number of data-dimensions is $d = 3 \times 3 \times 4 = 36$. There are six classes: 1 = ‘red soil’ (24.17%), 2 = ‘cotton crop’ (10.80%), 3 = ‘grey soil’ (21.67%), 4 = ‘damp grey soil’ (09.36%), 5 = ‘soil with vegetation stubble’ (10.60%), 6 = ‘very damp grey soil’ (23.40%).

We first show the results of the learning behavior on a 2-class classification problem. For this purpose we chose the pair of classes 2 and 4. We show the end points of learning in Figure 3.6. From this figure we see that the negative log-likelihood of the randomly initialized models is much larger than the models initialized with the maximum likelihood solution.

In this case the validation set performances were not significantly different: all methods misclassified 2 ± 1% of the test data. The initial PCA models misclassified 3 ± 1%. Hence, it seems that with PCA on this data set it does not make a difference for classification if the final models are ‘close’ to the data or not.

The time needed for learning is plotted as a function of the number of iterations in Figure 3.7. We see that SCO computation of the Lagrange multipliers does not take much time in this case.

As a final experiment we trained models on all the training data of all classes and determined the test error. The multi-class classification problem was split into 6 binary classification problems for which we trained $2 \times 6$ models in total. For each class, we constructed a classification boundary between the training examples of that class (200 randomly chosen patterns) and the examples of the remaining five classes (5 x 200 patterns). To do this we used the SCO procedure to train two GVQ models for each class, one for the positive training examples and one for the negative training examples.

In table (3.3) we show the test error rates. We see that there is a substantial improvement over using the initial PCA solutions. The test error differences between SCO and BPA are small.
Figure 3.6: Results of PCA learning on the LandSat image data. Each point represents the fraction of mis-classification versus the log-likelihood on the training set after learning. The labeling is chosen as follows: ‘x’ = BPA -R, ‘o’ = SCO -ML and ‘*’ = BPA -ML. SCO -R points fall outside the figure. All experiments were repeated 10 times with different randomly selected training sets.

Table 3.3: Test error rates on the Landsat image data-base

<table>
<thead>
<tr>
<th></th>
<th>PCA max.likelih.</th>
<th>PCA BPA</th>
<th>PCA SCO</th>
</tr>
</thead>
<tbody>
<tr>
<td>train error</td>
<td>27 %</td>
<td>16.2%</td>
<td>2.5 %</td>
</tr>
<tr>
<td>test error</td>
<td>27.4%</td>
<td>18.2%</td>
<td>17.4%</td>
</tr>
</tbody>
</table>

3.5 Discussion and Conclusions

In the standard approach generative models are optimized for maximum likelihood estimation and are therefore not directly optimized for the task of classification. In this chapter we proposed a method, sequential constraint optimization (SCO), to adjust the models to improve their performance as a combined classifier. The basic idea of our method is to use a deterministic approximation to the distribution of the hidden states within each model. While fixing this distribution we adjust the models to improve the separation of the training data. Iterating further in this manner leads to a new set of models which is better suited for the purpose of classification. We have
shown that in the limit for small upper bound $C$ this algorithm is equivalent to the Batch Perceptron Algorithm (BPA).

The main differences between SCO and BPA are: In BPA the value of $C$ is difficult to determine: taking it too large leads to oscillations and not to convergence; taking it too small leads to extremely slow learning. Adaptive adjustment of $C$ during learning could help. SCO is not sensitive to the value of $C$: a large value leads to convergence within a few iterations. The iterations in SCO are, however, computationally more demanding since we need to solve a quadratic programming in each step. The difficulty of solving the quadratic objective function depends on the problem at hand. Also our method for solving the quadratic programming problem is perhaps not very efficient. We can expect a significant speed up by using the faster methods that have been developed recently for Support Vector Machines. The methods do not differ much in their final performance.

We also investigated whether it is useful to initialize training with the maximum likelihood solution. Especially in the GVQ experiments we saw that it makes a big difference if we start with randomly initialized models or with models initialized with the maximum likelihood solution: With random initialization, the models end up far away from the data; in contrast, with maximum likelihood initialization the modified models are much closer to the input data. In the GVQ experiments this also resulted in a better test set performance.

From the experiments on the full OCR and Landsat image datasets show of all classes, we conclude that post-training (using either SCO or BPA) after maximum likelihood training does greatly improve the classification performance of the hidden variable models.

We applied the SCO method to two linear models. The method can be applied
to non-linear models e.g., sigmoid belief networks, as well. We can expect that the classification performance for these models will increase in the same manner.

Appendix 3.A Derivation of the objective function for the Lagrange multipliers

We want to find a new model $\theta$ as close as possible to the original model $\theta_0$ such that most training patterns $x_\mu$ are outside the margin (parameterized by $\gamma$). The solution is found by minimizing the Lagrangian

$$L(\theta, \lambda) = \sum_c (\theta^c - \theta_0^c) S(\theta^c - \theta_0^c) - \sum_t \lambda_t \left\{ (y_t L(x|F)) - \gamma \right\},$$

(17)

with respect to the model parameters $\theta$ and to maximize it with respect to the Lagrange multipliers $\lambda$. Since $L(x|\theta^c)$ is, in general, a non-linear function of $\theta$ we optimize (17) by considering sufficiently small steps in $\Delta \theta$. In that case the discriminant function can be approximated with the first order expansion

$$L'(\theta, \lambda) = \sum_c (\theta^c + \Delta \theta^c)^T S^{-1} (\theta^c + \Delta \theta) - \sum_t \lambda_t \left\{ \left( y_t \sum_c H_{x_\mu, \theta_0^c} \cdot \Delta \theta \right) - \gamma \right\} - \sum_t \lambda_t y_t L(x|\theta),$$

(19)

Setting the derivatives of (19) w.r.t. $\Delta \theta^c$ equal to zero gives the optimal value of $\Delta \theta^c$ in terms of the Lagrange multipliers

$$\nabla_{\theta^c} L(\theta, \lambda) = 0 \rightarrow \Delta \theta^c = \frac{1}{2} S^{-1} \sum_t \lambda_t y_t H_{x_\mu, \theta_0^c}$$

(20)

By substituting (20) into (19) we obtain the objective function $J(\lambda)$ for the Lagrange multipliers

$$J(\lambda) = \sum_{\mu} \lambda_\mu \gamma - \frac{1}{4} \sum_{\mu, \mu'} \sum_{\lambda_\mu, \lambda_{\mu'}} \sum_{H_{x_\mu, \theta_0^c}} S^{-1} H_{x_\mu, \theta_0^c} - \sum_{\mu} \lambda_\mu y_\mu \left\{ L(x_\mu|\theta) \right\}$$

(21)

where the Lagrange multipliers $\lambda_\mu$ are subject to the constraints $0 \leq \lambda_\mu \leq C$.

In a more compact notation

$$J(\lambda) = L^T \lambda - \lambda^T Q \lambda,$$

(22)
in which $Q$ is a $P \times P$ matrix with elements formed by the inner product of the derivatives $H$ of the discrimination error $\mathcal{L}$ at points $\mu$ and $\mu'$,

$$Q_{\mu,\mu'} = \frac{1}{4} y_{\mu} y_{\mu'} \sum_{c} H(x_{\mu} | \theta_{c})^T S^{-1} H(x_{\mu'} | \theta_{c}^c)$$  \hspace{1cm} (23)$$

and $L$ is a $P$-dimensional vector specifying how much each pattern $x_{\mu}$ lies at the wrong (defined by threshold $\gamma$) side of the classification boundary

$$L_{\mu} = \gamma - y_{\mu} \{ \mathcal{L}(x_{\mu} | \theta_{c}) \}.$$  \hspace{1cm} (24)$$

**Appendix 3.B  Solving the Quadratic Programming Problem**

Our method for doing the constraint optimization of the quadratic form (10) is related to the Sequential Minimal Optimization algorithm [52] which is developed for training Support Vector Machine (SVM) classifiers. At each stage in this algorithm two Lagrange Multipliers $\lambda$ are selected for optimization while holding the others fixed. This is repeated until the error is within a $\varepsilon$ distance from the (unique) maximum.

In SVM’s there is an additional constraint that the products of Lagrange multipliers with the class labels have to add up to zero $\sum_{\mu} y_{\mu} \lambda_{\mu} = 0$. In SCO we do not have this constraint and therefore we can maximize $J(\lambda)$ by doing 1-dimensional optimizations. Each optimization is done analytically for a given $\lambda$.

A new Lagrange multiplier $\lambda_{\mu}$ is selected by comparing all derivatives $\partial J / \partial \lambda_{\mu}$. We know that if $\lambda$ has reached the global maximum within the constraint region then $\partial J / \partial \lambda_{\mu} = 0$ if $0 < \lambda_{\mu} < C$. Otherwise $\partial J / \partial \lambda_{\mu} < 0$ if $\lambda_{\mu} = 0$ and $\partial J / \partial \lambda_{\mu} < 0$ if $\lambda_{\mu} = C$. The Lagrange multiplier $\lambda_{\mu}$ which violates these criteria the most is selected for optimization.

In the problems we tested, this optimization method was sufficiently fast. The speed depends, however, on the noise in the problem which determines the number of Lagrange multipliers for which $\lambda_{\mu} > 0$. 
Chapter 4

Inducing Classification Rules with Binary Hidden Variable Models

Abstract. Hidden variable models are perfectly suited to describe a distribution of structured patterns in complex data sets. This chapter studies models where the hidden variables are binary. Their values detect the presence or absence of elementary features in a data pattern. The standard approach to find such a representation is by maximum likelihood estimation. When the data are labeled, the hidden states can be associated to classes. This association can be used to form classification rules. If the models are found by un-supervised learning algorithms (maximum likelihood on the input data), this will in general produce models with a sub-optimal classification accuracy. Supervised learning algorithms are well suited for improving accuracy, but they will in general destroy the representation of the input distribution. The standard approach to optimize classification accuracy, while retaining the representation of the input distribution is maximizing the likelihood of the joint model of inputs and class labels. In this approach, input distribution and classification accuracy have equal value. In practice, however, one may want to put more emphasis on classification accuracy. This chapter proposes an alternative, which is maximizing the likelihood of the input distribution under the constraint of good classification. The constraint optimization and the maximization of joint probability methods are tested on four real world data sets. The constraint optimization shows better classification performance, whereas the joint probability model is better in describing the input distribution. The results show that it is possible to induce small sets of short but accurate classification rules from complex data sets.
4.1 Introduction

In hidden variable models it is assumed that the observed data can be explained more efficiently with a set of hidden or latent variables. Well known examples of such models are Gaussian mixture models [68], hidden Markov models [6], sigmoid belief networks [47] and hierarchical mixtures of experts [31]. Such models are well suited for un-supervised learning where the objective is to find an accurate and compact description of the structures that are present in the data. For example in cluster analysis these structures correspond to groups of similar data patterns which are well ‘separated’ from the other data. A cluster analysis provides important information in many data-mining and pattern recognition tasks.

The usual way to find the parameters of a hidden variable model is to maximize a likelihood function. However, in this chapter we are interested in models which not only describe structures in the data but which are also accurate in classification tasks. For example, suppose one has a set of face images and that each image has a class label from e.g., ‘male-female’ or ‘angry-sad-happy’. In such a case it is valuable to have a model that gives a sensible description of the clusters in the set of images. But it would be even more interesting if the model also relates these structures to the classes. Formulated in other words: it would be interesting to have an accurate model for classification which at the same time gives a better understanding of how and which components in a data pattern make it belong to the assigned class.

Note that ‘feed-forward’ models such as e.g., multi-layer perceptrons (see e.g., [9]), radial basis function networks [44] and Support Vector Machines [72] with supervised learning algorithms, are not suited for our purposes, since these models do not provide a description of the input data. Even if we would combine such models with a separate model describing the input data, this combination would still not give insight in, for example, the relevant features of an input pattern that are responsible for its classification.

The standard way to find models which are both useful for classification and for describing the ‘input’ distribution is to train separate models for each class and then afterwards classify new data points by comparing the likelihoods of each class-specific model. Results obtained in this way can be found in [60] and [25]. To improve these models for classification a class of algorithms has been suggested by [28] and [77]. The basic idea is that, as before, the class specific models are found by maximizing likelihoods but now under the constraint that the patterns are classified correctly. The constraints require that for each pattern the likelihood of, say, model A has to be larger than the likelihood of each of the other models if the pattern has class label A. The models that are found in this way are very close to the unconstrained maximum likelihood solutions but the classification accuracy is highly improved. A disadvantage is that we have a separate model for each class. This means that each class is described with its own specific features. The class-specific models do not share common features which are relevant for describing the input distribution.

In this chapter, we want a representation of all classes in a single hidden variable
model. We do this by associating class labels with hidden states. An input pattern is
then classified by the label of the most probable hidden state. The set of hidden states
with its class labels forms a set of classification rules. It is our goal to make this set
of rules such that 1) the rules accurately classify novel data patterns and 2) that the
hidden variable model provides an accurate description of the input distribution.

The standard way to achieve these goals is to maximize the joint likelihood of
patterns and their labels. So the labels are simply treated as an additional discrete in-
put attribute. In most models one can vary, during learning, the influence of the class
dimension by changing a single parameter. The optimal value of this parameter for
classification can be found by cross-validation. However, apart from this parameter,
accuracy in classification and accuracy of describing the input data are still treated
alike.

We propose a learning algorithm which improves classification accuracy. The al-
gorithm applies the idea of a constrained likelihood maximization, described above,
to a single model. By this we mean that the classification constraints are now differ-
ent: instead of comparing different models we now compare likelihoods of clusters
corresponding to different hidden states in one single model. We will compare this
algorithm with maximizing the joint likelihood.

The idea to relate classes with clusters is similar to what is done in ‘Learning
Vector Quantisation’ algorithms see [35]. The main difference is that in a Vector
Quantisation representation the clusters do not share common components. Hence in
Vector Quantisation the rules are quite trivial i.e., they are of the form: If (cluster= i)
then (class=A). In contrast, in a hidden variable model each cluster is formed by a
combination of more elementary components and each combination forms a rule, like
“If (component1 AND component2 AND component5) then (class=A)”.

The outline of the chapter is as follows: In section 4.2 we introduce the basic
class of models, namely “deterministic binary hidden variable models”, with which
we model the data. To adjust these models for classification, we present in section 4.3
two different algorithms, namely 1) the Constraint Optimization algorithm and 2) the
algorithm that optimizes the joint distribution. These algorithms are experimentally
compared on four ‘real world’ data sets in section 4.4. We end in section 4.5 with a
discussion.

4.2 Deterministic Binary Hidden Variable Models

We assume that data vectors $x$ are generated by an underlying process which is mod-
eled with probability distribution $p(x|s)$. This process depends on the state $s$ of the
hidden variable. This variable itself is generated by an independent process, here
modeled by a probability distribution $p(s)$.

We restrict ourselves to problems where the data patterns have numeric attributes
which we model with a $d$-dimensional real variable $x = (x_1, \ldots, x_D)^T \in \mathbb{R}^D$. The
hidden variables are represented by an $n$-dimensional binary vector with states $\mathbf{s} = \{s_1, \ldots, s_n\}^T$ in which $s_i \in \{0, 1\}$.

A hidden variable model with model parameters $F$ which generates visible states $\mathbf{x}$ has the form

$$p(\mathbf{x}|F) = \sum_\mathbf{s} p(\mathbf{x}|\mathbf{s}, F) p(\mathbf{s}). \quad (1)$$

In the models considered in this chapter, the state conditional probability density $p(\mathbf{x}|\mathbf{s}, F)$ is modeled with isotropic Gaussians

$$p(\mathbf{x}|\mathbf{s}, F) \propto e^{-\beta |\mathbf{x} - F\mathbf{s}|^2}, \quad (2)$$

in which $F$ is a real valued $D \times n$ matrix with columns $\mathbf{f}_1, \ldots, \mathbf{f}_n$, and $F\mathbf{s} = \sum_{i=1}^n \mathbf{f}_i s_i$. Note that this hidden variable model with its discrete hidden states is similar to the ‘Cooperative Vector Quantiser’ studied in [79], and the ‘Multiple Cause Mixture Model’ studied in [61].

The standard method to find optimal parameters $F = F_0$ so that the model best fits a set of training data $\mathcal{D} = \{\mathbf{x}_\mu\}_{\mu=1}^P$ is to maximize the log-likelihood $l(F|\mathcal{D})$ of the model on the data $\mathcal{D}$. The log-likelihood is defined as the logarithm of the probability of $\mathcal{D}$ according to the model i.e.,

$$l(F|\mathcal{D}) = \sum_{\mu} \log p(\mathbf{x}_\mu|F). \quad (3)$$

We will consider a special class of hidden variable models, namely that of deterministic hidden variable models [77]. In a deterministic model, only a single hidden state $s_\mathbf{x}$ is assumed to be responsible for generating a visible state $\mathbf{x}$, namely the state with the maximum a-posteriori (MAP) probability,

$$s_\mathbf{x} = \arg\max_\mathbf{s} p(\mathbf{x}|\mathbf{s}) p(\mathbf{s}). \quad (4)$$

This assumption is valid in the limit $\beta \to \infty$, since in this limit, the probability distribution of $p(\mathbf{s}|\mathbf{x}, F)$ will be sharply peaked around the MAP state $s_\mathbf{x}$. Otherwise, deterministic models can be considered as a MAP approximation of probabilistic models. Under the deterministic assumption, the distribution $p(\mathbf{x})$ is approximated by

$$p_M(\mathbf{x}|F) = p(\mathbf{x}|s_\mathbf{x}, F) p(s_\mathbf{x}). \quad (5)$$

In general, the computational complexity to find the MAP state scales exponentially with the number $n$ of hidden states. There exist a number of efficient algorithms that find approximating solutions of (4). Examples are variational mean-field algorithms [60], the belief revision algorithm [51] and Monte Carlo methods (see [75] for an experimental comparison of these methods for binary deterministic models). In this

\textit{Inducing Classification Rules}
Figure 4.1: The clusters (circles) are generated by a small set of basic vectors $f_1$, $f_2$, and $f_3$, which correspond to the states $(100)$, $(010)$ and $(001)$, respectively. The cluster vector corresponding to the state $(011)$, for example, is given by the sum of the vectors corresponding to the two states $(001)$ and $010$ (see broken lines).

In deterministic models we can do likelihood maximization for given data $D$ by consistently taking the MAP approximation. Well known models such as Vector quantization and Principal Component Analysis can also be seen as members of this class of deterministic models. An example of a set of clusters, each corresponding to a hidden state $s$, generated by 3 hidden variables $s_1, s_2$ and $s_3$ in a 2-dimensional space is shown in Figure 4.1. The data points for which this clustering is found are plotted with small dots.

4.3 Rule Induction with Binary Hidden Variable Models

In this section we extend the representation problem of the input space with a classification problem. We search models in which the hidden states play a double role. On the one hand, we want to be able to infer the class label $y$ from the hidden state representation $s$, while on the other hand, we want the hidden states to give a good representation of the input space, like in the previous section. To achieve this, we have to find feature vectors $f_i$ such that

1. The binary combination of features $\sum f_i s_j$ will represent a cluster of input data $\{x|x_j = s\}$ of a single class $y$. The class labels of the clusters imply a set of classification rules on their hidden representations $s \rightarrow y$;
2. The cluster centers \( \sum_{i} f_{n} s_{i} \) provide a faithful description of the input data. In other words, we want representations of the inputs in binary feature combinations.

In this section we describe two approaches to find such feature vectors.

### 4.3.1 Maximizing the Likelihood of the Input Distribution under Classification Constraints

Suppose we have a set of patterns \( \{ x_{\mu} \}_{\mu=1}^{P} \) with class labels \( \{ y_{\mu} \}_{\mu=1}^{P} \). To build a classifier we will construct class specific distributions \( p(x|y) \). Patterns \( x \) are classified by comparing these probabilities for each class \( y \).

In a hidden variable model the class specific probability distributions have the following form

\[
p(x|y) = \sum_{s \in S} p(x|s)p(s|y)p(y).
\]

Note that the summation is over the same binary state space \( S = \{0, 1\}^{n} \) for each label \( y \).

We construct the distribution \( p(x|y) \) in the following way: As discussed in section 4.2, in a deterministic model pattern \( x^{\mu} \) is generated by a single (MAP) state \( s_{x^{\mu}} \). The subset of training patterns associated with the same state \( s \) is \( D_{s} = \{ \mu | s_{\mu} = s \} \). The number of patterns with the same class label \( y \) in cluster \( D_{s} \) is \( n_{ys} = \sum_{\mu \in D_{s}} \delta_{y^{\mu}} \). Using these numbers we associate a class label \( y_{s} \) to each cluster \( s \) according to \( y_{s} = \text{argmax}_{y} n_{ys} \). With this relation between labels \( y \) and states \( s \) we form the conditional distribution of states \( s \) given label \( y \) as follows

\[
p(s|y) = \frac{1}{N_{y}} \delta_{ys},
\]

where \( N_{y} = \sum_{s} \delta_{ys} \) is the total number of states with the same class label \( y \). Using these statistics we construct the class conditional distribution of the inputs as follows

\[
p(x|y) = \sum_{s} p(x|s)p(s|y)p(y) \approx p(x|s_{y}^{*}) \frac{1}{N_{y}} p(y),
\]

where

\[
s_{y}^{*} = \text{argmax}_{s} p(x|s)p(s|y) = \text{argmax}_{s \in S_{y}} p(x|s),
\]

where \( S_{y} \) is the set of states \( s \) with the same label \( y \). For the prior distribution of the classes \( p(y) \), we use the empirical distribution of the classes in the training set i.e., \( p(y) = n_{y} / P \), where \( n_{y} \) is the total number of patterns with label \( y \).
To assess the quality of the classifier we define the log-odds discriminant function

\[ \mathcal{L}_{yy}(\mathbf{x}) \equiv \log \frac{p(\mathbf{x}|y)}{p(\mathbf{x}|y')}, \]

which measures how well the classes \( y \) and \( y' \) are discriminated at point \( \mathbf{x} \). If we take the maximum likelihood solution \( F = F_0 \) found by maximizing (3) the performance of the resulting classifier will not be optimal. Our aim is to use (10) for tuning the parameters \( F \) to improve the classification boundary. The central idea is to find a maximum likelihood solution of (3) but now under the constraint that the patterns are well classified. This is done by requiring each training point \( \mathbf{x}_\mu \) to satisfy the following (soft) classification constraints

\[ \mathcal{L}_{y'\mu}(\mathbf{x}_\mu) \geq \gamma - \xi_\mu \quad \forall y \neq y', \]

where \( \gamma \) is a parameter which has to be fixed in advance and \( \xi_\mu \) is a variable which should be positive i.e., it is subject to the constraint \( \xi_\mu \geq 0 \).

The parameter \( \gamma \) specifies a margin between the class conditional probabilities at the data point \( \mathbf{x}_\mu \). In many practical situations it will not be possible to find a parameter setting \( F \) such that at each training point the differences between the log-probabilities of the correct and incorrect classes are larger than the margin \( \gamma \). For this reason the slack variables \( \xi_\mu \) are introduced, which allow for violations of the strict margin constraint.

While satisfying the constraints (11) and \( \xi_\mu \geq 0 \) we want the new model \( F \) to be as close to the initial model \( F_0 \) as possible. Furthermore, we want to minimize the training errors \( \xi_\mu \). The new model that meets these criteria is found by minimizing

\[ E(F, \xi) = (F - F_0)^T S (F - F_0) + C \sum_{\mu \neq \nu} \sum_{y \neq y'} \xi_{\mu\nu} \]

with respect to \( F \) and \( \xi_\mu \) subject to the constraints (11) and \( \xi_\mu \geq 0 \). The local metric structure in \( F \)-space close to \( F_0 \) is parameterized by the symmetric positive definite Fisher matrix \( S = \langle \nabla_F \log p(\mathbf{x}|F) | \nabla_F \log p(\mathbf{x}|F) \rangle \). The distance \((F - F_0)^T S (F - F_0)\) corresponds to a second order expansion of the Kullback-Leibler divergence at \( F_0 \). The Kullback-Leibler divergence is a proper measure to determine differences between probability distributions since it is invariant to arbitrary re-parameterizations of the parameters \( F \).

The classification constraints can be dealt with by introducing Lagrange multipliers \( \lambda_{\mu y} > 0 \). The resulting Lagrangian

\[ L_{\mu}(F, \lambda) = (F - F_0)^T S (F - F_0) - \sum_{y \neq y'} \lambda_{\mu y} \left\{ \mathcal{L}_{y'\mu}(\mathbf{x}) - \gamma \right\} \]
has the same minimum for $F$ as (12) in the constraint region given by (11) and $\xi_{\mu y}^y \geq 0$ if $L(F, \lambda)$ is at a maximum for the Lagrange multipliers $\lambda$ subject to the constraints $0 \leq \lambda_{\mu y}^y \leq C$.

For fixed $\lambda$ and $s_{x y}^y$ the solution for $F$ that minimizes (13) is

$$F = [S^T + V]^{-1} \left( S^T F_0 + \beta \sum_{\mu} x_{\mu} \sum_{y \neq y_{\mu}} \lambda_{\mu y}^y \{ s_{y y}^y - s_{y y}^y \} \right),$$

where

$$V = \beta \sum_{\mu} \sum_{y \neq y_{\mu}} \lambda_{\mu y}^y \left( s_{y y}^y - s_{y y}^y \right).$$

To find the solution for $\lambda$ we must substitute (14) back into (13). The resulting objective function is a complicated function of $\lambda$. In a previous paper [76] we have found a second order approximation of this objective function in $\lambda$. These approximations imply, however, that the solutions for $F$ and $\lambda$ are only valid for constant associations $s_{x y}^y$ in a ‘small’ region around $F_0$ and for small $\lambda$ (or equivalently, small $C$). We therefore propose an iterative procedure, Sequential Constraint Optimization (SCO), where, after finding a solution for $F$, we repeat the procedure again but now replacing $F_0$ with $F$ and so on.

In SCO the use of a second order solution for $\lambda$ has some practical advantages e.g., fast convergence. In [76] we showed that in many practical situations one will, however, arrive at the same final solution $F$ by using a first order approximation for $\lambda$ which is simply given by

$$\lambda_{\mu y}^y = \begin{cases} C & \mathcal{L}_{y y_{\mu}}(s_{\mu}) < \gamma, \\ 0 & \mathcal{L}_{y y_{\mu}}(s_{\mu}) \geq \gamma. \end{cases}$$

Note from (16), (15) and (14) that $\beta$ and $\lambda_{\mu y}^y$ always appear as a product, so that $\beta C$ effectively operates as a single parameter. Note also that if $\beta C \to 0$ nothing will happen i.e., $F = F_0$. In this chapter we will use (14) and (16) in the iterative SCO procedure, which is summarized in the following subsection.

The SCO Algorithm

In summary, the algorithm that we propose to find the parameters $F$ consists of the following 5 steps:

**Step 0:** Find the maximum likelihood solution $F_0$ that maximizes (3). Set

$$F \leftarrow F_0.$$
**Chapter 4**

**Step 1:** Determine $s_{\mu} \leftarrow \arg\min \|x^\mu - Fs\|^2$ for $\mu = 1, \ldots, P$.

With $\mathcal{R}_s = \{\mu | s_{\mu} = s\}$, determine the resulting statistics $n_{ys} = \sum_{\mu \in \mathcal{R}_s} \delta_{ys\mu}$ and labels $y_s = \arg\max_y n_{ys}$.

**Step 2:** Associate binary states with patterns $x^\mu$ and labels $y$:

$$s_{y^{\mu}} \leftarrow \arg\min \|x^\mu - Fs\|^2 \text{ for } \mu = 1, \ldots, P \text{ and } y = 1, \ldots, n_c.$$  

Here $S_y = \{s | y_s = y\}$ is the set of all states with label $y$.

**Step 3:** Set the Lagrange multipliers according to (16).

**Step 4:** Update the weights $F$ with (14) and return to step 1.

4.3.2 Maximizing the Likelihood of the Joint Distribution

An alternative procedure to incorporate class information into the model is to use the joint probability distribution $p(x, y)$ of patterns $x$ and labels $y$. The class dimension is then simply treated as an additional (discrete) data dimension. The objective is then to maximize the joint log-likelihood of the model

$$L(F, \mathcal{D}) = \sum_{\mu} \log p(x^\mu, y^\mu | F) = \log \sum_s p(x^\mu | s, F)p(y^\mu | s)p(s)$$

on the data set $\mathcal{D} = \{(x^\mu, y^\mu)\}_{\mu=1}^P$ with respect to the parameters $F$.

As discussed in section 4.2, in deterministic hidden variable models a single state $s_{(x, y)}$ is responsible for generating the pattern $(x, y)$:

$$\sum_s p(x | s)p(y | s)p(s) \approx p(x | s_{(x, y)})p(y | s_{(x, y)})p(s_{(x, y)})$$

For fixed associated states $s_{(x, y)}$, it is easy to show that the optimal choice for the probability distribution $p(y | s)$ is

$$p(y | s) = \frac{n_{ys}}{n_s},$$

where $n_s$ is the number of elements in $\mathcal{R}_s = \{\mu | s_{\mu} = s\}$ and $n_{ys} = \sum_{\mu \in \mathcal{R}_s} \delta_{ys\mu}$. The weights $F$ are found by minimizing the quadratic form $\sum_\mu \|x^\mu - Fs_{(x, y)}^\mu\|^2$ with respect to $F$ which is a standard unconstrained quadratic optimization problem (see [77] for a solution).

We use a constant prior distribution $p(s)$ for the hidden states $s$ i.e., $p(s) = \text{constant}$, so this term does not play a role here.
Inducing Classification Rules

For fixed $n_y$, $n_s$ and $F$, the associated states $s_{(x^u,y^u)}$ are given by

$$s_{(x^u,y^u)} = \arg\min_s \left\{ \beta \| x^u - Fs \|^2 - \log \frac{n_{w^u}}{n_s} \right\}$$

(20)

Hence, for deterministic hidden variable models, the joint log-likelihood can be maximized using a three step procedure:

**Step 0:** Initialize $F$ with the maximum likelihood solution $F_0$ of (3).

**Step 1:** For fixed $F$, $n_y$ and $n_s$ determine $s_{(x^u,y^u)}$ with (20).

**Step 2:** For fixed $s_{(x^u,y^u)}$ determine $F$, $n_y$ and $n_s$ and return to step 1.

This procedure is guaranteed to converge to a local maximum of the MAP-approximated joint log-likelihood.

After training, a novel unlabeled pattern $x$ is classified by first determining the ‘closest’ state $s_x = \arg\min_s \| x - Fs \|^2$ and then determining the label $y_x$ by comparing frequencies i.e., $y_x = \arg\max_y n_{s_{x,y}}$.

### 4.4 Results

We tested and compared the algorithms on 4 data sets, namely Iris, an OCR data set, the Landsat data set and a speech data set for emotion classification. In the next subsection we give a brief description of each of these data sets. In section (4.4.2) we compare the classification performances of the algorithms on these data sets. Rule-extraction results are presented in section 4.4.3

#### 4.4.1 Description of the data sets

**Iris** Perhaps the best known benchmark data-set is the Iris data set. Iris is a small 4 dimensional data set consisting of 150 patterns with 3 possible classes. Each pattern contains the measurements of some relevant dimensions of an Iris plant. The objective is to predict the type of Iris based on the measurements.

**OCR** This data set [27] consists of 11000 handwritten digits compiled by the U.S. Postal Service Office of Advanced Technology. Each processed image is built up out of $20 \times 20$ black and white pixels. In the experiments below we simplified the problem to a three class classification problem, using the classes ‘two’, ‘three’ and ‘five’. For each of these classes we sampled 100 digits to get a data set of 300 patterns.

---

1The Iris data set can be obtained from the UCI Machine Learning Database Repository [http://www.ics.uci.edu/~mlearn/MLRepository.html](http://www.ics.uci.edu/~mlearn/MLRepository.html)
LandSat The Landsat Satellite Image data set is obtained from the Statlog data repository [43]. The spatial resolution of a pixel in a Landsat Image is about 80m x 80m. Each pattern in the data set corresponds to a 3x3 square neighborhood of pixels. Such a pattern contains the pixel values in the four spectral bands of each of the 9 pixels in the 3x3 neighborhood and a number indicating the classification label of the central pixel. Thus, the total number of data-dimensions is $d = 3 \times 3 \times 4 = 36$. There are six classes: 1 = ‘red soil’ (24.17%), 2 = ‘cotton crop’ (10.80%), 3 = ‘grey soil’ (21.67%), 4 = ‘damp grey soil’ (09.36%), 5 = ‘soil with vegetation stubble’ (10.60%), 6 = ‘very damp grey soil’ (23.40%). For LandSat we simplified the problem to the classes: ‘cotton crop’, ‘damp grey soil’ and ‘soil with vegetation stubble’. For each of these classes we sampled 200 digits to get a data set of 600 patterns.

ASSESS The ASSESS (Automatic Statistical Summary of Elementary Speech Segments) data set, compiled by the psychology group of the Queens Univ. in Belfast [62], consists of features from speech recordings. Each recording came from one of 40 subjects reading a passage where they expressed one out of the five emotions - fear, anger, happiness, sadness, and neutrality. In total there were 570 labeled (5 classes) patterns each corresponding to a passage. Each pattern consists of 69 numerical attributes. These attributes contain temporal statistical summaries of speech features such as pitch contours, and of features containing spectral information e.g., the value of the fundamental frequencies, and spectral energy.

4.4.2 Classification Results

Each data set was split up in 1/5 test set and 4/5 training set. The models were trained on the training set using SCO and joint learning, respectively. In input learning, the parameters $F_0$ are obtained by unsupervised learning on the basis of the input patterns. SCO and joint learning were initialized with model parameters $F_0$ obtained by input learning. The procedure was repeated 5 times with disjoint test sets. Each experiment was done with different numbers $n_f$ of hidden variables. As explained in section 4.3.2, the final solution of the ‘joint’ learning algorithm depends on the parameter $\beta$. In each experiment we tuned this parameter with an additional cross-validation loop within the 4/5 training for optimal classification performance. In SCO, the parameter $\beta C$ acts as a learning parameter which only affects ‘smoothness’ of the learning curve: if $\beta C$ is taken too large the model parameters will oscillate strongly and will not converge to an optimum, if it is taken too small learning will be very slow. Nice learning behavior was obtained with $\beta C = 0.001$ which we therefore used in all our experiments.

Each experiment produced a set of model parameters $F$, and a set of states $s$ with class labels $y_s$. In the unsupervised model $F_0$, class labels of hidden states were de-
Inducing Classification Rules

Figure 4.2: Classification scores on the training set (fraction correct). The solid line (-) corresponds to SCO, the dashed line (- -) corresponds to Joint learning and the dotted line (...) corresponds to learning only the input.

As defined according to step 1 of the SCO algorithm. New un-labeled patterns are classified by the class label of the closest state argmin \( s \). The input error was obtained by computing standard deviations over the 5 experiments.

As expected the classification scores on the training set increase (or at least remain constant) for all algorithms with the complexity of the models.

In terms of classification performance, application of SCO results in the best ‘fit’ in the majority of cases. The difference between joint learning and SCO is largest on the OCR and the ASSESS databases.

To see how this increased classification performance affects the likelihood values we plotted the input error in Figure 4.3. This error is defined as follows: First let \( -l = \sum \mu \| x^\mu - F s^\mu \|^2 \) be the ‘distance’ (=negative log-likelihood) of model \( F \) to the data \( \{ x^\mu \} \). Let \( -l_0 \) be the distance between the data and the initial maximum likelihood solution \( F_0 \). The input error as plotted in Figure 4.3 is defined as \( (l - l_0)/l_0 \). So, the larger this value the further the cluster centers have moved away from the data points. In general, we see that the SCO algorithm moves the model away from the input data. Apparently, this is the price that is payed for a better fit of the class labels. For joint learning this tendency is less profound. Surprisingly, on the LandSat data
Figure 4.3: The input error on the training set as a function of the number $n_f$ of hidden variables. The error compares the likelihood $l$ of each solution with the maximum likelihood solution $l_0$ as follows: $\text{input error} = (l - l_0)/l_0$. The solid line (−) corresponds to SCO, the dashed line (−−) corresponds to Joint learning.

base, the likelihood score is better after post-training (with either SCO (for $n_f > 3$) or the joint optimization algorithm) than that of the maximum likelihood solution. One explanation could be that the extra class information helped in finding a better local maximum of $l$.

The classification scores on the test sets are plotted in Figure 4.4. The overall performance of the algorithms on the test sets has the same qualitative properties as on the training sets: SCO has a better (or at least equal) classification performance on all data sets. Both supervised methods are clearly better than the un-supervised. As expected, if the complexity of the model is increased, then at some point the model starts to over-fit the training data and at that point the test set score will start to decrease.

4.4.3 Rule Extraction

We will now present some of the induced rule sets. The condition of each rule corresponds to a vector $v_s = \sum s_i f_i$ in the data space which is a representative of a group of data points $\mathcal{D}_s$. The vector $v_s$ is labeled with the most frequent class in $\mathcal{D}_s$. This label forms the outcome of the rule. The tables 4.1–4.4 show the rules obtained with
Inducing Classification Rules

Figure 4.4: Classification scores on the test set (fraction correct). The solid line (−) corresponds to SCO, the dashed line (−−) corresponds to Joint learning and the dotted line (..) corresponds to learning only the input.

the SCO algorithm from the experiments corresponding the first test-set/training-set division. Together with each rule $s \rightarrow y$ we show the fraction (the column ‘qys’ in the tables) of training points in $\mathbb{R}_s$ that have label $y$. The larger this fraction, the less noisy the rule. The fraction ‘qys’ of training points which are associated to each rule is given in the last column of each table. This value expresses how ‘important’ the rule is relative to the others. By comparing the values ‘qys’ in the left (= joint leaning) and right (= SCO) tables we see that after learning the rules obtained by SCO are less noisy than the ones obtained by joint learning.

For most data sets discussed here it is not straightforward to find an interpretation of the rules in data space which can be visualized easily. An exception are image analysis problems where we can visualize the vectors $f_i$ and $v_k$ as images. Each row in a sub-figure corresponds to a classification rule in table(4.1). The rules are represented in the same order as in this table. Together with the state of each hidden variable, $s_i = 0$ or $s_i = 1$, we displayed the corresponding vector $f_i$. The values the individual dimensions of this vector vary between 0 and 1 and are represented as pixels with grey values (‘1’ is black and ‘0’ is white). The left most column in each sub-figure plots the vectors $v$ which are correspond to the sum $v = \sum_i f_i s_i$. The class labels that are given to these vectors can be found in the column ‘y’ in table(4.1). If we look, in Figure 4.5a), at the vector $v_k$ (fifth column) corresponding to state
Table 4.1: Classification rules for \textit{OCR} using the maximum likelihood solution (left) and using the result from the SCO algorithm (right). The values for \(s_1, s_2, \ldots\) are the binary state values. The value of the class label \(c\) is given in the fourth column, the values correspond to 1: ‘two’, 2: ‘three’ and 3: ‘five’, respectively. The column ‘\(qs_s\)’ shows the fraction of training points \(n_s/P\) in cluster \(s\). The column ‘\(qys\)’ shows the fraction \(n_s/s1, s2, \ldots\) in cluster \(s\). The test set scores of the rules are 68% (left table) and 93% (right table).

\begin{tabular}{cccc}
\hline
\(s_1\) & \(s_2\) & \(s_3\) & \(y\) & \(qys\) & \(qs\) \\
\hline
0 0 0 & 1 & 0.94 & 0.08 \\
0 0 1 & 1 & 0.96 & 0.12 \\
0 1 0 & 3 & 0.02 & 0.21 \\
0 1 1 & 2 & 0.02 & 0.21 \\
1 0 0 & 1 & 0.86 & 0.13 \\
1 0 1 & 1 & 0.03 \\
1 1 0 & 3 & 0.03 & 0.15 \\
1 1 1 & 2 & 0.08 \\
\hline
\end{tabular}

\begin{tabular}{cccc}
\hline
\(s_1\) & \(s_2\) & \(s_3\) & \(y\) & \(qys\) & \(qs\) \\
\hline
0 0 0 & 1 & 0.95 & 0.08 \\
0 0 1 & 1 & 0.95 & 0.08 \\
0 1 0 & 3 & 0 & 0.16 \\
0 1 1 & 2 & 0.02 & 0.22 \\
1 0 0 & 1 & 0.97 & 0.13 \\
1 0 1 & 1 & 1 & 0.04 \\
1 1 0 & 3 & 0.02 & 0.18 \\
1 1 1 & 2 & 0.04 & 0.1 \\
\hline
\end{tabular}

Table 4.2: Classification rules for \textit{Landsat} using the maximum likelihood solution (left) and using the result from the SCO algorithm (right). The values for \(s_1, s_2, \ldots\) are the binary state values. The value of the class label \(c\) is given in the fourth column, the values correspond to 1: ‘cotton crop’, 2: ‘damp grey soil’ and 3: ‘soil with vegetation stubble’, respectively. The test set scores of the rules are 72% (left table) and 90% (right table).

\begin{tabular}{ccccc}
\hline
\(s_1\) & \(s_2\) & \(s_3\) & \(s_4\) & \(s_5\) & \(y\) & \(qys\) & \(qs\) \\
\hline
0 0 1 1 0 & 3 & 0 & 0.18 \\
0 0 1 1 1 & 3 & 0.04 & 0.05 \\
0 1 0 0 0 & 1 & 1 & 0.11 \\
0 1 1 1 1 & 2 & 0.07 & 0.38 \\
1 0 1 1 1 & 3 & 0.29 & 0.04 \\
1 1 0 0 0 & 1 & 1 & 0.1 \\
1 1 1 1 1 & 3 & 0.25 & 0.03 \\
\hline
\end{tabular}

\begin{tabular}{ccccc}
\hline
\(s_1\) & \(s_2\) & \(s_3\) & \(s_4\) & \(s_5\) & \(y\) & \(qys\) & \(qs\) \\
\hline
0 0 1 1 0 & 3 & 0 & 0.14 \\
0 1 0 0 0 & 1 & 1 & 0.12 \\
0 1 1 0 1 & 2 & 0 & 0.04 \\
0 1 1 1 1 & 2 & 0.01 & 0.27 \\
1 0 0 1 1 & 1 & 1 & 0.03 \\
1 0 1 1 1 & 3 & 0 & 0.07 \\
1 1 0 0 0 & 1 & 1 & 0.06 \\
1 1 1 1 1 & 3 & 0 & 0.13 \\
\hline
\end{tabular}

(1011) we see that the image neither shows a ‘three’ nor a ‘two’; it is somewhat in between these two. In contrast, in Figure 4.5b) this vector is clearly a ‘two’. The same holds for the state (1111): in subplot a) the image shows a mix between a ‘three’ and a ‘five’, after training with SCO it clearly shows a ‘three’. In order to get this improvement some of the other images have become somewhat corrupted, see for example, the images corresponding to states (0101) and (0111).
### Inducing Classification Rules

Table 4.3: Classification rules for Iris using the maximum likelihood solution (left) and using the result from the SCO algorithm (right). The values for $s_1, s_2, \ldots$ are the binary state values. The value of the class label $c$ is given in the fourth column. The test set scores of the rules are 94.6% (left table) and 96.6% (right table).

<table>
<thead>
<tr>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$s_3$</th>
<th>$y$</th>
<th>$q_{ys}$</th>
<th>$q_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0.28</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.16</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0.14</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0.18</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$s_3$</th>
<th>$y$</th>
<th>$q_{ys}$</th>
<th>$q_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0.16</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0.21</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.25</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0.19</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0.14</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Table 4.4: Classification rules for ASSESS using the maximum likelihood solution (left) and using the result from the SCO algorithm (right). The values for $s_1, s_2, \ldots$ are the binary state values. The value of the class label $c$ is given in the fourth column, the values correspond to 1:'neutral', 2:'sad', 3:'afraid', 4:'angry' and 5:'happy', respectively. The test set scores of the rules are 32% (left table) and 42% (right table).

<table>
<thead>
<tr>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$s_3$</th>
<th>$s_4$</th>
<th>$y$</th>
<th>$q_{ys}$</th>
<th>$q_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0.37</td>
<td>0.05</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0.31</td>
<td>0.1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>0.48</td>
<td>0.09</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.29</td>
<td>0.06</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0.35</td>
<td>0.09</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>0.45</td>
<td>0.05</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0.39</td>
<td>0.05</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0.36</td>
<td>0.07</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>0.44</td>
<td>0.07</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0.27</td>
<td>0.04</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0.46</td>
<td>0.16</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>0.55</td>
<td>0.08</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$s_3$</th>
<th>$s_4$</th>
<th>$y$</th>
<th>$q_{ys}$</th>
<th>$q_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.64</td>
<td>0.03</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0.31</td>
<td>0.1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0.69</td>
<td>0.18</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0.39</td>
<td>0.24</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>0.36</td>
<td>0.25</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>0.59</td>
<td>0.14</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>0.79</td>
<td>0.07</td>
</tr>
</tbody>
</table>
Figure 4.5: Extracted classification rules for OCR corresponding to the binary rules in table(4.1). a) maximum likelihood solution. b) The SCO solution. The first four columns show the vectors $f_1, \ldots, f_4$ (Note that $s_4$ is not included in table(4.1), since it corresponds to the ‘origin’ vector which is always ‘on’). The value 0,1 of the corresponding state variable is shown on top of each vector. The most right column shows the corresponding cluster centers $\sum_i f_i s_i$.

4.5 Discussion and Conclusions

In this chapter we aimed to build hidden variable models for classification and input representation. The idea is that the hidden state describes data in terms of the presence or absence of certain features that are typical for the data set. Classification is performed by applying rules on the hidden states by which data points are described. The rationale is to make the model more transparent, so that a user can get insight how a data point is classified and what features are important in the classification of a certain data point.

We compared two learning algorithms which optimize the classification accuracy and input likelihood. The idea behind the first algorithm is to optimize the input likelihood under constraint of optimal classification. In the resulting algorithm, which we called SCO, the feature vectors are incrementally updated on the basis of wrongly classified patterns. The other, more standard approach, maximizes the likelihood of the joint probability distribution of patterns and labels.

We tested and compared the algorithms on 4 different data sets. The SCO algo-
Inducing Classification Rules

Algorithm showed a superior performance as a classifier. To get this classification behavior the SCO models performed slightly worse in fitting the distribution of the input patterns. In real world tasks where we have a finite amount of data, it is in general not possible to find a model which performs optimal as a classifier and is at the same time optimal for describing the input distribution. In most applications, good classification will be more desirable than good input fitting. In such cases, the results indicate that SCO is preferable.

Note that the results obtained here, especially on OCR and LandSat, should not be compared with state of the art performances reported elsewhere e.g., in [39] and [43]. For the purposes of this chapter, we only used a small sample of these data-sets for training (4% of the full data set in OCR and 14% of the data in LandSat). It can be expected that with larger training sets more complex rules can be induced with a better performance. In a previous chapter [77], we demonstrated that with deterministic binary hidden variable models a classifier can be constructed with state of the art performance on OCR: better than reported results of multi-layer perceptrons and nearest neighbor classifiers, and comparable with large sigmoid belief networks [60]. In the experiments in [77] constraint optimization was used to train a large set of classifiers, two for each class. This approach can not be used for rule induction since all the hidden states in each of these models correspond to the same class.

For each data-set we presented typical rule sets that were induced with the SCO algorithm. These were given as relations between binary states and class labels. The set of binary variables is constructed such that it forms a minimal set of variables with which to identify clusters of uniquely labeled patterns. Analysis of these rules reveals that some variables play a role in discriminating between different classes and that other variables are not correlated with class labels i.e., the classification rules describe some general property of the input distribution. Having identified the role that each variable plays in identifying clusters and their classes, the next question will be how to interpret the variables in data space. In other words, how do we give meaning to the feature vectors $\mathbf{f}_i$ and their combinations? In image analysis tasks this is not difficult since the vectors themselves can be visualized as images. However, in problems where the data patterns $\mathbf{x}$ are represented as a collection of values which are meaningless human users, it will require more effort to assign meaning to the feature vectors and the classification rules. But even if feature vectors are difficult to interpret in terms of the original data dimensions, the relations between the hidden variables and class labels may still provide useful insights into the structure of data, albeit on a more symbolic level.

The alternative to finding hidden rules is to induce rules that operate directly on the data space like in decision trees, see [55], or in the rules extracted from neural networks, see [1]. These rules depend on the properties of individual components of data patterns $\mathbf{x}$, for example like in the rule: IF ($x_1 > \theta_1$ and $x_5 < \theta_3$) THEN (class=A). Such rules may provide insight if the variables $x_i$ have clear meaning like ‘blood pressure’ in medical diagnosis. In data where components interact on a more global level, these rules may be less useful. An example is in image analysis, where
Chapter 4

the $x_i$ correspond to pixel values. For such data a more global approach, like our hidden variable model approach, is desired.

The generalization accuracy of a set of rules will also depend on the shape of classification boundary which they try to describe. The ‘input’ rules, found by decision trees, are restricted to describing ‘block-shaped’ classification boundaries whereas our ‘cluster’ rules together form a Voronoi tessellation in which the classification boundary is part of the boundary between Voronoi regions.

In the models considered in this chapter, the position of the cluster centers depends linearly on the component vectors $f_i$. For some applications, a non-linear model could be more useful. For example, if the data variables are not numeric but binary or nominal then a sigmoid or a soft-max function could be used to relate the component vectors with the data\(^2\). To find an update rule for the vectors $f_i$ in SCO we could then use a linear approximation of these functions, which is allowed as long as we make small steps at each iteration.

In this chapter we assumed that the prior probability distribution $p(s)$ of the hidden states was constant. This restriction is not necessary. For example, one can bias the final representation to be sparse \(i.e.,\) that in each binary state vector only a small fraction of the components have value one. Such representations have been found to reveal interesting properties of the data, see for example [50].

We studied algorithms that balance in between supervised and un-supervised learning. Such methods have been investigated by others [15, 49] in a different context. There the objective is to construct classifiers using labeled as well as unlabeled training data. The idea is that the unlabeled data, which provide information about the input distribution, help to improve the generalization performance of the classifier. Clearly, the methods presented here also fall in this category, but we did not yet experimentally investigate the power of unlabeled data for classification.

\(^2\)So instead of $(x - \sum_i f_i s_i)$ we would get $(x - \sigma(\sum_i s_i f_i))$, where $\sigma(\cdot)$ is a sigmoid or soft-max function.
Chapter 5

Comparing mixture models with decision trees for case specific attribute selection

Abstract. Case specific attribute selection (CSAS) is the process of sequentially selecting attributes in order to make a reliable classification with the least amount of resources. The relevance of an attribute does not stand on itself but depends also on other attribute values that have already been observed. Hence the process is specific for the case under consideration. We examine two different approaches that address CSAS. The first approach uses mixture models to estimate a parametric model for the joint probability distribution of the attribute and class values. In order to obtain a high CSAS performance, a deterministic annealing procedure is used to fit the mixture models to the data. The second, more traditional approach, is to use a decision tree for the purpose of case specific attribute selection. In this chapter, the two approaches are compared within a common probabilistic framework. The performances of the mixture models and decision trees are contrasted on a variety of real world data-bases. The results show that the use of mixture models is better suited for case specific attribute selection.
5.1 Introduction

In many application domains it is important to have a classifier that is not only highly accurate but also makes predictions using the least amount of resources. For example, in the medical domain, diagnosing a disease may require a series of lab tests. In such a situation, it would be beneficial if the doctor had a system that predicts which of the tests are most relevant for diagnosing the patient under examination. In many traditional machine learning schemes this desideratum does not play a role in designing the classifier. Indeed, more commonly, the objective is to construct a classifier which will accurately predict the class $k$ of a novel randomly drawn attribute vector $x$ given a set of training examples $\{x^\mu\}$ and corresponding class labels $\{k^\mu\}$. In contrast to this approach, it is our concern to use the training data to construct a classifier for which, given a novel but now incomplete example $x_d$, we can determine which single extra attribute most likely determines the correct classification of this example. We shall hereafter refer to this procedure as ‘case specific attribute selection’ (CSAS).

There are various ways to address this issue. A well known classification method which has been extensively studied within the AI community over the last decade is to use induced decision tree classifiers. As we shall later describe, decision trees can also be adapted to solve the problem of case specific attribute selection. However, it is also well known that decision trees have limitations in terms of classification ability. For example, the decision boundaries in a tree are generally restricted to a certain shape which may not be optimally suited for particular classification problems [46], [13]. On the other hand, there exist more flexible classifiers, which provide good classification performance even in highly non-linear and noisy problems. In this chapter we shall formulate the use of these more flexible methods for case specific attribute selection, comparing and contrasting their performance with decision trees. The suggestion to use a more flexible model instead of a decision tree for case specific attribute selection is not new. However, previous studies, see e.g. [21], [32], [64], and [26], have not addressed the issue of performance, i.e. they did not investigate whether the selected attributes were really relevant for determining the class of a novel example.

We will adopt probability theory as our general theoretical framework, and the different approaches we shall subsequently be considering correspond to different models for the probability distributions inherent in this framework.

In principle, methods such as neural networks [9] and support vector machines [72] can be used to provide flexible parametric functions with high classification performance. However, in CSAS we need to infer class labels based on arbitrary numbers of measured attribute values. This can not be done with a standard feed-forward model because it needs all the input values to determine the class label. In principle, this problem can be solved by using a separate input density model to fill in the missing values. This approach is, however, computationally very demanding. For practical applications it is important that the method remains computationally tractable and for this reason we restrict ourselves to the simple yet powerful class of
mixture models [68]. We will use mixture models to estimate the joint distribution of class labels and attribute values. Previous studies, [37] and [67] have shown that, in addition to providing a fit to the input distribution, joint mixture models give a good classification performance. In these studies, standard EM ([16]) was used to train the models. In this chapter we shall use the slightly more sophisticated Deterministic Annealing algorithm [58]. This algorithm avoids many of the difficulties of the standard EM approach, like getting stuck in bad local minima and the problem of infinite likelihoods in density estimation.

In our treatment of case specific attribute selection we make two simplifying assumptions. First, we take a myopic approach [8], i.e. at each stage the single best attribute to measure is determined, regardless of the allowable number of future measurements. It is conceivable that in some practical applications a non-myopic approach, in which attributes are selected group-wise, might outperform a method based on the myopic approach. The extra computational effort, however, is the same for both mixture models and decision trees. There is no other reason to suspect that the extension to non-myopic selection will be more beneficial to one algorithm than to the other. Second, we assume that the measurement costs are the same for all attributes and for all misclassified classes. We will show that the costs of measurements and misclassifications can be easily incorporated in our framework. Under these simplifying assumptions, we choose our utility function for observing a particular attribute to be Shannon’s expected information [63]. Application of this function in the construction of decision trees is equivalent to applying the information gain measure of the C4.5 algorithm [55].

The outline of the chapter is as follows: The information theoretic framework of case specific attribute selection will be presented in section 5.2. In this section we shall derive the attribute selection criterion. Section 5.3 presents three classes of models with which the probabilities in the selection criterion are estimated, namely 1) Mixture models found by deterministic annealing, 2) Naïve Bayes which serves as a default method, and 3) decision trees. The performances of the different approaches are compared on several real-world data-sets in Section 5.4. Finally, section section 5.5 presents a discussion of the results and a summary of the main conclusions. Here we shall also discuss other work that is relevant in the context of CSAS.

### 5.2 Case specific attribute selection based on information

A well founded measure of uncertainty or information associated with a sample space \( k = 1, \cdots, n_k \) of a discrete class variable given some measured attribute values \( x_d \) is given by [63]

\[
S(x_d) \equiv - \sum_{k=1}^{n_k} p(k|x_d) \log p(k|x_d),
\]  

(1)
where \( p(k|x_j) \) is the conditional probability of class \( k \) given \( x_j \). In physics, the measure \( S \) defined by (1) is called the (conditional) entropy of the distribution \( p(k|x_j) \). In this chapter, the more appropriate term ‘information’ is used to refer to \( S \).

The information \( S \) is a suitable measure to determine the relevance (information gain) of observing the value \( x_j \) of attribute \( i \). In order to evaluate the relevance of an attribute \( i \) prior to its observation we need to integrate all its possible values \( x_j \) weighted with their probability of occurrence. In other words, an unobserved attribute \( i \) is ranked according to the expected value of the information

\[
\left< S(x_j, x_d) \right>_d = -\int_{x_j} p(x_j | x_d) \sum_k p(k | x_j, x_d) \log p(k | x_j, x_d) dx_j.
\]

The most informative variable \( v_p(x_d) \) with respect to the probability distribution \( p \) and depending on the knowledge \( x_d \) is then

\[
v_p(x_d) = \arg\min_i \left< S(x_j, x_d) \right>_d | x_d.
\]

In our myopic approach, we then find the value of the attribute \( X_{vp} \). This consequently increases our observed attributes \( x_d \leftarrow (x_d, x_v(x_d)) \) and the process can be repeated, selecting the next most relevant attribute given the observed attributes. This process can go on until sufficient information has been gathered to make a reliable classification.

In this chapter we have chosen to use the expected information (2) for CSAS. Depending on the application, other criteria may be more appropriate. Two alternatives that include misclassification costs and measurement costs are presented in appendix 5.A. Implementation of one of these alternatives does not greatly increase the computational complexity of CSAS algorithms compared with the information measure.

### 5.3 Estimating probabilities for CSAS

Note that for CSAS we need to estimate two types of probabilities, namely 1. ‘feed-forward’ probabilities \( p(k|x_j, x_d) \) to predict class labels based on partial information and 2. ‘input’ probabilities \( p(x_j|x_d) \) to predict outcomes of measurements. We shall compare three approaches to estimate these probabilities. The first approach is to use a parametric mixture model with which we estimate a joint probability distribution

\[1\]The notation \( \langle f(x) \rangle_x \) is used for the expectation value of a function \( f \) over the distribution \( p(x) \) i.e., \( \langle f(x) \rangle_x = \int f(x) p(x) dx \). If the distribution is conditioned on \( x_d \), we use the notation \( \langle f(x) \rangle_{d | x_d} = \int f(x) p(d | x_d) p(x_d) dx \).
Chapter 5

91

$p(k, x)$ (section 5.3.1). A simpler, default, approach to estimate probabilities is to fit a Gaussian distribution for each class. This approach is referred to as ‘naïve Bayes’ and is briefly described in section 5.3.2. A very different approach is to use a decision tree generating method. This approach is described in section section 5.3.3.

5.3.1 Mixture models

A good balance between predictive performance and computational complexity is provided by the class of mixture models [68]. A mixture model is a weighted summation of component distributions which, for a vector of observations $x = (x_1, \ldots, x_n)$ has the form

$$p(x) = \sum_{h=1}^{M} p(h)p(x|h).$$

(4)

In this chapter we will restrict ourselves to numeric valued attributes i.e., $x \in \mathbb{R}$. The mixture components are modeled with Gaussian distributions. The class labels are discrete and their distribution $p(k|h)$ will be modeled with a full probability table for each mixture component $h$. By choosing these parameterizations the joint probability $p(k, x)$ of a class label $k$ and a vector of attribute values $x$ can be written as

$$p(k, x) = \sum_{h=1}^{M} p(h)p(k|h)2\pi\Sigma_k^{-1} \exp \left\{ -\frac{1}{2} (x - \mu_h)^T \Sigma_k^{-1} (x - \mu_h) \right\}$$

(5)

where $\mu_h$ and $\Sigma_h$ are the mean and the covariance matrix of mixture $h$, respectively.

The mixture weights $p(h)$ are modeled with a full probability table.

We shall make the simplifying assumption that the covariance matrices are diagonal i.e., $(\Sigma_h)_{ij} = \sigma^2 \delta_{ij}$. Under this assumption, the individual mixtures are factorized:

$$p(x|h) = \prod_i p(x_i|h) = \prod_i \exp \left\{ -\frac{1}{2} \sigma^2 (x_i - \mu_i)^2 \right\}.$$  

In principle, there is no difficulty in using general correlated Gaussians as richer models. However, the number of parameters, generally roughly $n^2/2$ for a full Gaussian, needs to be limited to avoid over-fitting. In addition, with diagonal covariance matrices we can still model any dependency between any subset of attributes by including more components. In this way, there is only one regularization parameter that we need to tune for optimal generalization, namely the number of components $M$. When using a Gaussian mixture model, there is in principle no restriction on the class of decision functions that can be described. To give an idea, Figure 5.1 shows a possible situation of how decisions could depend on the values of the attributes.

To use mixture models in CSAS, we need to compute marginal probabilities and evaluate integrals to compute expectation values. Because each mixture component is modeled with a product distribution, marginalization is computationally easy: One can just eliminate the variables one is not interested in. For example, suppose we
Case Specific Attribute Selection

Figure 5.1: Using a mixture model, the decision boundaries can have any shape. In this example, the numbers are indices of the most informative attributes within each region.

a have mixture model for two attributes $X_1$ and $X_2$, and a class $k$. Integrating out variable $X_2$ gives

$$p(k, x_1) = \int \sum_{h=1}^{M} p(h)p(k|h)p(x_1|h)p(x_2|h)dx_2 = \sum_{h=1}^{M} p(h)p(k|h)p(x_1|h)$$

Therefore, the time to compute a marginal probability is linear in the number of mixtures $M$ and independent of the number of marginalized variables. This makes it possible to quickly compute (2), so that we can compare the expected relevance of many attributes in many different situations in a short time. In order to determine the expectation values of the information (2) we need to compute integrals of a function containing the logarithm of a sum of Gaussians. These can not be evaluated analytically. We therefore use Gaussian quadrature to compute these integrals numerically (see appendix 5.B).

Depending on the objective, there are different ways to find the parameters $\mu$, $\sigma$, $p(k|h)$ and $p(h)$ of a mixture model. In section 5.3.1 we will describe a procedure which finds parameters that are suitable for CSAS.

**Deterministic Annealing**

The usual approach to fit a mixture model to a dataset is to maximize the likelihood of the mixture model (5) with respect to its parameters. In this approach the complexity or the flexibility of the model is controlled by the number of mixtures $M$.

The optimization scheme that we will employ here is similar to the usual approach but instead of using $M$ as a regularization parameter we will use the variance $\sigma^2$ to control the ‘effective’ number of mixtures in the system. At each value of $\sigma$ the log likelihood $L = \sum_{v \in D} \log p(k^{v}, x^{v})$ on a training set $D$ is maximized with respect to
Figure 5.2: One dimensional example showing the evolution of the means $\mu_h$ of the mixture components $h$ as a function of the number $t$ of steps with $\sigma = \alpha \sigma_{t-1}, 0 < \alpha < 1$. The data (which are plotted at the right) are generated from four Gaussian distributions centered at 0, 8, 16, and 24 respectively with unit variance. Each data point has a class label “O” or “X”.

the parameters $\mu_h, p(k|h)$ and $p(h)$ using the EM-algorithm [16]. In deterministic annealing we start with a large value $\sigma_0$, and decrease it in small steps by $\sigma_i = \alpha \sigma_{i-1}$ with $1 > \alpha > 0$. A more detailed description of this algorithm can be found in [58].

To illustrate the deterministic annealing algorithm we sampled data from two classes distributed over four well separated one-dimensional Gaussian clusters. We assigned class labels ‘x’ or ‘o’ to the data in each of the clusters. The evolution of the means $\mu_h$ of the mixture components on these data as a function of the number of steps $t$ of the variance $\sigma^2$ is shown in Figure 5.2. The data with labels ‘x’ or ‘o’ are plotted along a vertical line at the right side of the figure. At large variances (small $t$) one mixture component is located at the center of mass $\mu = 12$ of the whole dataset. After $t = 21$ iterations there is spontaneous symmetry breaking and an extra component needs to be included. After further annealing each component moves to the center of mass of one of the classes. Decreasing $\sigma^2$ (increasing $t$), more bifurcations occur resulting in a more detailed mixture model with more mixture components.

The computation time of the annealing procedure on a dataset is proportional to the number of attributes $n$, the number of patterns $P$ in the training set, and the number $n_T$ of annealing steps that are taken. As an example: With our Matlab implementation, optimization (6 mixture components and 30 annealing steps) of a model on a data set with 13 attributes and 142 training patterns takes about 6 seconds on a 1GHz Pentium.
Case Specific Attribute Selection

Selecting the optimal value of $\sigma$
In order to optimally perform the task of CSAS it is crucial to select the right value for the noise parameter $\sigma$. We select $\sigma$ based on the performance of the mixture model as a classifier. In other words, we shall select $\sigma$ based on the accuracy of the estimates $p(k|x)$, where $x$ includes all attribute values\(^2\).

The selection is done by cross-validation. The resulting value of $\sigma = \sigma_{\text{opt}}$ is the value that gives (on average) maximal classification performance, measured by the values $\sum_{\mu \in \{\text{Val}\}} \log p(k|\mu|x^d)$, on a set of disjoint validation sets $\{\text{Val}\}_1, \ldots, \{\text{Val}\}_{n_{\text{res}}}$.

5.3.2 Naive Bayes
In Naive Bayes it is assumed that the data corresponding to each class $k$ are distributed according to a class specific normal distribution. Under this assumption, the joint probability distribution of class labels $k$ and and inputs $x$ is,

$$p(k, x) = p(k)p(x|k),\quad (6)$$

where

$$p(x|k) \propto \exp \left\{ -\frac{1}{2} (x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k) \right\}, \quad (7)$$

in which $\mu_k$ and $\Sigma_k$ are estimates of the mean and covariance of the data from class $k$.

The application of the model (6) for CSAS is similar to applying mixture models. The marginals $p(x_j|x_d)$ of (6) are also Gaussian and can be constructed by selecting sub-matrices from $\Sigma_k^{-1}$. Since in Naive Bayes the distribution $p(x|x_d) = \Sigma_k (p(k)p(x_j|x_d, k)$ is a mixture over classes, the expectation values (2) have to be evaluated numerically as in the previous section.

In contrast to the mixture model approach of section 5.3.1 the flexibility of a naive Bayes model can not be optimized for a specific data set: the number of Gaussian components is equal to the number of classes.

5.3.3 Decision trees
For CSAS we need estimates for the probabilities $p(k|x_d, x_d)$ and $p(x|x_d)$. In this section we show how these probabilities are modeled if we use a decision tree. More extensive descriptions of decision tree algorithms can be found in [10, 55].

Roughly speaking, the construction of a decision tree consists of recursively splitting a region of input space in two parts where each split separates data with different class labels as well as possible.

\(^2\)Choosing $\sigma$ based on the classification accuracy given incomplete information is clearly infeasible. Since we do not know a-priori which information will be important and which information will not be important.
The attribute space corresponds to the \( n \)-dimensional real space \( \mathbb{R}^n \). The first step in constructing a decision tree is to determine the optimal split which separates the labeled data as well as possible. Since we restrict ourselves to the myopic approach, there is only one variable involved in each split (i.e., we do not consider splits with an oblique orientation or splits which are curved). In that case, if we look at attribute \( X_i \), the split can be fixed by a single parameter \( \theta \) which splits the attribute space \( \mathbb{R}^n \) in two parts \( R_{x_i \leq \theta} \) and \( R_{x_i > \theta} \). In each of these parts there is a set of data points of which a fraction \( f_{R_{x_i \leq \theta}} \) or \( f_{R_{x_i > \theta}} \) has a class label \( k \). Together these fractions define a conditional class distribution,

\[
p_\theta(k|x_i) = \begin{cases} f(k|R_{x_i \leq \theta}) & x_i \leq \theta, \\ f(k|R_{x_i > \theta}) & x_i > \theta. \end{cases}
\]  

(8)

In a similar way, we can construct a probability density distribution \( p_\theta(x_i) \) for the attribute values \( x_i \),

\[
p(x_i) = \begin{cases} \frac{1}{x_i^{\text{min}} - \theta} f(R_{x_i \leq \theta}) & x_i^{\text{min}} \leq x_i \leq \theta, \\ \frac{1}{x_i^{\text{max}} - \theta} f(R_{x_i > \theta}) & \theta < x_i \leq x_i^{\text{max}}, \end{cases}
\]  

(9)

where \( x_i^{\text{min}} (x_i^{\text{max}}) \) is the minimum (maximum) value for attribute \( X_i \) found in the data set. Furthermore, \( f(R_{x_i \leq \theta}) \) and \( f(R_{x_i > \theta}) \) are the fractions of data points for which \( x_i \leq \theta \) and \( x_i > \theta \), respectively.

After substitution of the probabilities (8) and (9) into (2) the expected information \( \langle S \rangle \) still depends on the split parameter \( \theta \). To find the first split in the tree we minimize the expected information with respect to \( \theta \) for each attribute \( x_i \),

\[
\theta_i = \min_{\theta} \langle S_\theta(x_i) \rangle_{x_i}
\]  

(10)

and choose the attribute which gives the smallest value for \( \langle S \rangle \), i.e., we choose

\[
v = \arg \min_{i} \langle S_\theta(x_i) \rangle_{x_i}.
\]  

(11)

After having selected attribute \( v \), the resulting two regions \( R_{x_v \leq \theta_v} \) and \( R_{x_v > \theta_v} \) can each be partitioned again using the same procedure. This second partitioning is then conditioned on the value \( x_v \). Hence, in that case the probabilities in (8) and (9) are changed into \( p(v|x_v) \) and \( p(k|x_v,x_i) \). This process can be recursively repeated until each region contains data of one class only or until all attributes have been selected.

As explained above, both mixture models and decision trees are suitable model classes to use for CSAS. One of the important differences between the approaches is the way in which they choose a new attribute. To get an intuitive understanding of this difference consider a hypothetical classification problem with 6 input attributes.
Suppose that we wish to classify an example and that we already know the values $x_1$ and $x_2$. Depending on the values a decision will be made about which attribute to measure next. Because the decision tree consists of a set of axis parallel splits, the decisions depend on the values $x_1$ and $x_2$ as shown in Figure 5.3. In contrast, no such restriction holds for the decisions if we use a more flexible probability model like a mixture model, see Figure 5.1.

### Pruning and Model Selection

A crucial aspect of decision tree induction is to control the complexity or the level of detail in order to avoid over- or under-fitting. There are many techniques to do this, see for example [12], [10], and [42]. In our implementation we use the pruning criterion of C4.5. Here the observed distribution of errors is assumed to be a sample from a binomial distribution. Under this assumption one can estimate an error distribution. This estimate depends on a confidence factor $CF$. If the expected error of the children of a node is larger than that of the node itself, the children will be removed. An advantage of this approach is that the amount of pruning is controlled by a single parameter $CF$. If it is taken small, a small sample will be penalized harder so that the tree will be pruned further compared with larger values of $CF$. Hence, depending on the data set we can determine the optimal pruning level. In our implementation we select the value of $CF$ by cross-validation in the same manner as we select the regularization parameter $\sigma^2$ in the mixture models.

### 5.4 Experimental Results

In this section we will compare our probability model method with the decision tree approach. In order to obtain a clear understanding of the differences between the performance of the mixture model and the decision tree we first consider two toy
problems involving two dimensional input spaces. Subsequently, we compare the methods on some public domain databases.

Since a good classification performance is essential for CSAS, we begin with a comparison of classification performances using all the input attributes. Next, we compare the CSAS performances.

### 5.4.1 Comparison of the methods on UCI and Statlog databases

We compared the performance of mixture models with that of decision trees on a selection of databases from the UCI [45] and Statlog repositories [43] which have been commonly used in the literature.

#### Basic Classification using all attributes

Before investigating the CSAS performances on these databases we compared the classification performances using all the input attributes. From each database we randomly selected a training set and an independent test set. After training we determined the fraction of correctly predicted classes of the test set patterns. For each database, this procedure was repeated 5 times. The results shown in the second, third and fourth column of table(5.1) are the sample means and the sample standard deviations of these repeated experiments. Assuming that the results are independently drawn from a normal distribution\(^3\) we applied a significance test for the differences between the decision tree and mixture model scores. For this purpose we used a one-sided \(t\)-test which gives an upper bound \(\alpha\) to the probability of the null-hypothesis that the samples from the mixture models were drawn from a distribution with a mean which is not higher than the mean of the decision tree scores. The mixture model scores can be considered to be significantly higher if \(\alpha < 0.05\). Hence, from table(5.1) we see that the mixture model scores obtained are either significantly higher than the decision tree scores or are indistinguishable, but never significantly smaller.

In all the experiments, Naïve Bayes has an inferior performance and will therefore be left out in the following experiments.

#### CSAS results

To determine the CSAS performance on a given data base we trained a model (a mixture model or a decision tree) on a randomly sampled training set \(D_{\text{train}}\), where we used the same number of training patterns as in the classification experiments (table(5.1)). Next we performed CSAS on each pattern \((k^{\text{est}}, x^{\text{est}})\) in an independent

\(^3\)Which is reasonable for this purpose but not exactly correct: The distribution of the scores becomes more asymmetric if the score is close to 1. Furthermore, the samples are not completely independent since they were drawn from the same database resulting in an overlap between the samples which somewhat reduces the sample standard deviations.
Table 5.1: Classification results for Naïve Bayes, C4.5 and the mixture model on public domain databases using all attributes as input. The scores presented in the table are the fractions of patterns in a test set for which a method predicted the correct class. \( P \) is the number of patterns in the data set and \( n \) is the number of input attributes. \( \langle M \rangle \) is the average optimal number of mixture model components. \( \alpha \) is an upper bound to the probability of the null-hypothesis that the samples from the mixture models were drawn from a distribution with a mean which is not higher than the mean of the decision tree scores.
Figure 5.4: CSAS results for four real world databases. \( n_s \) is the number of selected attributes. Decision trees (dotted line and ‘o’). Mixture models (solid line and ‘x’). The vertical bars correspond to the sample standard deviations.

When considering a novel test pattern the algorithm at first does not have any knowledge about the pattern, i.e. the vector of measurements \( \mathbf{x}_d \) is empty, and it will select the first most informative variable \( i_1 \). The value of this variable \( x_d \leftarrow x_{d_1}^{est} \) is then obtained from the test example. Based on this value we determine the class prediction by \( \arg\max_k p(k|x_d) \) which is compared with the true class \( k_{test} \) of the pattern. The procedure continues until all the \( n \) attribute values of the test pattern have been read. The end result is a sequence \( score(1), \ldots, score(n) \) containing the test set scores as a function of the number of measurements.

Figures 5.4–5.6 show the classification results as a function of the number of measurements, where ‘fraction correct’ = \( score(n_s)/n_{test} \). As can be seen from these figures the mixture models in general have a higher classification score for each number of selected attributes \( n_s \). As expected, the score initially increases gradually if more measurements are made. At some point the score ceases to increase indicating that additional measurements do not provide any relevant information. On some
Figure 5.5: CSAS results for two real world databases. $n_s$ is the number of selected attributes. Decision trees (dotted line and ‘o’). Mixture models (solid line and ‘x’). The vertical bars correspond to the sample standard deviations.

of the problems, e.g. the Hepatitis and German data bases, the decision tree (even though the proper amount of pruning was determined with an extensive cross validation procedure) tends to over-fit the data resulting in a decreasing test set score as a function of the number of measurements.

In Figure 5.6 we show the results obtained on the Diabetes, Credit, and Australian databases. These problems have in common that the classification score does not increase after more than one measurement. This suggests that there is only one informative attribute involved in these problems (as far as we can see from the databases). This is supported by the observation that the classification scores obtained after one measurement are comparable to the scores (in general using all attributes as inputs) reported elsewhere, see [43].
Figure 5.6: CSAS results for four real world databases. $n_s$ is the number of selected attributes. Decision trees (dotted line and ‘o’). Mixture models (solid line and ‘x’). The vertical bars correspond to the sample standard deviations.
5.5 Discussion

In this chapter we have addressed the issue of case specific attribute selection. That is, given an example with missing attributes, to determine which single extra attribute will most likely aid the correct classification of the example. Instances of applications in which a solution to this problem is important are, among others, machine fault diagnosis, the construction of client specific insurance policies, and medical diagnosis.

In constructing our joint probability model, we chose a mixture distribution. In order to test its case specific attribute selection performance we compared it to the more standard technique of decision trees. Experiments on a variety of toy and real world problems indicate that the mixture model has a classification performance which is better or at least equal to that of a decision tree. More importantly, the experiments showed that an enhanced performance is also obtained if the models are compared for case specific attribute selection where the models have to select attributes and have to classify novel examples using a small subset of selected attributes.

In the following subsection we shall reflect on the use of the deterministic annealing algorithm for CSAS. Section 5.5.2 discusses the topic of missing values and ‘Lazy Decision Trees’ in the context of CSAS. Finally, section 5.5.3 discusses possible improvements and connections with other work on decision trees.

5.5.1 Deterministic Annealing and CSAS

In training our mixture model, we found that the parameter optimisation method was critical in determining the success of the classifier. The deterministic annealing procedure we used, which couples the input distribution to a complexity parameter $\sigma$ and which uses a non-constrained distribution for the classes, results in a joint probability distribution which proved accurate if used for case specific attribute selection. We tested several other optimisation schemes which all produced mixture models with an inferior CSAS performance. In particular, we obtained poor performance if the mixture model is trained in the ‘usual’ way, in which all parameters (including the variances $\sigma^2$ of the input variables) in the model are optimised with the EM-algorithm and where the complexity of the model is controlled by changing the number of mixture components. In our view, the inferior performance obtained by fitting all model parameters (including the variances $\sigma^2$) simultaneously is due to a fit with equal emphasis on both the probability density of the input space and that of the conditional class probabilities. By constraining the variances $\sigma^2$ for all mixture components in the deterministic annealing procedure the emphasis is on fitting the conditional class distribution with a smaller emphasis in the input (attribute) probability distribution. Hence, for case specific attribute selection, where we need a model of the joint distribution but where we are not primarily interested in a high likelihood of this model, the deterministic annealing procedure with an unconstrained class distribution seems to be a suitable approach.
The complexity of the mixture model was determined by considering the classification score using all the input variables during the cross-validation procedure. The optimal value for the complexity $\sigma$ was chosen as the value for $\sigma$ which gave the best classification performance. When, however, a mixture model is selected on a data set with a large number of attributes, it is not a priori clear whether the resulting complexity will also be optimal when the model is used for computing the class predictions using a small subset of the attributes. Interestingly, preliminary experiments have shown that the optimal complexity does not depend to a large extent on the number of specified attribute values.

### 5.5.2 Dealing with missing values

In terms of computational complexity, inducing a mixture model from a given data set is computationally more intensive than inducing a decision tree (a factor 10 to 100 in terms of CPU time). However, after the training phase, a mixture model is more flexible in dealing with missing attribute values and with knowledge obtained prior to the selection process since it has stored all the joint probabilities. In the standard approach a decision tree is fixed after training. If one already has some knowledge $x_d$ then the optimal sequence of measurements may no longer correspond to a path in the tree. Ideally, a new decision tree should be induced in which there is a path starting from the root node which contains the measurements $x_d$. The node succeeding this path will then indicate the most relevant attribute. A similar situation occurs if for some reason it is not possible to measure the value of an attribute at a certain point in the tree. In both these cases a new (sub) tree would have to be trained, pruned and cross-validated which is computationally burdensome especially if one has a large data set. Using a mixture model on the other hand, there is no need to refer to the data again after the training phase, regardless of the information contained in a novel, incomplete case.

At this point it is interesting to consider the work of Friedman and Kohavi [20] on Lazy Decision Trees. In standard decision trees splits are chosen based on their average performance on the training data. These splits may not be optimal for a specific test instance. For this reason lazy decision trees construct an optimal decision path specifically for the test instance. The algorithm is just as flexible regarding missing attribute values as the mixture model approach: one can just skip tests on attributes with unknown values. However, Lazy Decision Trees can not be used for Case Specific Attribute Selection. The main strength of the algorithm is that it uses the fact that all attribute values of the test instance are already known. With this information one does not have to integrate over possible test outcomes and one is better able to select the optimal test. Of course, the main issue of CSAS is that we do not know the test outcome in advance.
5.5.3 Alternative approaches and possible improvements

An alternative to the mixture model approach presented here would be to estimate two distributions - a feed-forward model (e.g. a Multi-Layer Perceptron) and a separate input density model (e.g. a Gaussian mixture model). In a feed-forward model we need to specify all input variables in order to obtain an output. For case specific attribute selection, however, we need to be able to infer the class probability conditioned on a limited set of input attributes. The separate input density model could then be used to ‘fill in’ the values of the missing attributes which are then fed into the feed-forward model. In this alternative approach, one would need to integrate the resulting feed-forward class prediction over all possible values of the unknown input attributes. In principle these integrations can be approximated using Monte-Carlo methods. For the scenarios we have in mind, however, this approach is intractable, even for small problems. In contrast, in the approach outlined in this chapter this integration is trivial and corresponds to simply omitting the terms associated with the unknown attributes.

On the side of the decision tree algorithms there are a number of interesting alternatives and refinements which may be relevant for CSAS. First, in this chapter we have considered binary splits. Possibly, the number of measurements can be reduced if we allow multiway splits where the range of attribute values is divided in \( m \geq 2 \) intervals for each of which subtree is constructed. The use of multiway splits is, however, computationally very demanding: the search for an optimal \( m \)-way split using \( P \) training examples has a computational complexity \( \mathcal{O}(P^m) \), which is infeasible for large data sets and large \( m \). Furthermore, it is difficult to determine the optimal value for \( m \). For this one has to use a local optimization criterion like the ‘gain-ratio’ criterion [55] or a ‘Minimum Description Length’ [57] criterion, which may not be optimal for the tree as a whole. In this context it might be interesting to consider faster approximate methods for finding multiway splits, see for example [19].

Other improvements might be found by constructing ‘hybrid’ models, which combine the properties of decision trees with those of other models. For example, Kohavi [34] combines decision trees with naïve Bayes classifiers, the tree the class distributions in a decision tree node are estimated with a naïve Bayes model instead of with empirical distributions. On some data-sets this approach gave a better classification performance compared with stand-alone versions of C4.5 and naïve Bayes. Another hybrid approach that is interesting for CSAS is to extract a decision tree from a neural network instead of directly from the data, see Craven [14]. As in our mixture model approach one then first fits a flexible interpolating function to the data

\[ p(k|x_{\text{known}}) = \int p_{\text{ff}}(\hat{k}|x_{\text{known}};x_{\text{unknown}})p_{\text{input}}(x_{\text{unknown}}|x_{\text{known}})dx_{\text{unknown}}, \]

where \( p_{\text{ff}} \) and \( p_{\text{input}} \) indicate the feed-forward model and the input model, respectively.
and afterwards selects attributes with this function. With this approach Craven found smaller and better generalizing decision trees.

Many recent techniques for improving the classification performance of decision trees are based on constructing ensembles, where the prediction is made by a voting or averaging scheme over the outcomes of different induced trees. Examples of these methods are boosting methods [54], [65] and Bayesian methods [12]. Clearly, these methods are not directly suitable for CSAS where only a single decision can be made at each step.

Acknowledgements

We wish to thank David Barber for helpful discussions. This research is supported by the Technology Foundation STW, applied science division of NWO and the technology programme of the Ministry of Economic Affairs.

Appendix 5.A Alternative Selection Criteria

In section 5.2 we proposed to use the information or entropy of the class distribution as a selection criterion. In some applications of case specific feature selection other criteria might be more appropriate. For example, in the medical domain, it could be important to also include the costs of a test. A simple solution is to weigh the relevance of an attribute with its cost of measurement $C_i$, i.e., as an alternative of (3) we would then use

$$v_p(x_d) = \arg\min_i \left\{ C_i \langle S(x_i, x_d) \rangle_{x_i | x_d} \right\}.$$  \hspace{1cm} (12)

In other situations, it is import to minimize the chance of finding ‘false-positive’ examples. Again, in medical diagnosis it is more dangerous to classify ‘benign’ when it should be ‘malign’ than the other way around. Instead of using information as a criterion, we could then select attributes based on the expected misclassification costs,

$$\langle K(x_i, x_d) \rangle_{x_i | x_d} \equiv \int p(x_i | x_d) \left\{ \sum_k p(k | x_i, x_d) \sum_{l \neq k} p(l | x_i, x_d) K(l, k) \right\} dx_i,$$  \hspace{1cm} (13)

where $K(l, k)$ is the cost of mis-classifying a pattern of class $k$ as class $l$. If the number of classes is not extremely large, the computational complexity of using (13) is not much bigger then using (2). Other similar cost sensitive test selection criteria have been proposed in [65], and in [69].
Appendix 5.B Calculation of the expected information from a mixture model

To obtain the expected most informative variable based upon the knowledge $x_d$ we need to evaluate the integrals (2). Because there is a summation over the mixture components, see (5), inside the logarithm there is no analytical solution for this integral so we have to make a numerical approximation.

From (5) we see that $p(x_i|x_d)$ is a Gaussian mixture distribution for variable $i$. The integral (2) can therefore be written as a sum of integrals each containing one Gaussian kernel. Each of these integrals can be approximated accurately and efficiently with Gaussian quadrature integration (see for example [53]), i.e.

$$\int e^{-z^2} f(z) dz = \sum_{j=1}^{N} w_j f(z_j),$$

(14)

where $w_j$ and $z_j$ are Gauss-Hermite weights and abscissas respectively, which can be found in mathematical tables. With $N = 16$ the approximation is in general already very accurate (by comparison with the slow but accurate Runge-Kutta method).
Publications

International refereed journals


Conference Proceedings


Technical Reports


Bibliography


Bibliography


[53] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flan-
[60] L. Rabiner. A tutorial on hidden markov models and selected applications in
[62] J. Rissanen. A universal prior for integers and estimation by minimum descrip-
[64] K. Rose, E. Gurewitz, and G.C. Fox. Vector quantization by deterministic an-
[66] B. Sallans, G.E. Hinton, and Z. Ghahramani. A hierchical community of ex-
[67] perts. In C. M. Bishop, editor, Neural Networks and Machine Learning, NATO
[71] Eric Saund. A multiple cause mixture model for unsupervised learning. Neural
[74] relates of emotion dimensions in view of speech synthesis. In Eurospeech 2001,
[76] C. E. Shannon. A mathematical theory of communication. part III. Bell System
[78] M. Stensmo and T.J. Sejnowski. A mixture model system for medical and ma-
[80] Processing Systems, volume 5, pages 1077–1085. Morgan Kaufmann, San Ma-
[82] Kai Ming Ting and Zijian Zheng. Boosting trees for cost-sensitive classifica-
[85] Report NCRG/97/010, Neural Computing Research Group, Aston University,


Inleiding en Samenvatting

Tegelijk met het toenemen van het gebruik van computer systemen in onze samenleving, worden er dagelijks steeds grotere hoeveelheden data verzameld en opgeslagen. Of je nu bij de bank bent, in de supermarkt, aan het surfen bent op het internet of een telefoongesprek aan het voeren bent, iedere interactie wordt geregistreerd en opgeslagen. Tegelijkertijd registreren digitale sensoren de hartslag van patiënten in ziekenhuizen, de status van druktanks in chemische fabrieken en de activiteiten aan het loket bij een benzinestation. Terwijl er steeds meer data beschikbaar komen worden de middelen om die data te verwerken en te analyseren steeds goedkoper. Tegenwoordig kan zelfs een eenvoudige PC miljarden data bits per seconde verwerken. Beide ontwikkelingen, het beschikbaar komen van data en het goedkoper worden van rekenkracht, creëren een geweldige potentie voor technieken waarmee patronen en relaties in de data onthuld kunnen worden. De gevonden kennis kan bijvoorbeeld gebruikt worden om het koopgedrag en het kredietrisico van een consument te voorspellen, om radarbeelden te classificeren of om een handtekening te herkennen.

Methoden waarmee automatisch patronen en relaties in data verzamelingen gevonden kunnen worden vormen het onderwerp van dit proefschrift. Het proefschrift presenteert een viertal nieuwe data analyse technieken. Iedere techniek wordt experimenteel vergeleken met alternatieve reeds bekende technieken. De vergelijkingen worden gedaan op data verzamelingen die uit echte situaties zijn voortgekomen. De doelstellingen bij het ontwikkelen van iedere techniek kunnen als volgt worden samengevat: 1) het vinden van compacte beschrijvingen of modellen van gecompliceerde data verzamelingen en 2) het gebruiken van deze modellen om relaties of functies te vormen waarmee nauwkeurig voorspellingen gedaan kunnen worden. Een extra doel bij de ontwikkeling van de technieken is dat ze modellen genereren die de data inzichtelijk maken voor gebruikers. De technieken zijn ontwikkeld om geneeck toepasbaar te zijn en zijn niet ontwikkeld met het oog op een specifieke toepassing. Ze zijn echter wel getest op specifieke toepassingen. Voorbeelden hiervan zijn: handgeschreven cijferherkenning, beeldcompressie, classificatie van satellietbeelden, emotieherkenning uit spraakdata, herkennen van bloemensoorten, wijnsoorten en bacteriesoorten, diagnose van hartziekten en diabetes, het classificeren van kredietrisico, etc.
 Een karakteristieke eigenschap van de modellen die in dit proefschrift worden bestudeerd is dat ze data representeren met behulp van afgeleide of *verborgen* variabelen. In veel praktische situaties bestaat de data verzameling uit waarden van vele variabelen of velden. Samen met het grote aantal variabelen maken de aanwezigheid van ruis, inconsistenties en ontbrekende waarden het moeilijk om relaties tussen data punten te vinden. Het is echter vaak het geval dat de schijnbaar ongestructureerde data verzameling uitgelegd of beschreven kan worden met een veel kleiner aantal vrijheidsgraden. Verborgen variabelen modellen worden gebruikt om de data efficiënter te representeren in termen van een klein aantal attributen: de verborgen variabelen.

Elk hoofdstuk in dit proefschrift beschrijft de theorie en de testresultaten van een nieuwe data analyse techniek. Hieronder wordt van ieder hoofdstuk een samenvatting gegeven.

**Hoofdstuk 2**

In hoofdstuk 2 wordt een model gepresenteerd, de ‘Generative Vector Quantiser’ (GVQ), dat in staat is om complexe data verzamelingen te representeren in termen van een klein aantal verborgen variabelen. Een belangrijke gemeenschappelijke eigenschap met het populaire Vector Quantisatie is dat GVQ de data representeert met een geheel aantal clusters. Het verschil is dat standaard vector quantisatie geen presentatie middels verborgen variabelen geeft. Andere technieken die representaties geven middels verborgen variabelen zoals Principale Componenten Analyse (PCA) en Factor Analyse kunnen daarentegen alleen lineaire structuren weergeven. Deze beperking geldt niet voor GVQ.

Net als PCA is GVQ deterministisch, waarmee we bedoelen dat een data punt gerepresenteerd wordt door één unieke toestand van de verborgen variabelen. In tegenstelling tot niet-deterministische (dwz. probabilistische) modellen maakt de deterministische aanpak de toepassing van een bijzonder type lerend algoritme zeer interessant. Dit algoritme betreft het combineren van het ‘Expectation-Maximisation’ (EM) algoritme met het ‘Belief Revision’ algoritme. In hoofdstuk 2 wordt dit algoritme uitgebreid vergeleken met andere methoden zoals de ‘mean-field’ en ‘belief propagation’ algoritmen. Uit de vergelijking blijkt dat het van de situatie afhangt welk algoritme het best functioneert.

Aan het eind van hoofdstuk 2 worden praktische toepassingen van GVQ gepresenteerd, namelijk handgeschreven cijferherkenning en beeldcompressie. Bij handgeschreven cijferherkenning word GVQ gebruikt om kenmerken of elementaire bouwstenen van de cijfers te ontdekken. Er wordt gedemonstreerd dat GVQ met een beperkt aantal kenmerken een grote variatie aan realistische cijfers kan reconstrueren. Voor beeldcompressie word GVQ vergeleken met standaard vector quantisatie. Het blijkt dat de GVQ methode een beeld kan comprimeren met veel minder verlies dan de standaard methode.
Hoofdstuk 3

Het onderwerp van hoofdstuk 3 is de toepassing van verborgen variabelen modellen voor classificatietaken. Verborgen variabelen modellen worden vaak gebruikt voor complexe classificatieproblemen, bijvoorbeeld voor spraakherkenning en het herkennen van eiwitstructuren. In de gebruikelijke aanpak wordt voor iedere klasse een apart model ontwikkeld middels de ‘maximum likelihood estimation’ techniek. Met deze techniek wordt voor iedere klasse een model gevonden dat een zo goed mogelijke representatie van de klasse geeft. Het algoritme dat in hoofdstuk 3 wordt gepresenteerd verbetert deze aanpak door gebruik te maken van randvoorwaarden in het leerproces (optimisatieproces). De randvoorwaarden eisen dat ieder model niet alleen een goede representatie van een klasse moet geven maar de modellen gezamenlijk ook optimaal onderscheid maken tussen de klassen. Twee varianten van deze methodologie worden onderzocht, namelijk 1) een aanpak die gebruik maakt van het Lagrange formalisme, het algoritme heeft in dat geval gelijkenis met het Support Vector algoritme en 2) een algoritme dat equivalent is met het Batch Perceptron Algoritme, ofwel het minimaliseren van de fout alleen op de verkeerd geclassificeerde patronen. Er wordt gedemonstreerd dat de eerste methode in minder stappen convergeert naar een optimale oplossing. Echter, bij iedere stap heeft het algoritme meer rekentijd nodig dan de tweede methode. Een probleem bij de tweede methode is dat er een leerparameter afgesteld moet worden om convergerend gedrag te krijgen. De eerste methode heeft deze afhankelijkheid niet. Uiteindelijk worden de methodes toegepast op een aantal realistische data verzamelingen en het blijkt dat de methodes een stuk nauwkeuriger zijn dan de standaard ‘maximum likelihood estimation’ methode.

Hoofdstuk 4

In hoofdstuk 4 wordt een methode gepresenteerd waarmee compacte classificatie-regels uit data verzamelingen gegenereerd kunnen worden. De GVQ methode uit hoofdstuk 2 wordt hierbij als uitgangspunt gebruikt. Het leeralgoritme dat gebruikt wordt om regels te vinden is een variant van het randvoorwaarde algoritme uit hoofdstuk 3. In dat hoofdstuk werd een apart model voor iedere klasse ontwikkeld. De methode van hoofdstuk 4 construeert één enkel model voor alle klassen. Het algoritme vindt hiermee een unieke symbolische representatie voor iedere zich onderscheidende groep in de data. Iedere waardecombinatie (symbolische representatie) van de verborgen variabelen in het model correspondeert tevens met één specifieke klasse. De symbolische representaties kunnen dus gebruikt worden om verschillende klassen te identificeren. De regelextractiemethode wordt gedemonstreerd op vier verschillende data verzamelingen, namelijk handgeschreven cijfer herkenning, herkenning van satelliet beelden, plantensoort herkenning en emotieherkenning uit spraak data. Op deze data verzamelingen wordt de methode vergeleken met 1) ’unsupervised’ leren waarbij de vorming van symbolen niet wordt beïnvloed door klasse
Inleiding en Samenvatting

Informatie en 2) met een andere standaard methode, namelijk ‘joint likelihood estimation’. De resultaten laten zien dat met de nieuwe methode veel nauwkeuriger classificatieregels gevonden worden dan met de standaard methodes.

Hoofdstuk 5

In veel toepassingen wil men objecten classificeren op basis van een minimum aantal attribuutwaarden. Een bekend voorbeeld hiervan komt men tegen in het medische domein. Voor het diagnostiseren van een ziekte is vaak een reeks van lab-testen nodig. In zo’n geval is het een voordeel als de arts een systeem ter beschikking heeft waarmee de testen voorspeld kunnen worden die het meest relevant zijn voor de onderzochte patiënt. Andere, gelijksoortige toepassingen zijn het bepalen van de oorzaken van een probleem in de werking van een machine of het bepalen van de optimale vragenreeks in een call center. Het probleem van het vinden van de meest relevante attributen in een specifiek geval zullen we hieronder aanduiden met het Engelstalige ‘Case Specific Attribute Selection (CSAS)’. In hoofdstuk 5 worden verschillende benaderingen voor het CSAS probleem onderzocht en vergeleken.

In verschillende studies [21, 32, 64, 26] is de praktische relevantie van CSAS reeds onderkend. De kwaliteit en de nauwkeurigheid van zulke technieken zijn echter nooit geëvalueerd of vergeleken. Het doel van hoofdstuk 5 is om twee fundamenteel verschillende benaderingen voor CSAS, namelijk beslisbomen [10, 55] en mixture modellen [68], te vergelijken. Beslisbomen vormen een voor de hand liggende methode voor CSAS, aangezien hun werking gebaseerd is op het sequentieel selecteren van attributen om de data optimal te verdelen in gebieden van verschillende klassen. Mixture modellen daarentegen modelleren de data met een kansdichtheid functie. Beide benaderingen worden vergeleken binnen het gemeenschappelijke raamwerk van de informatietheorie [63].

Om mixture modellen te optimaliseren voor CSAS hebben we ontdekt dat een variant van het ‘Deterministic Annealing’ algoritme [58] zeer goed werkt. Dit algoritme past het model zodanig aan dat het optimale detailniveau voor het beschrijven van een specifieke data verzameling wordt gevonden.

Beide methoden, beslisbomen en mixture modellen, worden uitgebreid vergeleken op een grote verzameling van realistische data sets. De resultaten laten zien dat mixture modellen gemiddeld beter werken voor CSAS dan beslisbomen.
Dankwoord

Dit proefschrift had ik nooit kunnen vervaardigen zonder de hulp van anderen. De namen van de auteurs boven de publicaties geven hier al een eerste indruk van. Laat ik dan ook met die auteurs beginnen. Eind 1995 werd de eerste stap gezet. Ik kwam vanuit Amsterdam op sollicitatiegesprek bij ‘hoogleraar Biofysica’ Stan Gielen. Het gesprek was wat formeel, hij sprak me aan met U, maar het klikte. Korte tijd later was ik aangenomen als OIO op een uitdagend project. Gedurende de volgende jaren is Stan een stimulerende factor geweest die me altijd weer een positief gevoel wist te geven, ook in de eerste moeizame ‘zwevende’ jaren. Ik bedank Stan voor het geven van de mogelijkheid om vijf jaar mijn gang te gaan binnen een inspirerend vakgebied. De moeizame jaren werden doorbroken met de komst van David, ‘the bright-young-lad from the UK’. Vanaf dat moment begon het echt leuk te worden. Hij wist mijn werkomgeving om te vormen tot een inspirerende en creativiteit stimulerende plek waarin we dagelijks urenlang brainstormden over de meest uiteenlopende onderwerpen binnen en buiten het vakgebied. Hieruit borrelden zeer goede ideeën naar boven (soms zo goed dat anderen ze helaas net iets eerder bedacht hadden) die de basis zijn gaan vormen voor het grootste deel van dit proefschrift. Daarnaast is David een zeer goede vriend geworden die ik nog lang zal blijven zien. Een ander vooraanstaand lid van het brainstorm-team was mijn co-promotor Wim. Wim fungeerde als een onmisbare rechterhand die mijn, soms wat rommelige, ideeën van de zwakke kanten wist te ontdoen en tot werkbare methoden wist om te smeden. Daarnaast was Wim altijd een stabiele factor die rustig mijn frustraties aanhoorde en naast me stond in moeilijke tijden (dwz. als we er na maanden zwoegen achter kwamen dat het werk al eerder was gedaan). Ik hoop dat we elkaar nog vaak zullen zien. Voor de dagelijkse lol afgewisseld met wetenschappelijk gefilosofeer bedank ik ‘nabir abi’ Taylan. Ik hoop dat we samen een keer naar Turkije zullen gaan. Tom Heskes bedank ik voor zijn, soms essentiële, kritische bijdragen en voor zijn talent om de gevoelige zenuw in een idee bloot te leggen.

Een belangrijke voorwaarde voor succes werd gevormd door de buitenschoolse activiteiten. Op dit gebied was ik erg blij met de komst van Marcel. Naast een gemeenschappelijke interesse voor feesten konden we ons goed inleven in elkaars ‘sores en mores’. Ik ben blij dat we de komende jaren in dezelfde stad blijven wonen.

119
Dankwoord

Voor alle gezelligheid in en buiten de kroeg bedank ik Vivek, Sonja, Henk en Ger. Natuurlijk bedank ik ook Annet voor de ‘praatpauzes’ en haar praktisch regeltalent.

Mensen die invloed hebben gehad die wat verder terug gaat in de tijd zijn mijn ‘natuurkunde vrienden’ Pieter, Han en Steven. Wellicht was ik zonder hun aanwezigheid nooit OIO geworden. Nog verder terug in de tijd, daar waar mijn wetenschappelijke nieuwsgierigheid zich begon te openbaren en daar waar ik voor het eerst zelfstandig leerde zijn, stonden mijn HTS vrienden, Carst en Sven aan mijn zijde. Wellicht was er helemaal geen proefschrift geweest als ik als kind niet voldoende had gespeeld. Ik bedank Fer voor al het maken van cassettebandjes, de verkleedpartijen en het hutten bouwen.

Natuurlijk bedank ik Cisca en Frans voor de steun en liefde door alle perioden heen. Lat-but-not-least bedank ik mijn vriendin Inge, die samen met mij ook door de zware tijden heen moest, voor het toevoegen van een extra dimensie in mijn leven.

September 2001
Machiel Westerdijk
Curriculum Vitae