Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®
Neurocomputing 56 (2004) 167–185

NEUROCOMPUTING

www.elsevier.com/locate/neucom

Improving classification with latent variable models by sequential constraint optimization

Machiel Westerdijk, Wim Wiegerinck*

Department of Medical Physics and Biophysics, SNN, University of Nijmegen, P.O. Box 9101, Nijmegen 6500 HB, The Netherlands

Received 1 December 2000; received in revised form 4 July 2002; accepted 1 October 2003

Abstract

In this paper we propose a method to use multiple generative models with latent 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 latent 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 latent variable models. The first has a discrete latent state space and the second, principal component analysis, has a continuous latent state space. We compare the effectiveness of both approaches on a handwritten digit recognition problem and on a satellite image recognition problem.

Keywords: Latent variable models; Semi-supervised learning; Support vector machines; PCA; Vector quantization; Image and character recognition

1. Introduction

Probabilistic graphical models, such as hidden Markov models [2], sigmoid belief networks [16] and hierarchical mixtures of experts [12] are excellently suited to

* Corresponding author. Tel.: +31-24-3615040; fax: +31-24-3541435.
E-mail addresses: machiel.westerdijk@cgey.nl (M. Westerdijk), wimw@mbfys.kun.nl, wimw@snn.kun.nl (W. Wiegerinck).

0925-2312/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2003.10.001

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 [22] 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) [9] 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 latent 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 paper we deal with this problem by using *deterministic* generative models, which have been introduced in [23,24]. In these models, the layer of latent 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 [5] to the

generative models. Techniques, which are based on the latter approach, are not new. They have been developed, for example, for vector quantization [14].

In Section 3, we derive a new learning rule from the constraint minimization objective. In a recent paper [25], we have derived a similar learning rule within the MED framework for one specific type of latent 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 latent 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 4.

In the next section, we start with our definition of latent variable models and its special case of linear Gaussian latent variable models.

2. Latent variable models

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

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

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

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

$$L(\theta|D) = \sum_{\mu} \log\, p(\mathbf{x}_\mu|\theta). \tag{2}$$

In this paper we also consider *deterministic* generative models. In a deterministic model, only a single latent state $\mathbf{s}_\mathbf{x}$ is responsible for generating a visible state $\mathbf{x}$. We will consider such models as limiting cases of probabilistic generative models, i.e., the probability distribution of $p(\mathbf{s}|\mathbf{x}, \theta)$ is sharply peaked around $\mathbf{s}_\mathbf{x}$. In this limit, the probability of latent states other than $\mathbf{s}_\mathbf{x}$ are infinitesimally small. However we can still do likelihood maximization for a given data, or compute the latent 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.

2.1. Linear Gaussian latent variable models

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

$$p(\mathbf{x}|\mathbf{s},\theta) = \|2\pi\Sigma\|^{-1/2} \exp\left(\frac{1}{2}(\mathbf{x}-F\mathbf{s})\Sigma^{-1}(\mathbf{x}-F\mathbf{s})\right) \tag{3}$$

in which $\theta = (F,\Sigma)$ are model parameters. F is a real-valued $D \times n$ matrix with columns $\mathbf{f}_1,\ldots,\mathbf{f}_n$, and $F\mathbf{s} \equiv \sum_{i=1}^{n} \mathbf{f}_i s_i$. The column vectors $\mathbf{f}_i$ 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), [11], factor analysis (FA) [1] and vector quantization (VQ) [8]:

- Standard PCA is a deterministic model. The starting point is a linear Gaussian model with continuous latent states s_i which have a prior distribution $p(\mathbf{s})$ 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 $\mathbf{f}_i$ around which the data are uniformly scattered with variance σ^2. The deterministic model can be viewed as a $\sigma \to 0$ limit. One can show, see [21], that in this limit, the maximum likelihood solution for the features $\mathbf{f}_i$ are the principal components of the covariance matrix of the data $\langle (x_i - \langle x_i\rangle)(x_i - \langle x_i\rangle)\rangle$ where the averages are taken over the data set. In the finite σ case with $p(\mathbf{s}) \sim \mathcal{N}(0,1)$ the model is known as Probabilistic PCA (PPCA) [20].
- **Factor analysis** is a proper probabilistic linear Gaussian model. In this model, the continuous latent states s_i have a Gaussian prior $p(s_i) \sim \mathcal{N}(0,1)$. The covariance matrix is related to the features F in the following manner $\Sigma = FF^{\mathrm{T}} + R$, 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 latent states $\mathbf{s}$ are binary (0/1) vectors with exactly one component being 1. So the state space consists of states $\mathbf{s}_1$=$(1,0,0,\ldots,0)$, $\mathbf{s}_2$=$(0,1,0,\ldots,0)$, …, $\mathbf{s}_n = (0,0,0,\ldots,1)$. Each of these states corresponds to a unique vector $\mathbf{f}_i = F\mathbf{s}_i$. In Gaussian mixture models the covariance matrix Σ may depend on $\mathbf{s}_i$ and is found by maximum likelihood estimation through an expectation maximization (EM) procedure.

Now consider a Gaussian mixture model with a covariance matrix proportional to the identity matrix $(\Sigma)_{ij} = \sigma^2\delta_{ij}$. In the deterministic limit $\sigma \to 0$, the distribution of the latent states given a certain input $p(\mathbf{s}|\mathbf{x})$ will then peak at a single state $\mathbf{s}_i$. In that case, if we take the priors $p(\mathbf{s}_i)$ to be constant, the Gaussian mixture model reduces to *vector quantization*. The corresponding centers $\mathbf{f}_i$ are known as codebook vectors and represent cluster centers in the data. Finally, if more components of the states $\mathbf{s}$ are allowed to have value 1 e.g., $\mathbf{s} = (1,0,1,0,1,1)$ such that the codebook vectors are additive compositions of the basic features we have *generative vector quantization*

[23,25]. We will discuss this model in more detail in Section 4. The probabilistic counterpart of the model is known as the cooperative vector quantization model [27].

3. Sequential constraint optimization

Suppose we have a set of training data $\{\mathbf{x}_\mu\}$ 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 $\mathbf{x}_\mu$. 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

$$\mathscr{L}(\mathbf{x}|\theta) = \log \frac{p(\mathbf{x}|\theta^{1})}{p(\mathbf{x}|\theta^{-1})}, \tag{4}$$

where $p(\mathbf{x}|\theta^{1})$ is the model trained on the class 1 data and $p(\mathbf{x}|\theta^{-1})$ 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 θ^{c} of each model to improve the classification boundary.

The first assumption that we make is that the pattern $\mathbf{x}$ is generated by a single latent state $\mathbf{s}_\mathbf{x}$ namely that state $\mathbf{s}_\mathbf{x}$ with the maximum a posteriori (MAP) probability

$$\mathbf{s}_\mathbf{x} = \arg\max_{\mathbf{s}} p(\mathbf{s}|\mathbf{x})\, p(\mathbf{s}). \tag{5}$$

Hence, we approximate the distribution over patterns $p(\mathbf{x})$ with

$$p_M(\mathbf{x}|\theta^{c}) = p(\mathbf{x}|\mathbf{s}_\mathbf{x}, F^{c})\, p(\mathbf{s}_\mathbf{x}). \tag{6}$$

In the Hidden Markov Model literature [18] the search for the MAP state $\mathbf{s}_\mathbf{x}$ 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 $\mathbf{s}_\mathbf{x}$ is already implicitly made.

The (soft) classification constraint that each training point $\mathbf{x}_\mu$ should satisfy is

$$y_\mu \mathscr{L}(\mathbf{x}_\mu|\theta)) \geqslant \gamma - \xi_\mu, \tag{7}$$

where γ is a parameter which has to be fixed in advance and ξ_μ is a variable which should be positive i.e., it is subject to the constraint $\xi_\mu \geqslant 0$. Suppose that both γ and ξ_μ in (7) are taken zero. In that case the constraint implies that y_μ and $\mathscr{L}(\mathbf{x}_\mu|\theta)$ 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, γ specifies a margin between the classification boundary and the data point $\mathbf{x}_\mu$. In many practical situations it will not be possible to find a parameter setting θ such that all training patterns are outside the margin. For this reason the *slack* variables ξ_μ are introduced, which allow for violations of the strict margin constraint.

While satisfying the constraints (7) and $\xi_\mu \geqslant 0$ we want the new model $\theta_0 + \Delta\theta$ to be as *close* to the initial model θ_0 as possible. Furthermore, we want to minimize the

training errors ξ_μ. The new model that meets these criteria is found by minimizing

$$E(\theta,\xi)=\sum_c(\theta^c-\theta_0^c)^{\mathrm{T}}S(\theta^c-\theta_0^c)+C\sum_\mu\xi_\mu \tag{8}$$

with respect to θ and ξ_μ subject to the constraints (7) and $\xi_\mu\geqslant 0$. The local metric structure in θ-space close to θ_0 is parameterized by the symmetric positive-definite Fisher matrix $S=\langle\nabla_\theta\log p(\mathbf{x}|\theta)\nabla_\theta^{\mathrm{T}}\log p(\mathbf{x}|\theta)\rangle$. The distance $(\theta^c-\theta_0^c)^{\mathrm{T}}S(\theta^c-\theta_0^c)$ corresponds to a second-order expansion of the Kullback–Leibler divergence at θ_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 θ.

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

$$L(\theta,\lambda)=\sum_c(\theta^c-\theta_0^c)^{\mathrm{T}}S(\theta^c-\theta_0^c)-\sum_\mu\lambda_\mu\{(y_\mu\mathscr{L}(\mathbf{x}_\mu|\theta))-\gamma\}. \tag{9}$$

has the same minimum for θ as (8) in the constraint region given by (7) and $\xi_\mu\geqslant 0$ if $L(\theta,\lambda)$ is at a maximum for the Lagrange multipliers λ subject to the constraints $0\leqslant\lambda_\mu\leqslant C$.

By setting the derivatives of $L(\theta,\lambda)$ w.r.t. to the model parameters θ equal to zero one arrives at the dual objective function $J(\lambda)$ which has to be maximized with respect to λ. In general, it is not possible to find a suitable expression for this dual objective function. In Appendix A, we show that if the discriminant function $\mathscr{L}(\mathbf{x}|\theta)$ is linearly dependent on the parameters θ in a region around θ_0 then the λ_μ are given by the solution of the following quadratic form:

$$J(\lambda)=L^{\mathrm{T}}\lambda-\lambda^{\mathrm{T}}Q\lambda, \tag{10}$$

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

$$Q_{\mu,\mu'}=\frac{1}{4}\,y_\mu y_{\mu'}\sum_c H(\mathbf{x}_\mu|\theta,c)^{\mathrm{T}}S^{-1}H(\mathbf{x}_{\mu'}|\theta,c). \tag{11}$$

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

$$L_\mu=\gamma-y_\mu\mathscr{L}(\mathbf{x}_\mu|\theta). \tag{12}$$

which specifies how much each pattern $\mathbf{x}_\mu$ is in- or outside the margin (specified by γ). Recently, due to the interest in support vector machines (SVM), many efficient techniques [4,10,19] have been developed with which the quadratic programming problem can be solved. A brief description of the method we used for this paper is given in Appendix B.

After obtaining the Lagrange multipliers λ that maximize (10) we can construct the new, improved, model parameters $\theta^c\leftarrow\theta_0^c+\Delta\theta^c$. The shift $\Delta\theta^c$ is given by a

weighted combination of the derivatives H of the discrimination error $\mathscr{L}$,

$$\Delta\theta^c = \frac{1}{2} S^{-1} \sum_\mu \lambda_\mu y_\mu H(\mathbf{x}|\theta, c). \tag{13}$$

Due to the nonlinearity of $\mathscr{L}$, the resulting parameters θ^c are only an approximation of the desired parameters that minimize (9). The assumption that the obtained θ^c are close to θ_0, motivates us to do the optimization a second time, but now with the obtained θ^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 $\mathscr{L}$ is linearized around the initial parameters θ_0. For the linearized discriminant function, the optimal parameters θ^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 $\theta^c \leftarrow \theta^c + \Delta\theta^c$ until some convergence criterion is reached (see Section 4).

3.1. The standard perceptron learning rule

The Lagrange multipliers in the SCO 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:

$$\lambda_\mu = \begin{cases} C & y_\mu \mathscr{L}(\mathbf{x}_\mu|\theta_t) < \gamma, \\ 0 & y_\mu \mathscr{L}(\mathbf{x}_\mu|\theta_t) \geqslant \gamma. \end{cases} \tag{14}$$

Hence, only those patterns that are close (within a region of size γ) 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 [6]), 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 (LVQA) [13]. Other choices for the loss function $\lambda(\mathbf{x})$ have also been proposed, such as in the LVQ2 learning algorithm [14].

A more direct correspondence between BPA and the SCO algorithm can be made by considering the limit $C \to 0$. 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.

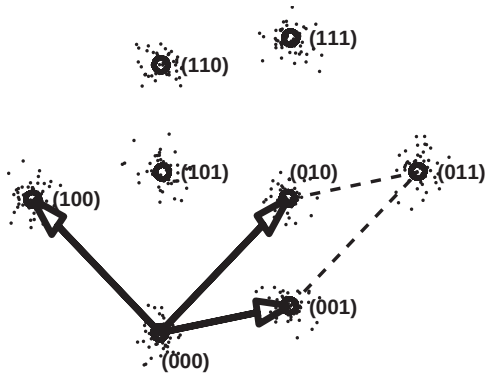

Fig. 1. The cluster (circles) are generated by a small set of basic features $\mathbf{f}_1, \mathbf{f}_2$ and $\mathbf{f}_3$, which correspond to the states $(1\,0\,0), (0\,1\,0)$ and $(0\,0\,1)$, respectively. The cluster corresponding to the state $(0\,1\,1)$, for example, is given by the sum of the features corresponding to the two states $(0\,0\,1)$ and $0\,1\,0$ (see broken lines).

4. Application to linear models and results

4.1. Application to generative vector quantization

In this section, we apply the SCO technique to generative vector quantization (GVQ) models discussed in [23]. 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 latent layer of n binary units with states $\mathbf{s} \in \{0,1\}^n$ and a continuous visible layer (the layer corresponding to the data) with values $\mathbf{x} = (x_1, \ldots, x_d, \ldots, x_D) \in \Re^D$. The prior latent state distribution $p(\mathbf{s})$ 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 Fig. 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 $\mathbf{x}$ the single most nearby cluster center $\mathbf{s}_\mathbf{x}$ (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 $\mathbf{s}$. In a recent paper [24] 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 latent variables.

Application of the SCO algorithm to GVQ models is quite straightforward. For these linear deterministic models, the log-likelihood at a data point $\mathbf{x}^\mu$ is proportional to the

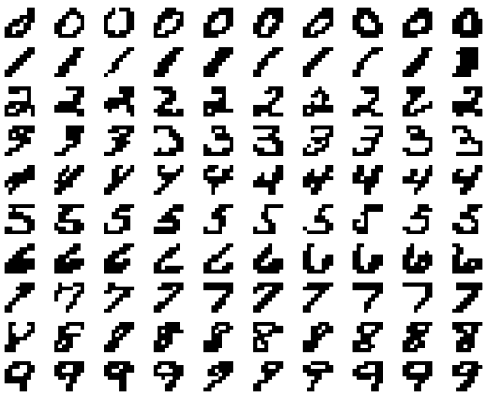

Fig. 2. A sample of binary images of handwritten digits.

'distance' of the data point to the model:

$$\log p(\mathbf{x}_\mu|\theta^c) \propto -\|\mathbf{x}^\mu - F^c\mathbf{s}^c_{\mathbf{x}}\|^2,$$

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

$$z^c_{i\mu} \equiv x_{i\mu} - \sum_l f^c_{il} s^c_{l\mu},$$

where $s^c_{l\mu}$ is the lth component of $\mathbf{s}^c_{\mathbf{x}}$, and f^c_{il} are the matrix entries of F^c. The derivative of the discriminant function w.r.t. to feature component $f^c_{\mathrm{d}n}$ is then

$$H(\mathbf{x}_\mu|\theta^c)_{\mathrm{d}n} = -2z^c_{\mathrm{d}\mu} s^c_{n\mu}, \tag{15}$$

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

$$S^c_{imjn} = 4/N_c \sum_\mu z^c_{i\mu} z^c_{j\mu} s^c_{m\mu} s^c_{n\mu}, \tag{16}$$

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

4.1.1. Results on handwritten digit recognition

The data set we used to test our method consisted of 11 000 handwritten digits compiled by the US Postal Service Office of Advanced Technology. We used the same preprocessed data as Saul et al. (1996) [20] and Sallans et al. (1998) [19]. Each processed image is built up out of 8×8 black and white pixels. A data sample of each digit class is shown in Fig. 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 the behavior of SCO. For both methods we chose $\gamma = 1$ 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 four or five 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 four 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.

Fig. 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 four feature vectors $\mathbf{f}_i$. 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 one after only three iterations. Although, SCO needs fewer iterations to converge, the training time needed per iteration differs considerably for SCO and BPA. In Fig. 4, we plotted the CPU time as a function of the number of iterations. Clearly, in the first two iterations SCO requires a lot of

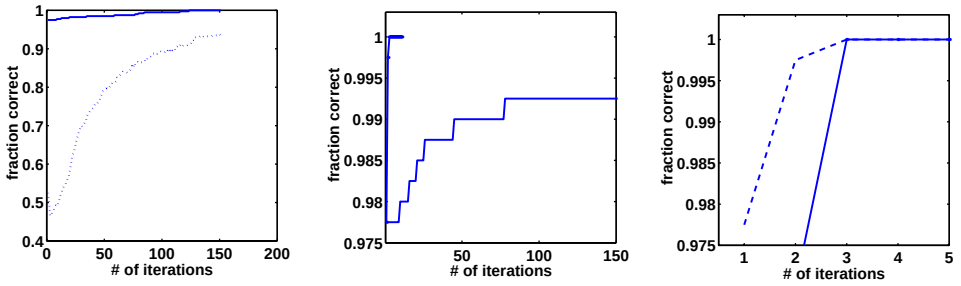

Fig. 3. Learning curves on two classes (3 and 5) of the digit data set. The plots only differ by the range of the horizontal and vertical axis. Left: Black solid curve: BPA-ML initialized with the maximum likelihood solution. Dotted curve: BPA-R. In both cases $C = 2 \times 10^{-4}$. Middle: Rescaled 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.

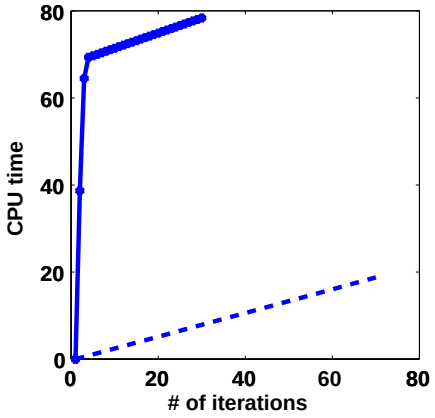

Fig. 4. Comparison of CPU times (s): SCO (solid line) and Perceptron (dashed line).

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 Fig. 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

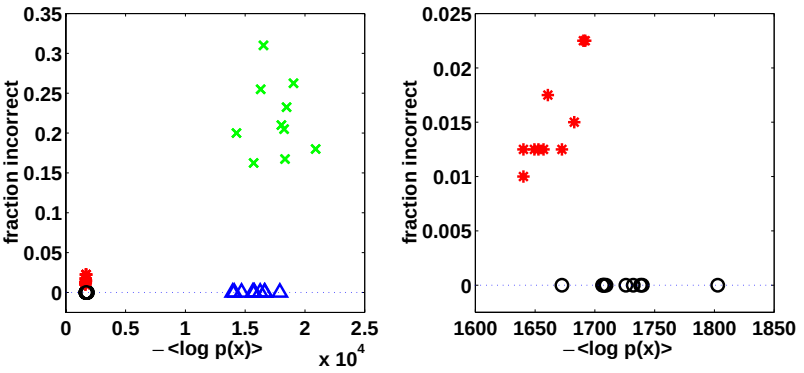

Fig. 5. Fraction of mis-classification versus the log-likelihood on the training set after learning. The labeling is chosen as follows: 'x'=BPA-R, 'Δ'=SCO-R, 'o'=SCO-ML and '*'=BPA-ML. All experiments were repeated 10 times with different randomly selected training sets. Left: Large ranges of the horizontal 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.

Table 1
Test error rates on discriminating between 3's and 5's using GVQ models with four latent variables

ML	SCO-R	BPA-R	SCO-ML	BPA-ML
$6 \pm 1\%$	$11 \pm 1\%$	$24 \pm 5\%$	$5 \pm 1\%$	$5 \pm 1\%$

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 $-\langle \log p(\mathbf{x}) \rangle = 1649$. 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 1. The first column 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) [20] and Hinton et al. (1995) [7]. 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

Table 2
Test error rates on the full 10-class digit recognition problem. Each GVQ model had eight latent variables

GVQ max. likelih. (%)	Nearest neighbor (%)	Back prop. (%)	Wake-sleep (%)	GVQ-SCO/BPA (%)	Mean field (%)
7.8	6.7	5.6	4.8	4.8	4.6

digit class, one for the positive training examples (the 700 examples corresponding to the digit) and one for the negative training examples (the $9 \times 700 = 6300$ examples corresponding to the remaining digits).[1] Cross-validation within the training set, where we varied the number of latent states from 1 to 8, revealed that we had to choose at least eight latent 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 paper, we did not incorporate any approximating techniques to find the Viterbi solutions $\mathbf{s}_\mathbf{x}$ 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 [25]. With the eight latent 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 h.

To classify a test example $\mathbf{x}$ we computed its distance $E_i^c(\mathbf{x}) = \min_\mathbf{s} \|\mathbf{x} - M_{(i)}^c \mathbf{s}\|$ to both the positive $c = 1$ and negative $c = -1$ models of each class i. We then computed, for each digit i, the difference ΔE_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 ΔE_i was minimal was then chosen to label the test example $\mathbf{x}$. This procedure corresponds to the intuitive notion that an example belonging to a certain class should be close to the model corresponding to that class and far away from the negative model of that class.

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 8×8 grid of visible units, a middle layer of 24 binary latent units and a top layer of 8 binary units.

An overview of the test error results is presented in Table 2.

[1] As in SVM, the SCO framework can also be extended to multi-class classification problems in a direct way. On the other hand, there are indications [26] that, for practical purposes, the multi-class formulation does not have a great advantage over the multiple two-class approach.

4.2. Application to principal component analysis

PCA 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 often used 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 [3]. In this section, we apply SCO to PCA 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 [21]. The latent states s_i are continuous in PCA. The Viterbi solution $\mathbf{s}_\mathbf{x}$ corresponding to data pattern $\mathbf{x}$ is simply given by the projection of $\mathbf{x}$ onto the columns $\mathbf{f}_i$ of F, where the vectors $\mathbf{f}_i$ now represent the principal components. In other words, in PCA the components of the states $\mathbf{s}_x$ are given by $s_{i\mathbf{x}} = \mathbf{f}_i \cdot \mathbf{x}/\|f_i\|^2$. A data pattern is classified by comparing the distances of the pattern to the projections: $\mathscr{L}(\mathbf{x}_\mu) = -\sum_c c\|\mathbf{x}_\mu - F^c\mathbf{s}^c_{\mathbf{x}_\mu}\|^2$.

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)

4.2.1. Results

We show results of SCO applied to PCA on the Landsat Satellite Image data set obtained from the Statlog data repository [15]. The spatial resolution of a pixel in a Landsat Image is about 80 m × 80 m. Each pattern in the data set corresponds to a 3 × 3 square neighborhood of pixels. Such a pattern contains the pixel values in the four spectral bands of each of the 9 pixels in the 3 × 3 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 gray soil (21.67%), 4. damp gray soil (09.36%), 5 soil with vegetation stubble (10.60%), 6. very damp gray 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 Fig. 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 $2 \pm 1\%$ of the test data. The initial PCA models misclassified $3 \pm 1\%$. 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 Fig. 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

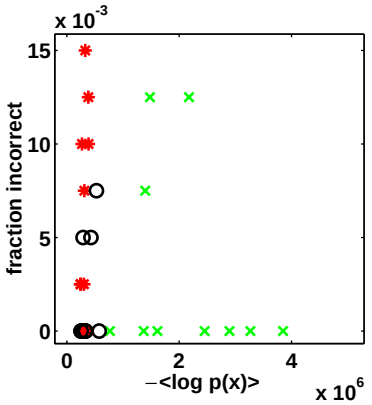

Fig. 6. Results of PCA learning on the LandSat image data. Each point represents the fraction of mis-classification versus the log-likelihood on the training set after learning. The labeling is chosen as follows: 'x'=BPA-R, 'o'=SCO-ML and '*'=BPA-ML. SCO-R points fall outside the figure. All experiments were repeated 10 times with different randomly selected training sets.

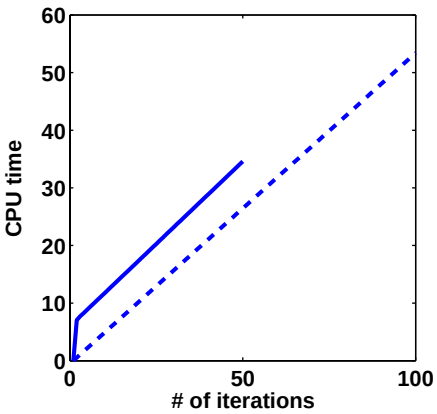

Fig. 7. Comparison of CPU times: SCO (solid line) and Perceptron (dashed line).

classification problems for which we trained 2×6 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 (5×200 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, 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.

Table 3
Test error rates on the Landsat image data base

	PCA max. likelih.	PCA BPA	PCA SCO
Train error	27%	16.2%	2.5%
Test error	27.4%	18.2%	17.4%

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 paper 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 latent 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 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 SVM. 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 handwritten digits and Landsat image datasets with all classes, we conclude that post-training (using either SCO or BPA) after maximum likelihood training does greatly improve the classification performance of the latent variable models.

Acknowledgements

We would like to thank the anonymous reviewers for their helpful comments and suggestions.

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

We want to find a new model θ as close as possible to the original model θ_0 such that most training patterns $\mathbf{x}_\mu$ are outside the margin (parameterized by γ). The solution is found by minimizing the Lagrangian

$$L(\theta,\lambda)=\sum_c(\theta^c-\theta_0^c)\,S\,(\theta^c-\theta_0^c)-\sum_t\lambda_t\{(y_t\mathscr{L}(\mathbf{x}|F))-\gamma\}, \tag{A.1}$$

with respect to the model parameters θ and to maximize it with respect to the Lagrange multipliers λ. Since $\mathscr{L}(\mathbf{x}|F)$ is, in general, a non-linear function of θ we optimize (A.1) by considering sufficiently small steps in $\Delta\theta=\theta_{t+1}-\theta_t$. In that case the discriminant function can be approximated with the first-order expansion

$$\mathscr{L}(\mathbf{x}|\theta_t^c+\Delta\theta^c)=\mathscr{L}(\mathbf{x}|\theta_t^c)+H_{\mathbf{x}_t,\theta_t^c}^{\mathrm{T}}\cdot(\Delta\theta)+\mathscr{O}((\Delta\theta)^2), \tag{A.2}$$

where $H_{\mathbf{x}_t,\theta_t^c}=\nabla_\theta\mathscr{L}(\mathbf{x}|\theta^c)|_{\theta=\theta_t}$. The approximation $L^*(\theta,\lambda)$ to $L(\theta,\lambda)$ then becomes

$$\begin{aligned}L^*(\theta,\lambda)=&\sum_c(\theta_t^c+\Delta\theta)^{\mathrm{T}}S^{-1}(\theta_t^c+\Delta\theta)\\&-\sum_t\lambda_t\left\{\left(y_t\sum_c H_{\mathbf{x}_t,\theta_t^c}^{\mathrm{T}}\cdot\Delta\theta\right)-\gamma_t\right\}-\sum_t\lambda_t y_t\mathscr{L}(\mathbf{x}|\theta_t),\end{aligned} \tag{A.3}$$

Setting the derivatives of (A.3) w.r.t. $\Delta\theta^c$ equal to zero gives the optimal value of $\Delta\theta^c$ in terms of the Lagrange multipliers

$$\nabla_{\theta^c}L^*(\theta,\lambda)=0\rightarrow\Delta\theta^c=\frac{1}{2}S^{-1}\sum_t\lambda_t y_t H_{\mathbf{x}_t,\theta_0^c}. \tag{A.4}$$

By substituting (A.4) into (A.3) we obtain the objective function $J(\lambda)$ for the Lagrange multipliers

$$\begin{aligned}J(\lambda)=&\sum_\mu\lambda_\mu\gamma-\frac{1}{4}\sum_\mu\sum_{\mu'}\lambda_\mu\lambda_{\mu'}y_\mu y_{\mu'}\sum_c H_{\mathbf{x}_\mu,\theta_t^c}^{\mathrm{T}}S^{-1}H_{\mathbf{x}_{\mu'},\theta_t^c}\\&-\sum_\mu\lambda_\mu y_\mu\{\mathscr{L}(\mathbf{x}_\mu|\theta_t)\},\end{aligned} \tag{A.5}$$

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

In a more compact notation

$$J(\lambda)=L^{\mathrm{T}}\lambda-\lambda^{\mathrm{T}}Q\lambda, \tag{A.6}$$

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

$$Q_{\mu,\mu'}=\frac{1}{4}y_\mu y_{\mu'}\sum_c H(\mathbf{x}_\mu|\theta_t^c)^{\mathrm{T}}S^{-1}H(\mathbf{x}_{\mu'}|\theta_t^c) \tag{A.7}$$

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

$$L_\mu=\gamma-y_\mu\{\mathscr{L}(\mathbf{x}_\mu|\theta_t)\}. \tag{A.8}$$

Appendix 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 (SMO) algorithm [17] which is developed for training SVM classifiers. At each stage in this algorithm two Lagrange multipliers λ_t are selected for optimization while holding the others fixed. This is repeated until the error is within a ε 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 λ_t.

A new Lagrange multiplier λ_μ is selected by comparing all derivatives $\partial J/\partial\lambda_\mu$. We know that if λ 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 λ_μ 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 $\lambda_\mu > 0$.

References

[1] D. Bartholomew, Latent Variable Models and Factor Analysis, Grin's Statistical Monographs and Courses, Charles Grin & Company Limited, London, 1987.
[2] L. Baum, T. Petrie, G. Soules, N. Weiss, A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains, Ann. Math. Statist. 41 (1970) 164–171.
[3] V. Brennan, J. Principe, Face classification using PCA and multiresolution, in: Proceedings of the IEEE Workshop on Neural Networks for Signal Processing, Cambridge, 1998, pp. 506–515.
[4] J.R. Bunch, K. Kaufman, Some stable methods for calculating inertia and solving symmetric linear systems, Math. Comp. 31 (1977) 162–179.
[5] R. Duda, P. Hart, Pattern Classification and Scene Analysis, Wiley, New York, 1973.
[6] R. Duda, P. Hart, D. Stork, Pattern Classification, Wiley, New York, 2000.
[7] G. Hinton, P. Dayan, B. Frey, R. Neal, The wake–sleep algorithm for unsupervised neural networks, Science 268 (1995) 1158–1161.
[8] J.J. Hull, A database for handwritten text recognition research, IEEE Trans. Pattern Anal. Mach. Intell. 16 (5) (1997) 550–554.
[9] T. Jaakkola, M. Meila, T. Jebara, Maximum entropy discrimination, Technical Report MIT-AITR-1668, MIT AI Lab, 1999.
[10] T. Joachims, Advances in Kernel Methods—Support Vector Learning, Making Large-Scale SVM Learning Practical, MIT Press, Cambridge, MA, 1999 (Chapter 11).
[11] I. Jolliffe, Principal Component Analysis, Springer, New York, 1986.
[12] M.I. Jordan, R.A. Jacobs, Hierarchical mixtures of experts and the EM algorithm, Neural Comput. 6 (1994) 181–214.
[13] T. Kohonen, Learning vector quantisation, Neural Networks 1 (1988) 3–16.
[14] T. Kohonen, The self-organizing map, Proc. IEEE 78 (9) (1990) 1464–1480.
[15] D. Michie, D.J. Spiegelhalter, C.C. Taylor (Eds.), Machine Learning, Neural and Statistical Classification, Ellis Horwood, Chichester, UK, 1994.
[16] R.M. Neal, Connectionist learning of belief networks, Artif. Intell. 56 (1991) 71–113.

[17] J. Platt, Advances in Kernel Methods—Support Vector Learning, Fast Training of Support Vector Machines using Sequential Minimal Optimization, MIT Press, Cambridge, MA, 1999.
[18] L. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition, Proc. IEEE 77 (1989) 257–285.
[19] B. Sallans, G.E. Hinton, Z. Ghahramani, A hierarchical community of experts, in: C.M. Bishop (Ed.), Neural Networks and Machine Learning, NATO ASI Series F, Springer, Berlin, 1998, pp. 269–284.
[20] L.K. Saul, T. Jaakkola, M.I. Jordan, Mean field theory for sigmoid belief networks, J. Artif. Intell. Res. 4 (1996) 61–76.
[21] M. Tipping, C. Bishop, Probabilistic principal component analysis, Technical Report NCRG/97/010, Neural Computing Research Group, Aston University, Birmingham, 1997.
[22] V. Vapnik, The Nature of Statistical Learning Theory, Springer, Berlin, 1995.
[23] M.J.D. Westerdijk, D. Barber, W. Wiegerinck, Generative vector quantisation, in: ICANN99: Artificial Neural Networks, Vol. 2, IEE, London, 1999, pp. 934–939.
[24] M.J.D. Westerdijk, D. Barber, W. Wiegerinck, Deterministic generative models for fast feature discovery, Data Mining and Knowledge Discovery 5 (4) (2001) 337–363.
[25] M.J.D. Westerdijk, W.A.J.J. Wiegerinck, Classification with multiple latent variable models using maximum entropy discrimination, in: Proceedings of the 17th International Conference on Machine Learning, Morgan Kaufmann, Stanford, CA, 2000, pp. 1143–1150.
[26] J. Weston, C. Watkins, Multi-class support vector machines, Technical Report CSD-TR-98-04, University of London, Department of Computer Science, 1998.
[27] R.S. Zemel, A minimum description length framework for unsupervised learning, Technical Report CRG-TR-93-2, University of Toronto, 1994.

M.J.D. Westerdijk received the M.S. degree in physics in 1995 from the Free University in Amsterdam. In 2001, he received a Ph.D. degree from the University of Nijmegen. Here he worked in the neural networks group SNN on novel machine learning algorithms with applications to data mining. The title of his thesis is 'Hidden Variable Models for Knowledge Discovery'. Currently, he is a consultant in the field of data mining and business intelligence at Cap Gemini Ernst & Young.

W.A.J.J. Wiegerinck received the M.Sc. degree (Hons.) in physics from the University of Amsterdam in 1988. From 1988 to 1990, he worked as research assistant at the Royal netherlands Meteorological Institute on dynamical systems. In 1990, he started as Ph.D. student at the University of Nijmegen, studying on-line learning in neural networks. He received his Ph.D. degree in 1996. Since 1995, he is researcher at SNN (Nijmegen). Since 2001, he is also assistant professor at the University of Nijmegen. His current research interest include theory and applications of graphical models.