IDIAP RESEARCH REPORT

IDIAP

Martigny - Valais - Suisse

Mixture Models for Unsupervised and Supervised Learning

Perry Moerland *

IDIAP–RR 00-18

June 2000

published as

Ph.D. thesis (#2189) accepted at the Computer Science Department of the Swiss Federal Institute of Technology at Lausanne (EPFL).
Thesis committee: Prof. Wulfram Gerstner (supervisor, EPFL), Prof. B. Faltings (EPFL), Dr. E. Mayoraz (Motorola), Prof. C. Pellegrini (University of Geneva), Dr. J. Schmidhuber (IDSIA).

Dalle Molle Institute for Perceptual Artificial Intelligence • P.O.Box 592 • Martigny • Valais • Switzerland

phone +41 – 27 – 721 77 11
fax +41 – 27 – 721 77 12
e-mail secretariat@idiap.ch
internet http://www.idiap.ch

* e-mail: Perry.Moerland@idiap.ch

IDIAP Research Report 00-18

Mixture Models for Unsupervised and Supervised Learning

Perry Moerland

June 2000

published as

Ph.D. thesis (#2189) accepted at the Computer Science Department of the Swiss Federal Institute of Technology at Lausanne (EPFL).

Thesis committee: Prof. Wulfram Gerstner (supervisor, EPFL), Prof. B. Faltings (EPFL), Dr. E. Mayoraz (Motorola), Prof. C. Pellegrini (University of Geneva), Dr. J. Schmidhuber (IDSIA).

Acknowledgements: This thesis has been supported by the Swiss National Science Foundation (21-45621.95 and 2100-55688.98).

Version Abrégée

Cette thèse s'inscrit dans le vaste domaine de l'apprentissage automatique. Dans une société qui produit et consomme une quantité croissante (voire débordante) d'information, des méthodes permettant de donner un sens à toute cette information deviennent indispensables. L'apprentissage automatique essaie de répondre à ce besoin par des modèles qui expliquent certains aspects d'un flot de données. Lors de la construction de tels modèles, il est important de se poser les questions suivantes :

- Quelle est la structure des données ? Cette question est particulièrement pertinente pour des données à haute dimension que l'on ne peut plus visualiser d'une façon informative.
- Quelles sont les caractéristiques principales des données ?
- Comment prédire si un motif appartient à une classe plutôt qu'à une autre ?

Cette thèse étudie ces trois questions avec, comme trait d'union, l'idée de construire des modèles complexes à partir de modèles très simples. La décomposition en sous-problèmes se traduit aussi dans les procédures utilisées pour estimer les valeurs des paramètres de ces modèles. Les algorithmes pour les modèles simples forment le noyau des algorithmes pour le modèle complexe.

Les questions posées ci-dessus sont traitées en trois volets :

Apprentissage non-supervisé Cette partie est consacrée au problème de l'estimation d'une densité de probabilité, qui a pour but de trouver une bonne représentation probabiliste des données. Un des modèles les plus utilisés pour l'estimation de densité est le mélange de Gaussiennes (ou multigaussiennes). Une alternative prometteuse au mélange de Gaussiennes consiste en un mélange de modèles à variables cachées comme l'analyse en composantes principales (ACP) ou l'analyse factorielle. L'avantage de ces modèles est qu'ils permettent de représenter des matrices de covariance avec un nombre inférieur de paramètres défini par le choix de la dimension d'un sous-espace. Une évaluation empirique sur une large collection de bases de données montre que des modèles à variables cachées donnent des résultats bien meilleurs que des multigaussiennes.

Pour pallier au choix par validation croisée de la dimension du sous-espace, une méthode d'estimation Bayesienne pour des mélanges de modèles à variables cachées est proposée. Cette méthode permet de déterminer automatiquement la dimension adéquate pendant l'entraînement du modèle.

Extraction de caractéristiques L'ACP est aussi (et surtout) une méthode classique pour l'extraction de caractéristiques. Cependant, elle est limitée à une extraction linéaire par le moyen d'une projection dans un sous-espace. L'ACP à base de fonctions à noyaux ("kernel PCA") permet l'extraction non-linéaire des caractéristiques des données. L'application de la kernel PCA à une base de données de N motifs demande de trouver les vecteurs propres d'une matrice de taille $N \times N$. Un algorithme "Expectation-Maximization" (EM) pour l'ACP qui ne nécessite pas le stockage de cette matrice, est adapté à la kernel PCA afin que l'on puisse l'appliquer aux grandes bases de données de plus de 10.000 motifs. Les expériences démontrent l'intérêt de cette approche et les caractéristiques extraites par ce

pré-traitement permettent l'entraînement de classifieurs simples mais performants. On décrit ici une nouvelle variante de l'algorithme EM pour l'ACP qui l'accélère considérablement en rendant possible l'adaptation des paramètres d'une façon incrémentale.

Apprentissage supervisé Cette partie montre deux manières de construire des modèles complexes à partir de modèles simples pour le problème de la classification. La première approche s'inspire directement des modèles de mélange pour l'apprentissage non-supervisé. Le modèle qui en résulte, nommé mélange d'experts, essaie de diviser un problème complexe en sous-problèmes et attribue des modèles simples à chaque sous-problème. La division de l'espace et la recombinaison des réponses des experts se fait par un autre modèle, nommé pondérateur, dépendant des entrées. Après une vue d'ensemble de ce modèle et des algorithmes existants destinés à l'entraîner, différents pondérateurs sont proposés et comparés. Parmi ceux-ci se trouvent les modèles de mélange pour l'apprentissage non-supervisé. Les expériences montrent qu'un mélange d'experts standard avec un réseau de neurones comme pondérateur donne les meilleurs résultats.

La deuxième approche est un algorithme constructif, nommé "boosting", et crée un ensemble de modèles en mettant de plus en plus de poids sur les données qui ont été classifiées d'une façon erronée par les classifieurs précédents. Un modèle a été développé qui se trouve à mi-chemin entre un mélange d'experts et le boosting. Le modèle ajoute au boosting une combinaison dynamique (comme un pondérateur). Ceci a l'avantage qu'avec un ensemble nettement plus petit le résultat obtenu est souvent aussi bon qu'avec le boosting. De plus, le modèle a des bases solides dans la théorie de l'apprentissage.

Finalement, les modèles étudiés ici ont été évalués sur deux bases de données dans le domaine de la vision. Les résultats confirment l'intérêt des mélanges de modèles à variables cachées avec lesquels on obtient des très bons résultats dans un classifieur Bayesien.

Summary

In a society which produces and consumes an ever increasing amount of information, methods which can make sense out of all this data become of crucial importance. Machine learning tries to develop models which can make the information load accessible. Three important questions one can ask when constructing such models are:

- What is the structure of the data? This is especially relevant for high-dimensional data which cannot be visualized anymore.
- Which features are most characteristic?
- How to predict whether a pattern belongs to one class or to another?

This thesis investigates these three questions by trying to construct complex models from simple ones. The decomposition into simpler parts can also be found in the methods used for estimating the parameter values of these models. The algorithms for the simple models constitute the core of the algorithms for the complex ones.

The above questions are addressed in three stages:

Unsupervised learning This part deals with the problem of probability density estimation with the goal of finding a good probabilistic representation of the data. One of the most popular density estimation methods is the Gaussian mixture model (GMM). A promising alternative to GMMs are the recently proposed mixtures of latent variable models. Examples of the latter are principal component analysis (PCA) and factor analysis. The advantage of these models is that they are capable of representing the covariance structure with less parameters by choosing the dimension of a subspace in a suitable way. An empirical evaluation on a large number of data sets shows that mixtures of latent variable models almost always outperform GMMs.

To avoid having to choose a value for the dimension of the subspace by a computationally expensive search technique such as cross-validation, a Bayesian treatment of mixtures of latent variable models is proposed. This framework makes it possible to determine the appropriate dimension during training and experiments illustrate its viability.

Feature extraction PCA is also (and foremost) a classic method for feature extraction. However, PCA is limited to linear feature extraction by a projection onto a subspace. Kernel PCA is a recent method which allows non-linear feature extraction. Applying kernel PCA to a data set with N patterns requires finding the eigenvectors of an $N \times N$ matrix. An Expectation-Maximization (EM) algorithm for PCA which does not need to store this matrix is adapted to kernel PCA and applied to large data sets with more than 10,000 examples. The experiments confirm that this approach is feasible and that the extracted features lead to good performance when used as pre-processed data for a linear

classifier. A new on-line variant of the EM algorithm for PCA is presented and shown to speed up the learning process.

Supervised learning This part illustrates two ways of constructing complex models from simple ones for classification problems. The first approach is inspired by unsupervised mixture models and extends them to supervised learning. The resulting model, called a mixture of experts, tries to decompose a complex problem into subproblems treated by several simpler models. The division of the data space is effectuated by an input-dependent gating network. After a review of the model and existing training methods, different possible gating networks are proposed and compared. Unsupervised mixture models are one of the evaluated options. The experiments show that a standard mixture of experts with a neural network gate gives the best results.

The second approach is a constructive algorithm called boosting which creates a committee of simple models by emphasizing patterns which have been frequently misclassified by the preceding classifiers. A new model has been developed which lies between a mixture of experts and a boosted committee. It adds an input-dependent combiner (like a gating network) to standard boosting. This has the advantage that with a considerably smaller committee results are obtained which are comparable to those of boosting.

Finally, some of the investigated models have been evaluated on two problems of machine vision. The results confirm the potential of mixtures of latent variable models which lead to good performance when incorporated in a Bayes classifier.

Contents

List of Figures

List of Tables

Notation and Abbreviations

Upper-case bold letters denote matrices, for example $\mathbf{R}$. Vectors are denoted by lower-case bold letters, for example $\mathbf{x}$, and are column vectors. Subscripting is used for indexing; thus, x_i denotes the ith element of vector $\mathbf{x}$ and R_{ij} the element in the ith row and jth column of matrix $\mathbf{R}$.

$\cdot^T$	transpose (of a vector or a matrix), thus $\mathbf{x}^T = (x_1, x_2, \ldots, x_d)$
$\cdot \sim \cdot$	"distributed according to"
$\cdot \propto \cdot$	"proportional to"
$\langle \cdot \rangle$	expectation of a random variable
$\vert \cdot \vert$	determinant
$\vert\vert \cdot \vert\vert$	length of a vector
$\cdot \otimes \cdot$	Kronecker product of two matrices (A.13)
$\cdot \circ \cdot$	Element-wise product of two matrices (A.14)
$\mathbf{0}$	all-zero matrix
α_j	mixing coefficients (but sometimes also Lagrange coefficients)
$\boldsymbol{\gamma}$	vector of all hyperparameters
γ_i	regularization parameter or hyperparameter
δ	the number of well-determined parameters
δ_{ij}	Kronecker delta ($\delta_{ij} = 1$ if $i = j$, 0 otherwise)
μ	mean of a Gaussian distribution
$\boldsymbol{\theta}$	parameters of a model
$\pi_j(\cdot, \cdot)$	posterior probabilities in a mixture of experts
$\boldsymbol{\Sigma}$	covariance matrix of a multivariate Gaussian distribution
$\sigma(\cdot)$	logistic or sigmoid activation function: $1/(1 + \exp(-x))$
σ^2	variance of a Gaussian distribution
$\Phi(\cdot)$	map from data space to feature space
$\phi(\cdot)$	probability density function
$\Psi(\cdot)$	empirical kernel map
$\mathbf{a}$	summed inputs to the output units of a neural network model
a_j	summed input to the jth output unit of a neural network model
C	number of classes
$\mathcal{C}_k$	class k
C^n_k	n choose k: $\binom{n}{k}$
D	training set
d	dimension of input (or data) space
$\mathrm{diag}(\cdot)$	diag operator on a vector (A.11)
$E(\cdot)$	error function
$\mathcal{E}(\cdot)$	expectation of a random variable
F	feature space
$f(\cdot)$	activation function

g_j	jth output of a gating network (more explicitly $g_j(\mathbf{x}, \boldsymbol{\theta}_g)$)
$h_j(\cdot)$	posterior probabilities in a mixture model
$\mathbf{I}$	identity matrix
$\mathbf{I}_k$	identity matrix of size $k \times k$
$\mathbf{J}$	all-ones matrix
$\mathcal{L}(\cdot)$	likelihood function
K	kernel matrix
$k(\cdot, \cdot)$	kernel function
ℓ	dimension of latent space
$\ln(\cdot)$	logarithm to base e
m	number of mixture components
N	number of training examples
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	multivariate Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$
$P(\cdot)$	probability
$p(\cdot)$	probability density function
$\mathbf{R}$	factor analysis noise covariance matrix
$\mathbb{R}$	the set of real numbers
$\mathbf{S}$	sample covariance matrix
$\mathbf{T}$	matrix of target outputs
$\mathbf{t}^n$	target output n
$\mathrm{tr}(\cdot)$	trace operator
$\mathbf{W}$	factor loading matrix
$\mathbf{w}$	parameters of a model (often the weights)
$\mathbf{w}_j$	parameters of sub-model j
$\mathbf{v}$	parameters of a gating network
$\mathrm{vec}(\cdot)$	vec operator on a matrix
$\mathbf{X}$	matrix of input patterns
$\mathbf{x}^n$	input pattern n
$\{\mathbf{x}^n, \mathbf{t}^n\}$	data set
$\mathbf{y}$	output of a model (more explicitly $\mathbf{y}(\mathbf{x}, \boldsymbol{\theta})$)
$\mathbf{y}_j$	output of sub-model j
y_j	jth output of a model
$\mathbf{z}$	missing or latent variables

ARD	automatic relevance determination
CV	cross-validation
EM	Expectation Maximization (algorithm)
FA	factor analysis
GLM	generalized linear model
GMM	Gaussian mixture model
HME	hierarchical mixture of experts
IRLS	iteratively reweighted least squares
ME	mixture of experts
MFA	mixture of factor analyzers
ML	maximum likelihood
MLP	multi-layer perceptron
MPCA	mixture of principal component analyzers
NN	neural network

PCA	principal component analysis
PPCA	probabilistic principal component analysis
RBF	radial basis function
SVD	singular value decomposition
SVM	support vector machine

CHAPTER 1

Introduction

When looking back it is hard, even for me, to imagine that this thesis started out of an interest in optical implementations of artificial neural networks. This indeed seems to have little to do with the title which you, dear reader, saw on the cover page. In a certain sense, this change of scope is illustrative for the development the neural network field has gone through over the past decade. Shortly after its revival during the eighties, it was characterized by a period of optimism and exaggerated claims about machines which mimic the way the human brain processes information. While this anthropomorphic metaphor has certainly been an inspiration for some of the models developed during this period, it seems to me that has also been counterproductive. The fact that we do not understand most of the processes in the brain might have enhanced the idea that one need not really understand the model which has been inspired by it. It has also led to reinventing wheels already developed in older fields such as pattern recognition and statistics.

Luckily things have changed considerably by now and the artificial neural network field has found its place among these older disciplines as a form of applied statistics (Bishop 1995; Ripley 1996). It is also in this context that one should place this thesis. Almost all models which will be discussed have their roots in *probability* theory which provides a fruitful framework for the problem I was interested in: *learning* from examples in the presence of uncertainty.

This problem arises in a context in which we are given a data set of examples and our goal is to automatically find structure in these examples. We can discern at least two different settings of the learning problem: unsupervised and supervised. The problem of *unsupervised* learning is the one which is more difficult to define and more diverse. Typical examples are the problem of grouping similar data into clusters or the extraction of useful features from data. Imagine, for example, that the given data set comes from a high-dimensional space and we would like to visualize the data in two-dimensional space to make it easier to interpret for a human observer. A way to tackle this problem could be to cluster the data and, as a form of feature extraction, find an informative projection into two dimensions for each cluster separately. As we will see, a convenient way of formulating many forms of unsupervised learning in probabilistic terms is as a density estimation problem.

We speak of *supervised* learning when the data set D consists of pairs of *patterns* $\mathbf{x}^i$ and *outputs* $\mathbf{t}^i$ represented as vectors:

$$D = \{(\mathbf{x}^1, \mathbf{t}^1), \ldots, (\mathbf{x}^N, \mathbf{t}^N)\},$$

known as the *training* set. Our goal in this case is to learn a mapping from patterns to outputs based on this data set. The patterns could, for example, be the pixel values of pictures of Swiss mountain tops and the corresponding output a discrete label describing its canton. This kind of problem where the outputs are discrete classes is called a *classification* problem. Of course, the outputs could also be real-valued in which case the task is referred to as a *regression* problem. This thesis focuses mostly on classification tasks although many of the techniques described are readily extended to regression problems.

What does it mean to *learn* a mapping from patterns to outputs? One solution could be to make a look-up table out of the training set and predict an arbitrary constant value for patterns not in the training set. While this gives perfect results on the training set, it clearly does not work for other patterns: if an image of the Matterhorn was not in the training set this model is very unlikely to predict that it can be found in the canton of Wallis. What we really want is a model which learns a mapping that performs well not only on the training set but also *generalizes* well on unseen examples. In probabilistic terms, we assume that there is some unknown probability density function $p(\mathbf{x}, \mathbf{t})$ from which all examples are drawn independently and of which the training set is a sample. Learning a mapping then involves extracting that information from the training data which is characteristic for the density $p(\mathbf{x}, \mathbf{t})$ without capturing the noise inherent in the finite training set. The look-up table in our example does exactly the opposite and should have been discarded right away. The other extreme would be a model which is not flexible enough. Imagine that our model makes a simple decision: if there is snow on the mountain top, it is in Wallis. This clearly is too simple a model and will not generalize well either. This illustrates that the learning problem is a kind of balancing act which requires a careful choice of the complexity of the model to obtain good generalization.

Note that, although the above example was given in the context of supervised learning, the problem of complexity control also arises in unsupervised learning. Here the training set D just consists of patterns $\mathbf{x}^i$:

$$D = \{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$$

and is again to be considered as a sample from some unknown probability density function $p(\mathbf{x})$. As said before, a convenient way of expressing unsupervised learning probabilistically is as a density estimation problem, that is, trying to recover the density $p(\mathbf{x})$ which generated the data. A training set consisting of the height of various Swiss mountain[1] tops could, for example, be modeled in a non-parametric way using a histogram. This involves dividing data space in equally sized bins and counting the fraction of training patterns in each bin. The complexity of this model is governed by one single *smoothing* parameter which determines the number of bins: using too few bins the model is poor and with too many bins the model is too flexible and will fit the noise in the training data.

Now that I have defined more or less informally the second part of the title, time has come to turn our attention to what really is the main theme of this thesis: *mixture models* or, to be more precise, how to make complex models from simpler ones and the advantages of such an approach. Along the way, we will encounter various models that have been proposed over the last decade in the fields of neural networks and machine learning. My contribution consists of several extensions of existing models while also tying these models together and developing some connections between them. The rest of this introductory chapter is meant to give the reader a feeling of why this approach of making complex models from simpler ones is an interesting one. I will also recall some of the basics of pattern recognition and statistics without striving for completeness. The reader is referred to standard text books on the subject for a more complete and accurate description (Bishop 1995; Duda and Hart 1973; Ripley 1996) and also to the later chapters in which I will define things more precisely.

A very simple toy example which is still interesting enough to make a point will be used throughout the introduction. Figure 1.1 shows this example which consists of 20 data points in the two-dimensional plane belonging to two classes (the pluses and the crosses). But for the moment we will forget the class labels and have a look at the problem of ...

[1] I will not use the Dutch definition of a mountain which would include bridges and dikes.

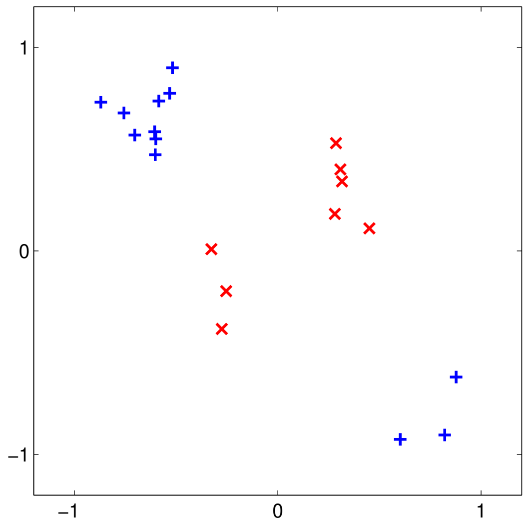


Figure 1.1: Toy example with two classes: $+$ and $\times$.

Unsupervised Learning

Within the variety of tasks arising in unsupervised learning, I will focus on the problem of probability density estimation. Instead of following a non-parametric approach such as using histograms as in the above example, I will stay in the realm of *parametric* methods. This typically involves the choice of a family of probability density functions $p(\mathbf{x}|\boldsymbol{\theta})$ containing adjustable parameters $\boldsymbol{\theta}$. The parameter values are then determined by optimizing some criterion on the training data. A simple and widely used example of a density function is the Gaussian or normal distribution defined on $x \in \mathbb{R}$:

$$p(x|\boldsymbol{\theta}) = \mathcal{N}(x|\mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]$$

with mean μ and variance σ^2 as parameters. A univariate Gaussian distribution is bell-shaped as illustrated in Figure 1.2. This can readily be extended to a multivariate form for $\mathbf{x} \in \mathbb{R}^d$:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{|\boldsymbol{\Sigma}|^{1/2}(2\pi)^{d/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right], \tag{1.1}$$

with mean $\boldsymbol{\mu}$ and $d \times d$ symmetric and positive definite covariance matrix $\boldsymbol{\Sigma}$. For the moment, let us assume that the covariance matrix can be simplified to one single variance parameter as in the one-dimensional case: $\boldsymbol{\Sigma} = \sigma^2\mathbf{I}_d$. This means that a contour of constant density is a hypersphere of all points that have the same distance to the mean $\boldsymbol{\mu}$.

Which criterion should we optimize to determine the values of the parameters $\boldsymbol{\mu}$ and σ^2 on a training set $D = \{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$? The standard choice is to maximize the joint probability density of D:

$$\mathcal{L}(D, \boldsymbol{\theta}) = p(D|\boldsymbol{\theta}) = \prod_{i=1}^{N} p(\mathbf{x}^n|\boldsymbol{\theta}),$$

where as always, we have made the assumption that the data is independently distributed which allows us to factorize the joint probability. This approach seems reasonable since maximizing the

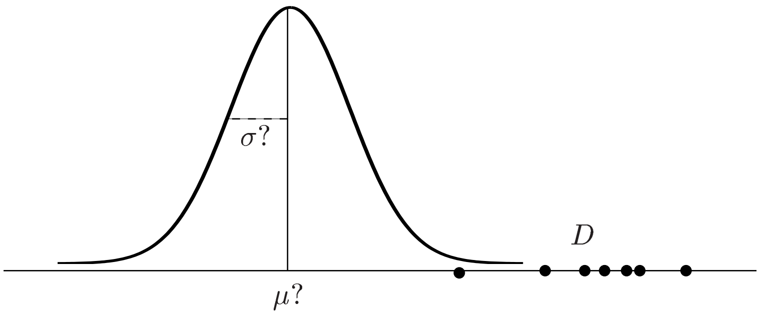


Figure 1.2: Density estimation problem for a one-dimensional normal density function $\mathcal{N}(x|\mu, \sigma^2)$ on data set D.

joint probability can be seen as trying to find that member of the chosen family of density functions which is most likely to have generated the data. This technique is therefore known as *maximum likelihood* and the function $\mathcal{L}$ is called the likelihood function. The idea is illustrated in Figure 1.2 where we suppose that the given data D is more or less Gaussian distributed. The Gaussian bell as given in the figure is of course not likely to have generated this data and maximum likelihood will shift the Gaussian to the right and make it more peaked to explain the data.

But how to find these maximum likelihood estimates? One of the advantages of the Gaussian distribution is that this can be done analytically by differentiating the likelihood function with respect to its parameters. For the one-dimensional case this gives the familiar sample mean and sample variance:

$$\hat{\mu} = \frac{1}{N}\sum_{n=1}^{N} x^n,$$
$$\hat{\sigma}^2 = \frac{1}{N}\sum_{n=1}^{N} (x^n - \hat{\mu})^2.$$

This is nice and simple but of course often the probability density generating the data is far from Gaussian. The data in Figure 1.1 is not likely to come from a simple Gaussian distribution, for example. But it does make sense to suppose that each of the 4 clusters has been generated by a different Gaussian density and that the underlying process is a mix of these 4 components. This can be made more precise by considering mixtures of simple parametric density functions.

Unsupervised Learning: Mixture Models

A *mixture model* is defined as the linear combination of m *component* densities $p_j(\mathbf{x}|\boldsymbol{\theta}_j)$:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{j=1}^{m} \alpha_j p_j(\mathbf{x}|\boldsymbol{\theta}_j), \tag{1.2}$$

with *mixing coefficients* α_j which sum to one and are non-negative to make the mixture model a valid distribution. We can give a generative interpretation of a mixture model as choosing a component j with probability α_j and then generating a data point from $p_j(\mathbf{x}|\boldsymbol{\theta}_j)$. Based on a priori information, we might very well choose component densities which do not belong to the same family of distributions.

We assume that this is not the case in our toy data example and try to model it with a mixture of four spherical (that is, $\boldsymbol{\Sigma} = \sigma^2 \mathbf{I}_d$ as before) Gaussians:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \alpha_1 \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_1, \sigma_1^2) + \alpha_2 \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_2, \sigma_2^2) + \alpha_3 \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_3, \sigma_3^2) + \alpha_4 \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_4, \sigma_4^2).$$

The likelihood function in this case is:

$$p(D|\boldsymbol{\theta}) = \prod_{i=1}^{N} [\alpha_1 \mathcal{N}(\mathbf{x}^i|\boldsymbol{\mu}_1, \sigma_1^2) + \alpha_2 \mathcal{N}(\mathbf{x}^i|\boldsymbol{\mu}_2, \sigma_2^2) + \alpha_3 \mathcal{N}(\mathbf{x}^i|\boldsymbol{\mu}_3, \sigma_3^2) + \alpha_4 \mathcal{N}(\mathbf{x}^i|\boldsymbol{\mu}_4, \sigma_4^2)].$$

Is there also an analytical solution for the maximum likelihood estimates of the mixture parameters as was the case for a single Gaussian distribution? Alas, there is not and even for a mixture of two one-dimensional Gaussians we have to recur to iterative methods (Titterington, Smith, and Makov 1985). One of the first treatments of such a mixture model actually showed that using the method of moments, one needs to find a negative root of a nonic equation (Pearson 1894).

However, an elegant iterative algorithm exists for finding optima of the likelihood function of a mixture model. The idea behind the algorithm is simple and can be illustrated on the toy data. Our problem could be easily solved if for each pattern in our toy data set, we also had a label indicating to which of the four mixture component it belongs. Maximum likelihood estimation can then be done for each component separately for "its" part of the data set. Of course, we do not have such a label but we could cast it into probabilistic terms and define a discrete distribution $P(z|\mathbf{x})$ on the component label $z = 1 \ldots 4$. We can then work with these as a sort of "responsibility" that each component takes for a data point. Parameter estimation can again be done for each component separately but this time on the whole data set with the $P(z|\mathbf{x}^n)$ acting as weighting factor for pattern $\mathbf{x}^n$. The estimate for the means of the Gaussian mixture components becomes, for example:

$$\hat{\boldsymbol{\mu}}_j = \frac{\sum_n P(z=j|\mathbf{x}^n)\mathbf{x}^n}{\sum_n P(z=j|\mathbf{x}^n)},$$

which in the case of one component indeed reduces to the sample mean as one would expect.

This is a particular instantiation for the case of mixture models of a very general algorithm known as the *Expectation-Maximization* algorithm (EM) of Dempster, Laird, and Rubin (1977). This is an iterative two-step procedure for finding optima of a likelihood function and in the case of our Gaussian mixture model it takes the following form:

Initialize coefficients α_j, mean $\boldsymbol{\mu}_j$, and variance σ_j^2

1. Expectation (or E-) step: determine the responsibilities $P(z|\mathbf{x}^n)$ given the current parameter values.
2. Maximization (or M-) step: using the responsibilities find new estimates for the α_j, $\boldsymbol{\mu}_j$, and σ_j^2.

The M-step often reduces to a simple density estimation problem. The working of the EM algorithm is illustrated on our toy example in Figure 1.3. Starting from some random initialization of the four Gaussian component densities, it indeed succeeds in placing the Gaussian circles on the different clusters of data. The example might also give us a better idea of the responsibilities which can be interpreted as how well a density performs on a pattern $\mathbf{x}^n$ relative to the whole mixture model (this is explained in Chapter 2):

$$P(z=j|\mathbf{x}^n) = \frac{\mathcal{N}(\mathbf{x}^n|\boldsymbol{\mu}_j, \sigma_j^2)}{\sum_{i=1}^4 \mathcal{N}(\mathbf{x}^n|\boldsymbol{\mu}_i, \sigma_i^2)}.$$

Let us have a look at the part of Figure 1.3 corresponding to the fifth iteration of the EM algorithm. In this case, two overlapping Gaussians are positioned on the crosses. Clearly, the left one of the two

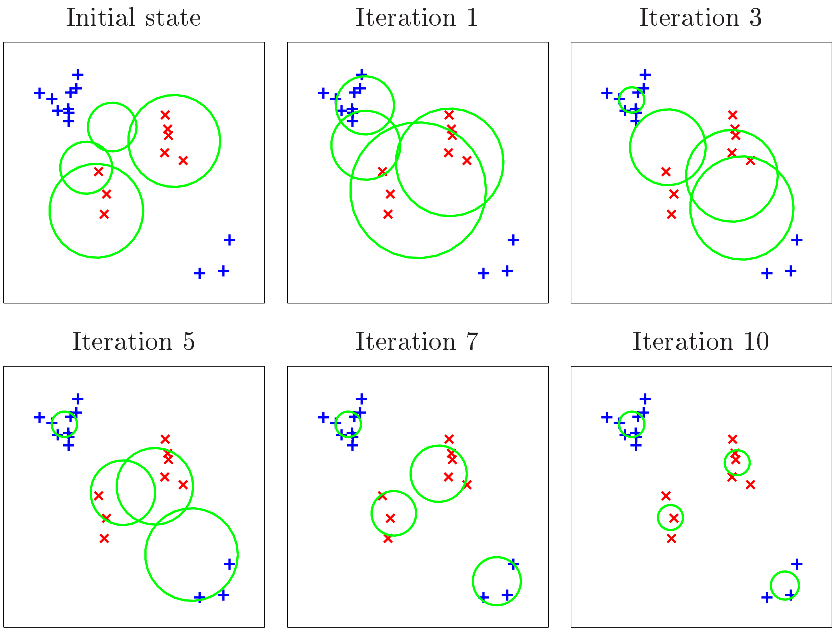


Figure 1.3: Toy example: Gaussian mixture model with 4 components trained in an iterative way with the EM algorithm. The means and variances were initialized randomly while the mixture coefficients were supposed to be equal (1/4). The circles are the contour lines at distance σ_j of the mean of component j.

will take a high responsibility for the cluster of crosses on the left-hand and a low responsibility for all other patterns (E-step). This means that in the new estimate for the mean of this Gaussian the left-hand crosses will be weighted more heavily than the other patterns and that the Gaussian circle will be moved closer to the left-hand crosses (M-step). In the next panel one can see that this is indeed the case and that the tie between the two overlapping Gaussians has been broken. I will deal with the EM algorithm in its general formulation in Chapter 2 and it will be used over and over again throughout this thesis.

It is important to realize that until now, we have focused on finding a good density estimation on the training data. But, as said in the beginning of the introduction, what we are really interested in is the *generalization* on unseen examples. In fact, by increasing the number of mixture components we keep improving the likelihood score on the training data. This is illustrated in Figure 1.4 where the number of mixture components was varied when estimating the density on the toy data problem. In order to estimate the generalization performance of the Gaussian mixture model found, a *test set* was generated from the same underlying distribution. The results clearly show that the likelihood on the training data increases when adding components. On the test set, on the other hand, the optimum number of mixture components is equal to four and with more components the likelihood decreases. The model is said to *overfit* the training data when more than four components are used.

This example illustrates once more the importance of choosing the appropriate model complexity to obtain good generalization. This has motivated much of the research that will be presented in Chapter 2, which deals with variants of Gaussian mixture models that are amenable to a more fine-tuned model selection. It will also be shown how so-called *Bayesian* techniques can improve upon the standard maximum likelihood framework for these mixture models and enable estimation and model

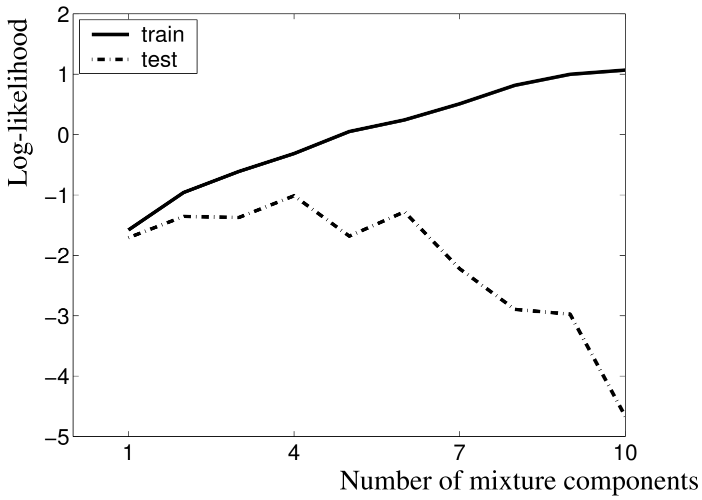


Figure 1.4: Illustrating the importance of the appropriate model complexity for a Gaussian mixture model by varying the number of mixture components. The training set consisted of 20 data points drawn from the same distribution as in Figure 1.1. The test set was made up of 200 data points from this distribution. Results are the mean over 10 different realizations of the training and test sets and the measure used is the log-likelihood $\ln \mathcal{L}(D, \boldsymbol{\theta})$.

selection at the same time.

Supervised Learning: Bayes Classifiers

We now shift our attention to the problem of supervised learning and more specifically classification, where the patterns come with a discrete label. Our toy example is a classification problem with two classes which are denoted as $\mathcal{C}_+$ and $\mathcal{C}_\times$. We want to construct a classifier that predicts whether a given pattern is more likely to be a cross or a plus. Thus, we are interested in modeling the *posterior* probability of each class given a pattern: $P(\mathcal{C}_+|\mathbf{x})$ and $P(\mathcal{C}_\times|\mathbf{x})$. One way to attack this problem is to express it as a density estimation problem which is possible through the famous *Bayes' rule*:

$$P(\mathcal{C}_+|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_+)P(\mathcal{C}_+)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|\mathcal{C}_+)P(\mathcal{C}_+)}{p(\mathbf{x}|\mathcal{C}_+)P(\mathcal{C}_+) + p(\mathbf{x}|\mathcal{C}_\times)P(\mathcal{C}_\times)} = \frac{\text{class-conditional} \times \text{prior}}{\text{normalization}}. \qquad (1.3)$$

The *prior* $P(\mathcal{C}_+)$ represents the probability that a pattern, about which one has no information at all, is a plus. Thus, a sensible way to estimate it is to just count the number of occurrences of each class in the training set. This gives us $P(\mathcal{C}_+) = 3/5$ and $P(\mathcal{C}_\times) = 2/5$ in the toy problem. If we really have no other information about a pattern, the best decision one could make is to attribute it to the class with the highest prior probability.

Of course, normally we do have more information, for example in our case the coordinates (x_1, x_2) of a pattern. This enables the estimation of the *class-conditional* densities $p(\mathbf{x}|\mathcal{C}_+)$ and $p(\mathbf{x}|\mathcal{C}_\times)$ of $\mathbf{x}$, given that it belongs to a certain class. This brings us back to a separate density estimation problem for each class and the unsupervised techniques described previously can be readily employed here. Then Bayes' rule (1.3) can be used to estimate the posterior probabilities and attribute the pattern to the class for which the posterior probability is highest:

$$\begin{array}{lcl}\mathbf{x} \text{ is assigned to } \mathcal{C}_{+} & \Leftrightarrow & P(\mathcal{C}_{+}|\mathbf{x}) > P(\mathcal{C}_{\times}|\mathbf{x}) \\ \mathbf{x} \text{ is assigned to } \mathcal{C}_{+} & \Leftrightarrow & p(\mathbf{x}|\mathcal{C}_{+})P(\mathcal{C}_{+}) > p(\mathbf{x}|\mathcal{C}_{\times})P(\mathcal{C}_{\times}).\end{array}$$

It often is convenient to take the logarithm on both sides and obtain the equivalent (because of the monotonicity of the logarithm) decision rule:

$$\mathbf{x} \text{ is assigned to } \mathcal{C}_{+} \quad \Leftrightarrow \quad \ln p(\mathbf{x}|\mathcal{C}_{+}) + \ln P(\mathcal{C}_{+}) > \ln p(\mathbf{x}|\mathcal{C}_{\times}) + \ln P(\mathcal{C}_{\times}).$$

In our toy example, the prior probabilities of the two classes are assumed to be equal:[2]

$$\mathbf{x} \text{ is assigned to } \mathcal{C}_{+} \quad \Leftrightarrow \quad \ln p(\mathbf{x}|\mathcal{C}_{+}) > \ln p(\mathbf{x}|\mathcal{C}_{\times}).$$

Such a decision rule gives rise to so-called *decision boundaries* determined by the points at which the inequality becomes an equality.

Let us now try to model each class in our two-dimensional toy problem with a spherical Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_{+}, \sigma_{+}^2)$ and $\mathcal{N}(\boldsymbol{\mu}_{\times}, \sigma_{\times}^2)$ using (1.1) with $\boldsymbol{\Sigma}_j = \sigma_j^2 \mathbf{I}_2$ and $d=2$:

$$\begin{array}{rcl}\ln p(\mathbf{x}|\mathcal{C}_{+}) & = & \ln \frac{1}{2\pi} + \ln \frac{1}{\sigma_{+}^2} - \frac{(\mathbf{x}-\boldsymbol{\mu}_{+})^T(\mathbf{x}-\boldsymbol{\mu}_{+})}{2\sigma_{+}^2} \\ \ln p(\mathbf{x}|\mathcal{C}_{\times}) & = & \ln \frac{1}{2\pi} + \ln \frac{1}{\sigma_{\times}^2} - \frac{(\mathbf{x}-\boldsymbol{\mu}_{\times})^T(\mathbf{x}-\boldsymbol{\mu}_{\times})}{2\sigma_{\times}^2}.\end{array}$$

Subtracting the second line from the first one gives the following equation for the decision boundary between the pluses and the crosses:

$$0 = \ln \frac{\sigma_{\times}^2}{\sigma_{+}^2} + \frac{(\mathbf{x}-\boldsymbol{\mu}_{\times})^T(\mathbf{x}-\boldsymbol{\mu}_{\times})}{2\sigma_{\times}^2} - \frac{(\mathbf{x}-\boldsymbol{\mu}_{+})^T(\mathbf{x}-\boldsymbol{\mu}_{+})}{2\sigma_{+}^2} \tag{1.4}$$

and it is easy (but rather boring) to show that the resulting decision boundary is a circle.

This is illustrated in the upper part of Figure 1.5. It is clear that the idea to model each class-conditional density with only one spherical Gaussian is not that successful. Not only does the resulting classifier make an error on one of the crosses but also the error on a test set is likely to be high, given the tight boundary around the crosses. The lower part of the figure shows that when using a mixture of two spherical Gaussians for each class, the result looks much more reasonable. Also in the case of Bayes classifiers, mixture models can lead to improved results. The mixture models defined in Chapter 2 will be used to construct Bayes classifiers in Chapter 4.

Let us now have a look at the special case in which we also assume the variances of the two Gaussian distributions $\mathcal{N}(\boldsymbol{\mu}_{+}, \sigma_{+}^2)$ and $\mathcal{N}(\boldsymbol{\mu}_{\times}, \sigma_{\times}^2)$ to be equal: $\sigma_{+}^2 = \sigma_{\times}^2$. Then (1.4) can be further simplified to a *linear* function of $\mathbf{x}$:

$$(\boldsymbol{\mu}_{+}^T - \boldsymbol{\mu}_{\times}^T)\mathbf{x} + 2(\boldsymbol{\mu}_{\times}^T\boldsymbol{\mu}_{\times} - \boldsymbol{\mu}_{+}^T\boldsymbol{\mu}_{+}) := \mathbf{w}^T\mathbf{x} + b.$$

This has motivated direct estimation of the parameters $(\mathbf{w}, b)$ of such a linear decision boundary both in statistics (Fisher 1952) and pattern recognition (Nilsson 1965). This approach has the advantage of not having to choose a specific parametric density, in this sense it might be considered more robust. It also has the advantage that the direct approach often requires far less parameters (only d+1). However,

[2] I can assure the reader that, in fact, they are, even if the small training set does not show this.

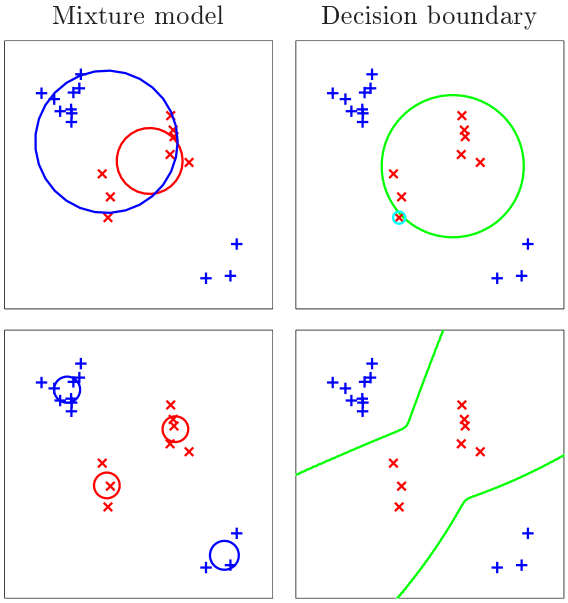


Figure 1.5: Toy example with two classes (+ and ×): Bayes classifier. The top part shows a Bayes classifier with a spherical Gaussian for each class (left) and the resulting circular decision boundary (right) leading to one error (indicated by a circle around the point) on the toy problem. The bottom part uses a mixture of two spherical Gaussians for each class (left) and nicely separates the two classes (right).

it is not a panacea: a linear decision boundary is obviously very limited. In our toy example, it would not be of much use in separating the two classes.

A logical way to go is to extend these linear models to *non-linear* ones and this has led to a variety of new models in the neural network field (multi-layer perceptrons) and statistics (projection pursuit regression and multivariate adaptive regression splines, for example). These models all have the following general form:

$$y(\mathbf{x}) = \mathbf{w}_2^T g(\mathbf{w}_1^T \mathbf{x} + b_1) + b_2, \tag{1.5}$$

where g is some non-linear function; it is often preceded by yet another non-linear function. While these models are far more expressive, they also call for rather complex non-linear optimization routines. In the rest of this introduction, I will present three alternatives which are all based on the idea of making complex models out of a simple linear classifier.

Supervised Learning: Kernel Trick

In the first alternative, we take one step back from the general non-linear form (1.5) and consider the following model with only one "layer" of parameters:

$$y(\mathbf{x}) = \mathbf{w}_2^T \Phi(\mathbf{x}) + b_2,$$

where $\Phi : \mathbb{R}^d \to F$ is a function from data space into some higher-dimensional *feature* space F. The roots of this approach lie in the sixties and at that time it was known as the method of potential

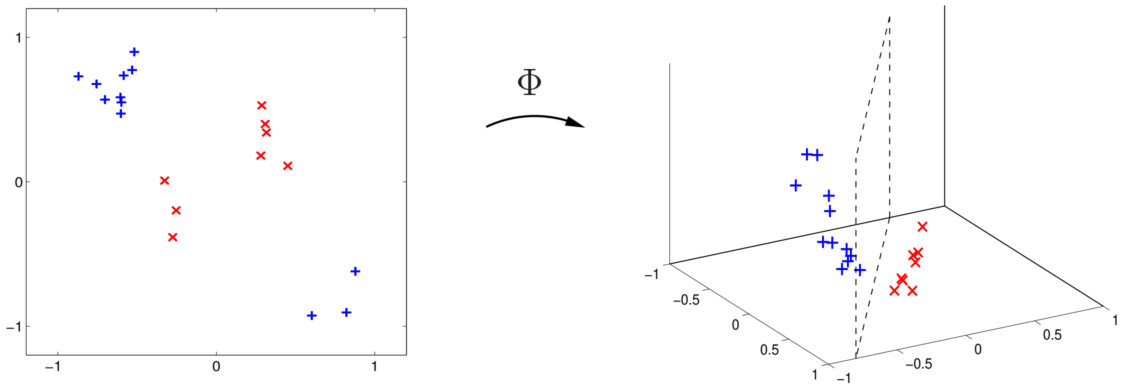


Figure 1.6: Toy example with two classes ($+$ and $\times$) with a mapping Φ of the data from $\mathbb{R}^2$ into $\mathbb{R}^3$. The mapped data is linearly separable in $\mathbb{R}^3$.

functions in the field of pattern recognition (Aizerman, Braverman, and Rozonoer 1964; Nilsson 1965). The idea behind it is that by choosing a suitable mapping Φ into feature space, one might obtain a high-dimensional representation of the data which is better suited for a simple linear model. Consider, for example, the following mapping to the unordered products of degree 2 of the input coordinates:

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3$$
$$(x_1, x_2) \to (y_1, y_2, y_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2).$$

When applied to our toy example, the mapped data in three-dimensional space can be correctly classified by a linear decision surface (see Figure 1.6).

But this idea of a mapping into a high-dimensional space can become expensive: taking all unordered degree p products of the coordinates $(x_1, \ldots, x_d)$ leads to a feature space of dimension C_p^{d+p-1}. Consider again the grey-level images of Swiss mountain tops and let us assume that they are of size 16×16 ($d = 256$). When taking all unordered products of 3 pixels ($p = 3$) the dimension of feature space becomes 2829056 ... This mere fact might persuade us that this is not the way to go. However, let us have a closer look at the mapping $\Phi : \mathbb{R}^2 \to \mathbb{R}^3$ and more specifically at dot products in $\mathbb{R}^3$:

$$\begin{aligned}\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}) &= (x_1^2, \sqrt{2}x_1x_2, x_2^2) \cdot (y_1^2, \sqrt{2}y_1y_2, y_2^2) \\ &= x_1^2y_1^2 + 2x_1x_2 + x_2^2y_2^2 = ((x_1, x_2) \cdot (y_1, y_2))^2 \\ &= (\mathbf{x} \cdot \mathbf{y})^2 := k(\mathbf{x}, \mathbf{y}).\end{aligned}$$

Thus, taking the dot product in feature space can be done without explicitly mapping the data but via a so-called *kernel* function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. There are actually many classes of kernel functions which correspond to a dot product in some feature space.

An important consequence of the above is that any algorithm that only depends on the data through dot products $\mathbf{x}^i \cdot \mathbf{x}^j$ can be made non-linear by replacing dot products by $k(\mathbf{x}^i, \mathbf{x}^j)$. This is the *kernel trick* made popular by Vapnik and his colleagues (Vapnik 1995). It is one of the several elegant ingredients which make up *support vector machines* and turned them into powerful non-linear classifiers.

Actually, I will apply the kernel trick in an unsupervised learning context, viz. for extracting features from the data in a non-linear way. This is based on a recent extension of the well-known method of principal component analysis using kernel functions (Schölkopf, Smola, and Müller 1998).

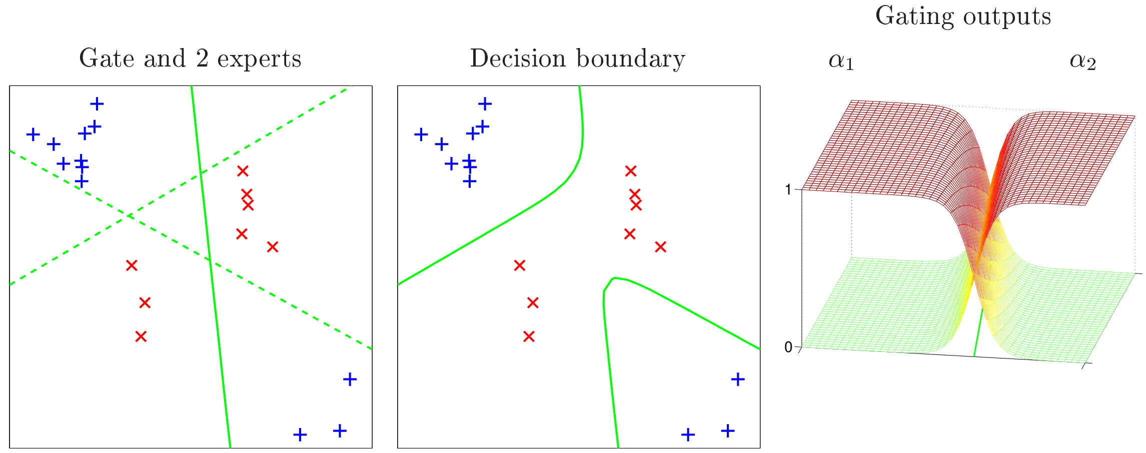


Figure 1.7: Toy example with two classes (+ and ×): a mixture of two experts. The left panel shows the problem decomposition found with the (solid) decision line of the gate which creates a simple subproblem for each of the two experts, decision lines of the experts are dashed. The middle panel shows the decision boundary of the entire mixture of experts. The right panel gives the outputs of the gate and illustrates the softness of the split.

This is yet another part of Chapter 2 and surprisingly enough will be based on some of the techniques described in the mixture model part of Chapter 2.

Supervised Learning: Mixture of Experts

The second alternative is inspired by the use of mixture models in unsupervised learning and extends them to supervised learning. The motivation for this approach comes from the fact that often complex problems can be solved by breaking them down into, hopefully simpler, sub-problems and combining their responses. This idea is at the heart of the *mixture of experts* model (Jacobs et al. 1991) in which a *gating network* splits up the data space with *expert networks*[3] specializing on the different regions. More formally, the architecture of a mixture of experts consists of m linear experts, the outputs $y_j(\mathbf{x}, \mathbf{w}_j)$ of which are weighted by the outputs of a linear gating network $\alpha_j(\mathbf{x}, \mathbf{V})$:

$$y(\mathbf{x}, \boldsymbol{\theta}) = \sum_{j=1}^{m} \alpha_j(\mathbf{x}, \mathbf{V}) y_j(\mathbf{x}, \mathbf{w}_j).$$

The outputs of the *gating network* are constrained to sum to one and be non-negative. This allows for its interpretation as a classifier which attributes patterns to the expert networks in a probabilistic way. This also implies that the gating network splits the data space in a "soft" way: data can lie in several regions simultaneously. This is an important difference with so-called decision trees that tend to perform hard splits.

Let us come back to the toy example when modeled by a mixture of 2 linear experts. The left panel of Figure 1.7 shows the problem decomposition found by the gating network. The two sub-problems have become linearly separable and each of them can be solved by a simple linear expert model which only gives a correct response on "its" part of data space. The decision boundary of the

[3] The use of the term "expert" is a little overblown since it need not be an expert at all!

entire model, that is a linear combination of the expert outputs weighted by the gating outputs, does indeed separate the pluses from the crosses (second panel of Figure 1.7). The third panel illustrates that the gating network splits the data space in a soft way: the two outputs of the gating network $\alpha_1(\mathbf{x})$ and $\alpha_2(\mathbf{x})$ are equal to a 1/2 on its decision line and away from it one expert dominates.

The mixture of experts model can be given a probabilistic interpretation as a *conditional* density estimation problem. Given a training set $D = \{(\mathbf{x}^1, \mathbf{t}^1), \ldots, (\mathbf{x}^N, \mathbf{t}^N)\}$ we want to estimate a conditional density $p(\mathbf{t}|\mathbf{x})$ and a conditional mixture model can be defined as follows:

$$p(\mathbf{t}|\mathbf{x}) = \sum_{j=1}^{m} \alpha_j(\mathbf{x}) p_j(\mathbf{t}|\mathbf{x}).$$

When comparing this with the definition of a standard mixture model (1.2), this is indeed a mixture in output space conditioned on pattern $\mathbf{x}$. Roughly we can say that expert network j is modeling the mean of density p_j and the gating network a multinomial $\boldsymbol{\alpha}$. A mixture of experts being a conditional mixture model, it should not come as a surprise that it can be trained within the maximum likelihood framework by the EM algorithm.

Chapter 4 will provide a link between the unsupervised mixture models of Chapter 2 and mixtures of experts by using these unsupervised mixtures as gating network.

Supervised Learning: Boosting

A third way of making a complex model out of simpler ones is with so-called committee or *ensemble methods*. I will limit myself to one of the most promising ensemble methods: *boosting*. The theoretical motivation of boosting came from the wish to transform a model which performs only slightly better than random guessing into an arbitrarily accurate learning algorithm (Freund 1995; Schapire 1990). The first practical boosting method for classification problems is known as *AdaBoost* and was proposed in 1995 by Freund and Schapire. The idea of AdaBoost is to construct in an iterative fashion a linear combination of m simple *base* classifiers:

$$y(\mathbf{x}) = \sum_{j=1}^{m} \alpha_j y_j(\mathbf{x}),$$

adding classifiers one by one during training. When comparing this with the definition of a mixture of experts, one immediately sees that there is no such thing as a gating network here. The sub-problems in this case are determined by the performance of the preceding classifiers and *modifying* the training set: more and more weight is put on the examples which have been misclassified by the previous classifiers. This modification of the training set can be done by interpreting these example weights as probabilities and by sampling according to this distribution. Often a deterministic interpretation is also possible in which the example weights are included as factors into some cost function to be optimized.

This can again be illustrated on our toy example[4] for which an ensemble of simple linear classifiers has been constructed (Figure 1.8). The first classifier correctly classifies all patterns except the lowest plus pattern. The reweighting idea of boosting implies that this pattern takes on a higher weight in the training set for the subsequent classifier. This is indicated in the second panel by scaling the size of a symbol proportionally to its weight. The second classifier, therefore, mainly focuses on correctly classifying this lowest plus pattern. The third classifier correctly classifies the patterns which were misclassified by either of the two previous classifiers. When combining these three classifiers, the resulting ensemble does separate the pluses from the crosses (the lower part of Figure 1.8).

[4] With only 8 patterns this time to make the plot more readable.

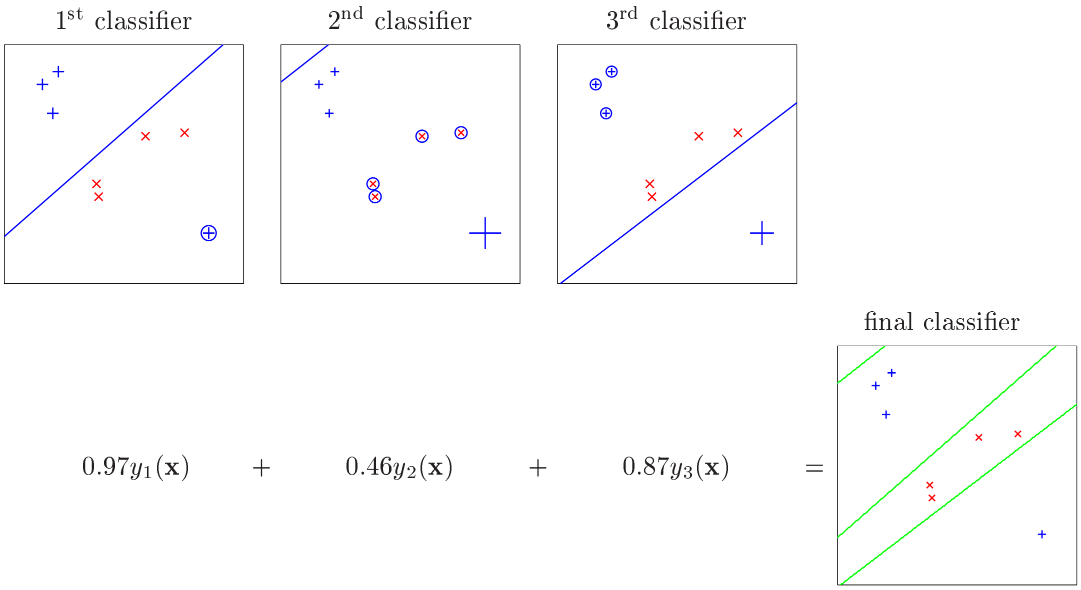


Figure 1.8: Toy example with two classes ($+$ and $\times$): boosted ensemble of three classifiers. The upper part shows the three classifiers, that have been generated iteratively by boosting, with their decision lines. The errors made by each classifier are indicated by circles around the data point. The weight given to each data point is indicated by the size of the pluses and the crosses. The lower part shows the combination of the three boosted classifiers together with its decision boundary.

One might think that the boosting principle of constructing classifiers on ever more specialized versions of the training set leads to overfitting and poor generalization. While this can indeed be the case on very noisy data sets, empirical evidence and results from statistical learning theory have shown that boosting performs well in a low noise regime. I will come back to this in greater detail in Chapter 5. There, I will also develop a model in between a boosted ensemble and a mixture of experts by allowing the fixed weights α_i of AdaBoost to be input-dependent.

Roadmap

Now that I have presented the main characters of the piece, time has come to rise the curtain. The general flow goes from unsupervised to supervised learning.

Chapter 2 focuses on the use of mixture models for density estimation. Starting with a general description of the EM algorithm, it is then applied in the specific case of Gaussian mixture models (GMMs). It is outlined that there is a large gap between the use of standard multivariate Gaussians and restricted ones such as the spherical Gaussians used throughout the Introduction. A more flexible alternative is presented in the form of the recently proposed mixtures of latent variable models (Tipping and Bishop 1999). These latent variable models are strongly related to dimensionality reduction methods as factor analysis and principal component analysis and these are discussed in detail. An EM algorithm for mixtures of factor analyzers and principal component analyzers is derived and its computational complexity is examined. After these theoretical preliminaries, the results of an empirical comparison of the various mixture models on about 20 data sets are discussed. The empirical results show that mixtures of latent variable models are often preferable both in terms of computational complexity and generalization performance to standard GMMs.

One of the problems with mixtures of latent variable models is the selection of the appropriate model. One not only has to choose the number of mixture components but also the dimension of latent space in each component. A search over all combinations of these is nearly impossible. A new Bayesian technique is presented that finds the appropriate dimension of latent space for each component of a mixture of factor analyzers. A recent Bayesian treatment of PCA (Bishop 1999a) is derived as a special case. Empirical results illustrate that this not only simplifies model selection but can also lead to improved results.

The last part of Chapter 2 deals with another form of unsupervised learning, viz. feature extraction. A recent method for non-linear feature extraction called kernel principal component analysis (PCA) (Schölkopf, Smola, and Müller 1998) is presented. Standard kernel PCA on a training set with N examples requires storage of an $N \times N$ matrix. It is shown how an EM algorithm for PCA can avoid storing this matrix and make kernel PCA applicable to large data sets with more than 10,000 examples. The extracted non-linear features are shown to be interesting by training simple linear classifiers on them that perform very well. A novel on-line EM algorithm for PCA is derived and shown to give further speed-ups.

Chapter 3 is mainly of tutorial nature. After a short recall of the principles of Bayes classifiers, the mixture of experts model is defined in detail. An overview of various methods for training mixtures of experts is given going from gradient-based approaches to the EM algorithm. The chapter ends with a proof of a new consistency result which shows that in the limit of infinite data the posterior class probabilities minimize the error function of a mixture of experts.

Chapter 4 links the models of the previous two chapters by using mixtures of latent variable models as a gating network in a mixture of experts. This extends work of Xu, Jordan, and Hinton (1995) and it is shown that the resulting localized mixture of experts can be trained in the EM framework. These localized mixtures of experts are compared with standard mixtures of experts in an empirical evaluation on about 20 data sets. Some problems of the localized approach are outlined. Furthermore, Bayes classifiers are constructed from the mixture models described in Chapter 2 on these same data sets and shown to perform well also in supervised learning.

Chapter 5 describes the idea behind boosting and outlines its recent interpretation as a stage-wise gradient descent optimization of a cost function of the margins. A new algorithm is presented which extends AdaBoost by using a separate model for determining the input-dependent coefficients of each expert. Experiments show that one can often obtain performance that is at least as good as with AdaBoost but with a much smaller ensemble.

Chapter 6 concludes with a series of experiments on two computer vision problems, viz. face versus non-face classification and handwritten digit recognition. Some of the models encountered along this thesis are shown to give results which rank well among those obtained with other state-of-the-art methods.

Before starting out, I would like the reader to know that these pages contain some rather detailed and technical proofs. Since I do not like "exercises left to the reader" or saying "it is straightforward"[5] when it is not, I tried to spell out proofs as completely as possible. If you think the proofs too detailed, it will not harm to read them faster!

[5] I did put in a few of them for old habit's sake.

CHAPTER 2

Unsupervised Learning: Mixture Models and Feature Extraction

Unsupervised learning deals with modeling or extracting information from an unlabeled data sample:

$$D = \{\mathbf{x}^1, \ldots, \mathbf{x}^N\},$$

with $\mathbf{x}^n = (x_1^n, \ldots, x_d^n)^T \in \mathbb{R}^d$. Two classic problems in unsupervised learning are density modeling and feature extraction. The goal of density estimation is to find a descriptive model of the data. As we already saw in Chapter 1, this is often done by assuming that the data has been generated according to some underlying parametric density function and by estimating the parameters of this density. Descriptive models can, for example, be used for clustering data in groups of similar examples or as an initial step in supervised learning of labeled data such as Bayes classifiers and radial basis function networks.

Feature extraction tries to find a compact description of the interesting features of the data. This can be useful for visualization of high-dimensional data in two or three dimensions or for data compression. It can also be applied as a pre-processing step in supervised learning which enables to reduce the dimension of the data to be handled by a subsequent model.

Roadmap

In this chapter, some interesting examples of both the descriptive and the feature extraction approach to unsupervised learning are developed. Along the way, we will see that the boundary between these two approaches is not that rigid and that a cross-over can be fruitful. We start with a general and precise description of the Expectation-Maximization algorithm for maximum likelihood (ML) in the presence of hidden or latent variables. This forms the basis of most algorithms described in this and later chapters. As a first example of latent variable models, we consider mixture models which we already encountered in Chapter 1. In this case, the single latent variable is just a discrete label for the mixture components. Section 2.2 shows how to apply the EM algorithm to general mixture models and to Gaussian mixture models in particular. It is outlined that the issue of model complexity can only be handled in a very coarse-grained way in GMMs. An elegant manner of controlling model complexity of GMMs in a more flexible way is the introduction of continuous latent variables $\mathbf{z} = (z_1, \ldots, z_\ell)^T$ with $\ell < d$; this can be interpreted as a form of dimensionality reduction or feature extraction. Section 2.3 describes how this idea forms the basis of factor analysis (FA) (Bartholomew and Knott 1999) and a recent probabilistic formulation of PCA (Tipping and Bishop 1999). These linear latent variable models can be readily included as component distributions of a mixture model. Section 2.4 describes how the EM algorithm can be used for learning the parameters of a mixture of factor analyzers (MFA) (Ghahramani and Hinton 1996). An EM algorithm for mixtures of principal component analyzers (MPCA) (Tipping and Bishop 1999) is derived as a special case of the one for MFAs. A detailed analysis of the computational complexity of these EM algorithms is given and it is shown that also in this respect they are a valuable alternative to basic GMMs. I conclude this review of the state-of-the-art with a short derivation of an EM algorithm for standard PCA as proposed by Roweis (1998). It is shown that the algorithm is an alternative to standard numerical methods for PCA when dealing with high-dimensional data. Moreover, the algorithm can handle the data in an incremental way which is useful on large data sets (section 2.5).

The first original contribution of this chapter is empirical, viz. an experimental comparison of mixtures of latent variable models and GMMs on about 20 data sets. The empirical density estimation results show that mixtures of latent variable models are often preferable both in terms of computational complexity and generalization performance to standard GMMs (Moerland 1999b). We will then address the issue of model selection in more detail. The experiments of section 2.6 used validation data to select values for free parameters such as the number of mixture components and the dimension of latent space ℓ. An approximate Bayesian inference technique is described that automatically selects the appropriate dimension of latent space for factor analysis. A recent Bayesian treatment of PCA (Bishop 1999a) is derived as a special case. Empirical results on about 20 data sets illustrate that the Bayesian technique not only simplifies model selection but can also lead to improved results (section 2.7).

The last part of this chapter illustrates the usefulness of the EM algorithm for standard PCA presented in section 2.5 in a particular context, viz. that of kernel PCA (Schölkopf, Smola, and Müller 1998). This is a non-linear extension of PCA based on the kernel trick described in Chapter 1. It requires the eigendecomposition of a so-called kernel matrix of size $N \times N$. A disadvantage of using standard numerical methods for this eigendecomposition is that they require storing this kernel matrix. The incremental version of EM for PCA is used to avoid storage of the kernel matrix. The application of EM to kernel PCA necessitates a reformulation of kernel PCA which I developed independently of Mika (1998). Experimental results are given where EM for kernel PCA extracts up to 512 non-linear features from a data set with 15,000 examples. The extracted features of various data sets are also used as pre-processed data for a simple linear classifier. The performance of the resulting classifiers turns out to be very good. Finally, a novel on-line EM algorithm for PCA is derived based on a general approach of Neal and Hinton (1999). Experiments show that this can give a further speed-up of EM for kernel PCA.

2.1 Maximum Likelihood Estimation

Suppose that we have unlabeled data $D = \{\mathbf{x}^n\}$ which we assume to be generated from a probability density function $p(\mathbf{x}|\boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$. As we saw in Chapter 1, a common approach to estimate the parameters of $p(\mathbf{x}|\boldsymbol{\theta})$ given a set of N examples $\{\mathbf{x}^n\}$ is *maximum likelihood* (for example, Duda and Hart 1973) and this is the approach I will take in most of this thesis. We make the standard assumption that the examples are drawn independently from the same distribution which allows us to factorize the joint probability $p(\{\mathbf{x}^n\}|\boldsymbol{\theta})$:

$$\mathcal{L}(\boldsymbol{\theta}) = p(\{\mathbf{x}^n\}|\boldsymbol{\theta}) = \prod_n p(\mathbf{x}^n|\boldsymbol{\theta}). \tag{2.1}$$

This function is defined as the *likelihood* of $\boldsymbol{\theta}$ with respect to the data $\{\mathbf{x}^n\}$. Maximum likelihood estimation consists of finding values for $\boldsymbol{\theta}$ which maximize $\mathcal{L}(\boldsymbol{\theta})$. This is intuitively appealing since it corresponds to values which describe the data well. An alternative approach would be to consider not only the single most likely parameter values but to treat them as random variables, which is the so-called Bayesian approach. We will come back to this and its possible advantages in section 2.7.

It often is easier to maximize the *log-likelihood* (which has identical solutions due to the monotonicity of the logarithm):

$$\ln \mathcal{L}(\boldsymbol{\theta}) = \sum_n \ln p(\mathbf{x}^n|\boldsymbol{\theta}). \tag{2.2}$$

If one wants to interpret it as an *error function* to be minimized, we take the negative log-likelihood:

$$E(\boldsymbol{\theta}) = -\sum_n \ln p(\mathbf{x}^n|\boldsymbol{\theta}). \tag{2.3}$$

A more general approach is to assume that we also have unobserved or *hidden* variables $\mathbf{z}$ that help modeling the observed data $\mathbf{x}$. The hidden variables can, for example, be discrete component labels which represent a sort of imaginary class labels for the observed data. In fact, this is a way to define mixture models, as we will see in section 2.2. The log-likelihood of the observed data is then obtained by marginalizing over the hidden variables. For the moment, we assume the hidden variables to be discrete and marginalization boils down to applying the sum rule (A.16):

$$\ln \mathcal{L}(\boldsymbol{\theta}) = \sum_n \ln \sum_{\mathbf{z}} p(\mathbf{x}^n, \mathbf{z}|\boldsymbol{\theta}). \tag{2.4}$$

A first and simple approach to optimize this log-likelihood would be to calculate the partial derivatives with respect to each of its parameters θ_j and see whether the resulting equations (put to zero) can be solved. Starting with (2.4) and taking the derivative of the logarithm gives:

$$\frac{\partial \ln \mathcal{L}(\boldsymbol{\theta})}{\partial \theta_j} = \sum_n \frac{1}{\sum_{\mathbf{z}} p(\mathbf{x}^n, \mathbf{z}|\boldsymbol{\theta})} \frac{\partial}{\partial \theta_j} \Big[\sum_{\mathbf{z}} p(\mathbf{x}^n, \mathbf{z}|\boldsymbol{\theta}) \Big].$$

Using the sum rule (A.16) and shifting the sum:

$$= \sum_{n,\mathbf{z}} \frac{1}{p(\mathbf{x}^n|\boldsymbol{\theta})} \frac{\partial}{\partial \theta_j} p(\mathbf{x}^n, \mathbf{z}|\boldsymbol{\theta}).$$

Introducing the logarithm via the partial derivative:

$$= \sum_{n,\mathbf{z}} \frac{p(\mathbf{x}^n, \mathbf{z}|\boldsymbol{\theta})}{p(\mathbf{x}^n|\boldsymbol{\theta})} \frac{\partial}{\partial \theta_j} \ln p(\mathbf{x}^n, \mathbf{z}|\boldsymbol{\theta})$$

which can be rewritten with Bayes' rule (A.18):

$$= \sum_{n,\mathbf{z}} p(\mathbf{z}|\mathbf{x}^n, \boldsymbol{\theta}) \frac{\partial}{\partial \theta_j} \ln p(\mathbf{x}^n, \mathbf{z}|\boldsymbol{\theta}) = 0. \tag{2.5}$$

This system of equations is in all interesting cases coupled and non-linear because the parameters $\boldsymbol{\theta}$ appear both in the *posterior* of the hidden variables $p(\mathbf{z}|\mathbf{x}^n, \boldsymbol{\theta})$ and the partial derivatives. This means that we cannot hope for a direct closed-form solution of the system. Of course, it could be handled iteratively by any of the standard non-linear optimization techniques but in what follows a simpler approach is derived in the form of the EM algorithm. The basic idea of the EM algorithm is to decouple the system of equations (2.5) by first estimating the posterior $p(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta})$ and then determine the parameter values in (2.5) while keeping the posterior fixed. This is generally a simpler problem since it involves the *joint* distribution $p(\mathbf{x}^n,\mathbf{z}|\boldsymbol{\theta})$ of the observed and hidden variables. It will be shown that this intuitively appealing approach is a theoretically sound way of performing maximum likelihood estimation.

Actually, we consider a more general problem based on the observation that in (2.5) the summation over all configurations of the hidden variables might be computationally intractable. Imagine, for example, a binary hidden state of length 50: marginalizing the hidden variables would require summing over all 2^{50} hidden states; this is actually a problem which occurs in Boltzmann machines and sigmoid belief networks (Jordan et al. 1999). It can sometimes be remedied by approximating the posterior of the hidden variables $p(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta})$ by a restricted distribution $Q(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta})$. We therefore start out with the log-likelihood (2.2):

$$\ln \mathcal{L}(\boldsymbol{\theta}) = \sum_n \ln p(\mathbf{x}^n|\boldsymbol{\theta})$$

and introduce the approximating distribution Q with the sum rule (A.16):

$$= \sum_n \ln p(\mathbf{x}^n|\boldsymbol{\theta}) \sum_{\mathbf{z}} Q(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta}).$$

This can be rewritten with the product rule (A.17) and Bayes' rule (A.18):

$$\begin{aligned}
&= \sum_{n,\mathbf{z}} \left[\ln \frac{p(\mathbf{x}^n,\mathbf{z}|\boldsymbol{\theta})}{p(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta})}\right] Q(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta}) \\
&= \sum_{n,\mathbf{z}} Q(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta}) \ln p(\mathbf{x}^n,\mathbf{z}|\boldsymbol{\theta}) - \sum_{n,\mathbf{z}} Q(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta}) \ln p(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta}).
\end{aligned}$$

Introducing the entropy of Q in both terms:

$$\begin{aligned}
&= \sum_{n,\mathbf{z}} Q(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta}) \ln p(\mathbf{x}^n,\mathbf{z}|\boldsymbol{\theta}) - \sum_{n,\mathbf{z}} Q(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta}) \ln Q(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta}) - \sum_{n,\mathbf{z}} Q(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta}) p(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta}) \\
&\quad + \sum_{n,\mathbf{z}} Q(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta}) \ln Q(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta}) \\
&= \sum_{n,\mathbf{z}} Q(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta}) \ln \frac{p(\mathbf{x}^n,\mathbf{z}|\boldsymbol{\theta})}{Q(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta})} - \sum_{n,\mathbf{z}} Q(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta}) \ln \frac{p(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta})}{Q(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta})}.
\end{aligned}$$

Thus, the log-likelihood can be rewritten for any Q as:

$$\begin{aligned}
\ln \mathcal{L}(\boldsymbol{\theta}) &= \sum_{n,\mathbf{z}} Q(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta}) \ln \frac{p(\mathbf{x}^n,\mathbf{z}|\boldsymbol{\theta})}{Q(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta})} - \sum_{n,\mathbf{z}} Q(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta}) \ln \frac{p(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta})}{Q(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta})} \qquad (2.6) \\
&= \mathcal{L}(Q,\boldsymbol{\theta}) + \mathrm{KL}(Q||p) \\
&= \text{free energy} \quad + \quad \text{Kullback-Leibler divergence.}
\end{aligned}$$

This rewriting of the log-likelihood stems from the work of Neal and Hinton (1999). Knowing that the Kullback-Leibler divergence is non-negative (Cover and Thomas 1991) directly leads to the observation that the free energy $\mathcal{L}(Q,\boldsymbol{\theta})$ is a lower bound of the log-likelihood. This forms the basis of the *generalized* EM algorithm by coordinate ascent in the lower bound $\mathcal{L}(Q,\boldsymbol{\theta})$ (Algorithm 1). It is

Algorithm 1 Generalized EM algorithm

loop

{ (2.6): $\ln \mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}(Q, \boldsymbol{\theta}) + \mathrm{KL}(Q||p)$}

E-step: Choose Q^{new} such that it decreases the Kullback-Leibler divergence $\mathrm{KL}(Q||p)$ while keeping the parameters $\boldsymbol{\theta}$ fixed

{$\mathcal{L}(Q^{\text{new}}, \boldsymbol{\theta}) \geq \mathcal{L}(Q, \boldsymbol{\theta})$ since $\ln \mathcal{L}(\boldsymbol{\theta})$ does not depend on Q}

M-step: Choose the parameters $\boldsymbol{\theta}^{\text{new}}$ such as to increase the free energy $\mathcal{L}(Q^{\text{new}}, \boldsymbol{\theta})$ while keeping Q^{new} fixed

{$\mathcal{L}(Q^{\text{new}}, \boldsymbol{\theta}^{\text{new}}) \geq \mathcal{L}(Q, \boldsymbol{\theta})$}

$Q, \boldsymbol{\theta} := Q^{\text{new}}, \boldsymbol{\theta}^{\text{new}}$

end loop

Algorithm 2 EM algorithm

loop

{ (2.6): $\ln \mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}(Q, \boldsymbol{\theta}) + \mathrm{KL}(Q||p)$}

E-step: $Q^{\text{new}}(\mathbf{z}|\mathbf{x}^n, \boldsymbol{\theta}) := p(\mathbf{z}|\mathbf{x}^n, \boldsymbol{\theta})$ (for all n)

{$\mathcal{L}(Q^{\text{new}}, \boldsymbol{\theta}) = \ln \mathcal{L}(\boldsymbol{\theta})$ since $\ln \mathcal{L}(\boldsymbol{\theta})$ does not depend on Q and $\mathrm{KL}(Q^{\text{new}}||p) = 0$}

M-step: Choose the parameters $\boldsymbol{\theta}^{\text{new}}$ such as to maximize the free energy $\mathcal{L}(Q^{\text{new}}, \boldsymbol{\theta})$ while keeping Q^{new} fixed.

{$\ln \mathcal{L}(\boldsymbol{\theta}^{\text{new}}) \geq \ln \mathcal{L}(Q^{\text{new}}, \boldsymbol{\theta}^{\text{new}}) \geq \ln \mathcal{L}(\boldsymbol{\theta})$}

$Q, \boldsymbol{\theta} := Q^{\text{new}}, \boldsymbol{\theta}^{\text{new}}$

end loop

an iterative two-step procedure consisting of a "E (expectation) step" which increases $\mathcal{L}(Q, \boldsymbol{\theta})$ with respect to Q and a "M (maximization) step" which increases $\mathcal{L}(Q, \boldsymbol{\theta})$ with respect to the parameters $\boldsymbol{\theta}$. This view of the EM algorithm has been exploited recently in the context of variational methods for graphical models; see (Frey 1998; Jordan et al. 1999) for excellent overviews. Its typical use is to restrict the family of distributions from which Q can be chosen in order to simplify the E-step. Of course, this approach is not guaranteed to maximize the log-likelihood but if the family of Q distributions is rich enough the lower bound $\mathcal{L}(Q, \boldsymbol{\theta})$ can be close to the actual value of the log-likelihood. This causes a balancing act of choosing the Q distributions sufficiently simple to make the E-step tractable and also rich enough to have a tight lower bound.

The reader might suspect that since I coined the previous algorithm generalized EM, it is in some way related to the well-known standard EM algorithm (Dempster, Laird, and Rubin 1977). This is indeed the case if we do not impose any restrictions on Q. $Q(\mathbf{z}|\mathbf{x}^n, \boldsymbol{\theta})$ can then be chosen equal to the true posterior $p(\mathbf{z}|\mathbf{x}^n, \boldsymbol{\theta})$ and make the Kullback-Leibler divergence after the E-step equal to zero (Cover and Thomas 1991) to give the tightest lower bound $\mathcal{L}(Q, \boldsymbol{\theta})$. The M-step of standard EM typically maximizes and not just increases $\mathcal{L}(Q, \boldsymbol{\theta})$ with respect to the parameters $\boldsymbol{\theta}$. This gives

Algorithm 2 which performs coordinate ascent in the log-likelihood, which is exactly what we are looking for. At each cycle the EM algorithm is guaranteed to increase the log-likelihood unless it is already at a local maximum (Neal and Hinton 1999). The fact that Algorithm 2 is indeed standard EM as traditionally presented can be seen by looking more closely at the M-step. Using the choice of Q in the E-step and the definition of the free energy $\mathcal{L}(Q^{\text{new}}, \boldsymbol{\theta})$ (the first term in (2.6)) the M-step is equivalent to:

$$\begin{aligned}
\boldsymbol{\theta} &:= \underset{\boldsymbol{\theta}^{\text{new}}}{\operatorname{argmax}} \sum_{n,\mathbf{z}} Q^{\text{new}}(\mathbf{z}|\mathbf{x}^n, \boldsymbol{\theta}) \ln \frac{p(\mathbf{x}^n, \mathbf{z}|\boldsymbol{\theta}^{\text{new}})}{Q^{\text{new}}(\mathbf{z}|\mathbf{x}^n, \boldsymbol{\theta})} \\
&= \underset{\boldsymbol{\theta}^{\text{new}}}{\operatorname{argmax}} \sum_{n,\mathbf{z}} p(\mathbf{z}|\mathbf{x}^n, \boldsymbol{\theta}) \ln p(\mathbf{x}^n, \mathbf{z}|\boldsymbol{\theta}^{\text{new}}) - \sum_{n,\mathbf{z}} p(\mathbf{z}|\mathbf{x}^n, \boldsymbol{\theta}) \ln p(\mathbf{z}|\mathbf{x}^n, \boldsymbol{\theta}) \qquad (2.7) \\
&= \underset{\boldsymbol{\theta}^{\text{new}}}{\operatorname{argmax}} \quad \text{expected complete log-likelihood} \quad - \quad \text{constant},
\end{aligned}$$

which is, apart from an irrelevant constant, the standard formulation of the EM algorithm (Dempster, Laird, and Rubin 1977).

It is enlightening to compare the M-step (2.7) in terms of the expected complete log-likelihood with the system of coupled equations we derived earlier (2.5). The necessary conditions for maximization in the M-step can namely also be written as solving the following system (for each θ_j^{new}):

$$\sum_{n,\mathbf{z}} p(\mathbf{z}|\mathbf{x}^n, \boldsymbol{\theta}) \frac{\partial}{\partial \theta_j^{\text{new}}} \ln p(\mathbf{x}^n, \mathbf{z}|\boldsymbol{\theta}^{\text{new}}) = 0,$$

which through the estimation of the posterior of the hidden variables in the E-step is not coupled anymore: things have really been simplified by the EM algorithm and, moreover, convergence to a local maximum of the log-likelihood is guaranteed.

We could summarize the main idea of the EM algorithm as follows. Maximum likelihood estimation in a model with hidden variables leads to a system of coupled highly non-linear equations which can be decoupled by first estimating the distribution of the hidden variables given the data and the model parameters; this is the E-step, viz. estimate $p(\mathbf{z}|\mathbf{x}^n, \boldsymbol{\theta})$. All one needs to do in the M-step is to maximize the expected (with respect to the hidden variables) log-likelihood of the complete data, which is often a far easier problem because the hidden variables provide extra information.

In the rest of this thesis, the standard and generalized EM algorithms will be applied again and again to a variety of models with hidden variables. We start with one of the simplest of such models: mixture models.

2.2 Mixture Models and the EM Algorithm

As said at the beginning of section 2.1, the basic approach for density modeling is the parametric approach in which we choose a specific probability density function and maximize its likelihood. While this method is often simple, it is also sensitive to model mismatch: obviously it is difficult to know in advance which density function will fit the data best. Moreover, the parametric approach for analytically simple densities such as those belonging to the exponential family (for example, Duda and Hart 1973) is restricted to modeling unimodal distributions. If the data has several modes or clusters, the resulting model will be poor.

The toy example of Chapter 1 already illustrated the need for a more flexible model and the usefulness of mixing simple density functions. I recall that a mixture model is defined as a linear

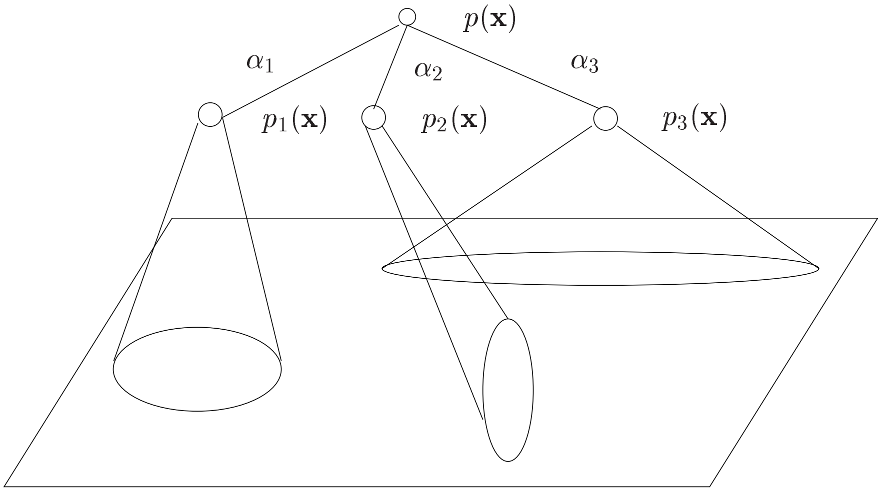


Figure 2.1: Density modeling with a mixture model.

combination of m *component* densities $p_j(\mathbf{x}|\boldsymbol{\theta}_j)$:

$$p(\mathbf{x}|\boldsymbol{\theta}, \boldsymbol{\alpha}) = \sum_{j=1}^{m} \alpha_j p_j(\mathbf{x}|\boldsymbol{\theta}_j), \tag{2.8}$$

where the α_j are the *mixing coefficients* which are non-negative and sum to one. This guarantees that $p(\mathbf{x})$ is a valid density function. The parameters of the mixture model are $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_m)$ and $\boldsymbol{\alpha}$. The mixture model approach of combining several simple density functions into a more complex one is illustrated in Figure 2.1.

We can also introduce mixture models in a slightly more formal way from the perspective of a model with hidden variables. This involves the introduction of a discrete hidden variable $z = 1 \ldots m$, labeling the component density functions (using the sum (A.16) and product (A.17) rules):

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) = \sum_{j=1}^{m} p(\mathbf{x}, z = j|\boldsymbol{\theta}) = \sum_{j=1}^{m} P(z = j|\boldsymbol{\theta}) p(\mathbf{x}|z = j, \boldsymbol{\theta}).$$

Assuming that the parameter vector $\boldsymbol{\theta}$ is partitioned among the components $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_m)$:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{j=1}^{m} P(z = j) p(\mathbf{x}|z = j, \boldsymbol{\theta}_j), \tag{2.9}$$

which is identical to (2.8). In the rest of this thesis, I will often use a more compact notation and leave the parameters implicit since they will normally be clear in the specific context:

$$p(\mathbf{x}) = \sum_{j} \alpha_j p_j(\mathbf{x}).$$

We now discuss in some detail how the EM algorithm can be applied to mixture models. Specific choices for the component densities $p_j(\mathbf{x})$ and derivations of the corresponding instantiations of the EM algorithm are described in sections 2.2.1 and 2.4. However, part of the EM algorithm is independent of the choice of the component densities, viz. the E-step and the optimization of the mixing coefficients in the M-step.

The mixture model error function which we want to minimize is the negative log-likelihood (2.3) of the mixture density (2.9) on the training data $\{\mathbf{x}^n\}$:

$$E(\boldsymbol{\theta}) = -\sum_n \ln \sum_{j=1}^{m} P(z=j)p(\mathbf{x}^n|z=j,\boldsymbol{\theta}_j). \tag{2.10}$$

Note that, in general, this error function has multiple local minima of different values and that the EM algorithm only guarantees converge towards one of them.

With respect to the E-step, we recall from the previous section that it involves estimation of the posterior of the hidden variables $p(z=j|\mathbf{x}^n,\boldsymbol{\theta})$. For a mixture model, this is particularly simple since a direct application of Bayes' rule (A.18) gives:

$$\text{E-step:} \qquad h_j(\mathbf{x}^n) := p(z=j|\mathbf{x}^n,\boldsymbol{\theta}) = \frac{P(z=j|\boldsymbol{\theta})p(\mathbf{x}^n|z=j,\boldsymbol{\theta}_j)}{p(\mathbf{x}^n|\boldsymbol{\theta})} = \frac{\alpha_j p_j(\mathbf{x}^n)}{p(\mathbf{x}^n)} = \frac{\alpha_j p_j(\mathbf{x}^n)}{\sum_{i=1}^{m} \alpha_i p_i(\mathbf{x}^n)}. \tag{2.11}$$

These posteriors for the mixing coefficients can be interpreted as indicators of the "responsibility" that component j takes for a data point $\mathbf{x}^n$. They might be seen as the probabilistic counterpart of a hard decision which attributes each of the data points to only one component and which would decouple the problem in separate parts which can easily be solved.

The M-step consists of the maximization of the expected complete log-likelihood (first term in (2.7)) or equivalently the minimization of its negation, with respect to the parameters of the mixture model:

$$\mathcal{E}(E_c) = -\sum_{n,\mathbf{z}} p(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta}) \ln p(\mathbf{x}^n,\mathbf{z}|\boldsymbol{\theta}^{\text{new}}) = -\sum_n \sum_{j=1}^{m} p(z=j|\mathbf{x}^n,\boldsymbol{\theta}) \ln p(\mathbf{x}^n, z=j|\boldsymbol{\theta}^{\text{new}}). \tag{2.12}$$

Using the definition of the posterior (2.11) and the product rule, we have:

$$\begin{aligned} &= -\sum_n \sum_{j=1}^{m} h_j(\mathbf{x}^n) \ln\{\alpha_j^{\text{new}} p_j(\mathbf{x}^n|\boldsymbol{\theta}_j^{\text{new}})\} \\ &= -\sum_n \sum_{j=1}^{m} h_j(\mathbf{x}^n) \ln \alpha_j^{\text{new}} - \sum_n \sum_{j=1}^{m} h_j(\mathbf{x}^n) \ln p_j(\mathbf{x}^n|\boldsymbol{\theta}_j^{\text{new}}), \end{aligned} \tag{2.13}$$

which is called the (expected) *complete error function*. Since the first term of (2.13) does not depend on the choice of the component densities, the mixture coefficients α_j^{new} can be optimized independently. The constraint on the mixing coefficients that $\sum_j \alpha_j^{\text{new}} = 1$ can be included in the optimization problem with a Lagrangian function for the first part of the complete error function (2.13):

$$L(\boldsymbol{\alpha}^{\text{new}}, \lambda) = -\left[\sum_n \sum_{j=1}^{m} h_j(\mathbf{x}^n) \ln \alpha_j^{\text{new}}\right] + \lambda \left[\sum_j \alpha_j^{\text{new}} - 1\right],$$

and setting the partial derivatives with respect to the $\boldsymbol{\alpha}^{\text{new}}$ and λ to zero gives a system of $m+1$ linear equations, the solution of which is:

$$\alpha_j^{\text{new}} = \frac{1}{N} \sum_n h_j(\mathbf{x}^n), \tag{2.14}$$

where N is the number of patterns in the training set $\{\mathbf{x}^n\}$. The M-step for the other parameters now involves the minimization of the second term of the complete error function (2.13) only.

It is also interesting to see that this part of the complete error function forms a completely decoupled optimization problem: $-\sum_n \sum_{j=1}^m h_j(\mathbf{x}^n) \ln p_j(\mathbf{x}^n|\boldsymbol{\theta}_j^{\text{new}})$ can be dealt with separately for each of the component densities. The estimates of the parameters are similar to the ones one would find by maximum likelihood on the single component density but weighted by the posteriors $h_j(\mathbf{x}^n)$ as we already saw in the case of a Gaussian mixture model in Chapter 1. We will see many other examples of this later on.

As a first example, the next section describes in more detail the most popular kind of mixture model: a Gaussian mixture model. Its popularity is of course mostly due to its simplicity and the nice analytical and statistical properties of the Gaussian distribution. This choice makes that the M-step can be performed analytically in a simple way. While this is actually the case for any density from the exponential family (Titterington, Smith, and Makov 1985, Chapter 4), I will limit myself to Gaussian component densities here.

2.2.1 Gaussian Mixture Models

Gaussian mixture models are a standard tool for density estimation and are described in many textbooks (for example, (Bishop 1995; McLachlan and Basford 1988; Titterington, Smith, and Makov 1985)). A GMM is defined as a mixture model (2.8) with component distributions which are multivariate Gaussian with a $d \times d$ symmetric and positive-definite covariance matrix $\boldsymbol{\Sigma}_j$ and $d \times 1$ mean $\boldsymbol{\mu}_j$:

$$p_j(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) = \frac{1}{|\boldsymbol{\Sigma}_j|^{1/2}(2\pi)^{d/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_j)^T {\boldsymbol{\Sigma}_j}^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)\right\}. \tag{2.15}$$

The parameters of a GMM can be determined by maximum likelihood estimation with the EM algorithm within the general framework for mixture models described in the previous section. This already gives us the E-step (2.11) and the estimation of the mixing coefficients (2.14). The updates in the M-step for the parameters $\boldsymbol{\Sigma}_j$ and $\boldsymbol{\mu}_j$ of the component densities can be found by minimizing the complete error function (2.13) and are summarized in the following instantiation of the EM algorithm (for example, (Duda and Hart 1973, Chapter 6)):

E-step: Estimation of the posteriors, for all j:

$$h_j(\mathbf{x}^n) = \frac{\alpha_j p_j(\mathbf{x}^n)}{\sum_{i=1}^m \alpha_i p_i(\mathbf{x}^n)}.$$

M-step: Re-estimation of the parameters of the GMM (where the new parameter values are denoted with a prime) for all j:

$$\begin{aligned}
\alpha'_j &= \frac{1}{N}\sum_n h_j(\mathbf{x}^n) \\
\boldsymbol{\mu}'_j &= \frac{\sum_n h_j(\mathbf{x}^n)\mathbf{x}^n}{\sum_n h_j(\mathbf{x}^n)} \\
\boldsymbol{\Sigma}'_j &= \frac{\sum_n h_j(\mathbf{x}^n)(\mathbf{x}^n - {\boldsymbol{\mu}_j}')(\mathbf{x}^n - {\boldsymbol{\mu}_j}')^T}{\sum_n h_j(\mathbf{x}^n)}.
\end{aligned}$$

The updates for the mean $\boldsymbol{\mu}_j$ and the covariance matrix $\boldsymbol{\Sigma}_j$ correspond indeed to a sort of weighted (by $h_j(\mathbf{x}^n)$) versions of the sample mean and the sample covariance matrix. GMMs with full covariance matrices have several disadvantages, all related to the fact that a full covariance matrix contains $d(d+1)/2$ free parameters (the factor of $1/2$ is due to symmetry) which becomes unwieldy for high-dimensional data. Consequently, in each M-step the update of each of the $\boldsymbol{\Sigma}_j$ has a complexity of $O(d^2N)$. Moreover, the update of the posteriors in the E-step and the calculation of the likelihood function (for example, to monitor convergence on a training or validation set) require calculating the inverse and determinant of the $d \times d$ matrices $\boldsymbol{\Sigma}_j$. Even if one exploits the fact that the sample covariance matrix is symmetric and supposed to be positive definite, in order to use a Cholesky decomposition (Press et al. 1992), the computational complexity is $O(d^3)$. With respect to the positive definiteness of the estimate $\boldsymbol{\Sigma}'_j$, one can observe that this constraint is satisfied if the number of data points is large enough with respect to the dimension of data space: the matrix has to be full rank, that is, a necessary condition is $N > d$. This is especially restrictive in the mixture model context since the number of data points associated with a specific component can be a small subset of the total number of data points.[1] Therefore, in order to obtain well-determined estimates and to avoid overfitting on the training data, a large training set is often needed.

A standard remedy for the problem of badly determined parameters and overfitting when data is scarce is using regularization, that is, an extra penalty term in the error function which encourages smoother estimations. In the rest of this thesis, I will use a penalized likelihood approach which has been proposed for GMMs by Ormoneit and Tresp (1998). This requires only some additional factors in the M-step update of the covariance matrix and is numerically more stable:

$$\boldsymbol{\Sigma}'_j = \frac{\{\sum_n h_j(\mathbf{x}^n)(\mathbf{x}^n - \boldsymbol{\mu}'_j)(\mathbf{x}^n - \boldsymbol{\mu}'_j)^T\} + \beta \mathbf{I}_d}{\{\sum_n h_j(\mathbf{x}^n)\} + 1}. \tag{2.16}$$

Of course, this approach requires tuning the β parameter on a validation set.

Another natural way of dealing with scarce data is to limit the number of free parameters in a GMM by imposing constraints on the form of the covariance matrix. Three obvious possibilities are covariance matrices which are either:

- spherical: $\Sigma_j = \sigma_j^2 \mathbf{I}_d$ as throughout Chapter 1. A single parameter for the whole covariance structure which gives an unflexible model.
- diagonal: with all off-diagonal elements equal to zero. This gives d parameters but a model in which the axes of the Gaussians are aligned with the data axes; it does not capture correlation amongst the variables.
- tied: parameters of the covariance matrices are tied across the component densities (Bellegarda and Nahamoo 1990). One of the simplest examples is to have one covariance matrix common to all Gaussian components.

Spherical and diagonal covariance matrices limit the computational complexity of the update of each covariance structure in the M-step to $O(dN)$. Moreover, also the inverse and their determinant are $O(d)$. Appendix F contains a detailed derivation of the EM algorithm for a GMM with spherical covariance matrices for those who want to have a look at the update formulas. I will refer to these models as spherical, diagonal, tied, and full GMMs respectively.

A problem with maximum likelihood estimation for GMMs is that the likelihood goes to infinity for certain parameter values on the boundary of parameter space. This is, for example, the case when in a mixture of spherical Gaussians, one of the data points is used as mean $\boldsymbol{\mu}_j$ and σ_j approaches zero (Duda and Hart 1973, Chapter 6). While this is disturbing, a large body of empirical and theoretical

[1] And α_j gives an estimate of the portion of the data associated with a component.

evidence (Titterington, Smith, and Makov 1985, Chapter 4) points to the existence of satisfactory finite local optima. I take a pragmatic point of view and in the experiments later on (section 2.6), good initialization and a small lower bound on the values of the variance parameters seem to be sufficient for avoiding the singularities of the likelihood surface. Actually, the penalized update of $\mathbf{\Sigma}_j$ (2.16) can be interpreted as bounding the diagonal of the covariance matrix away from zero by the term $\beta \mathbf{I}_d$.

The constraints on the form of the covariance structure introduced above might seem natural, they are also quite restrictive. Modeling power can be increased by adding components to the mixture but that seems to be begging the question. Is there a way to design models which cover the big gap between having a diagonal covariance matrix (d parameters) and a full covariance matrix ($d(d+1)/2$ parameters)? In the next section, it is shown how the introduction of continuous hidden or *latent* variables can lead to more flexible models which smoothly fill up this gap. We first treat the case of simple parametric modeling and extend it to mixture models in section 2.4.

2.3 Linear Latent Variable Models

The problem of the GMMs with a full covariance matrix in the previous section is their huge number of free parameters for high-dimensional data. How could we use hidden variables to control the number of parameters? A possible answer is the introduction of a latent space of dimension $\ell \leq d-1$ for the latent variables $\mathbf{z} = (z_1, z_2, \ldots, z_\ell)^T$ and specifying probabilistically how latent and observed variables are related. This can be interpreted as a form of dimensionality reduction or feature extraction. Since our starting point is a GMM, we can safely assume that the latent space is $\mathbb{R}^\ell$ and the data space is $\mathbb{R}^d$. The latent variables now being continuous, marginalizing over them is done by integration and not by summing as was assumed in section 2.1:

$$p(\mathbf{x}^n|\boldsymbol{\theta}) = \int_{\mathbf{z}} p(\mathbf{x}^n, \mathbf{z}|\boldsymbol{\theta}) d\mathbf{z} = \int_{\mathbf{z}} p(\mathbf{x}^n|\mathbf{z}, \boldsymbol{\theta}) p(\mathbf{z}) d\mathbf{z}. \tag{2.17}$$

To keep this integration tractable, we limit ourselves to linear mappings and Gaussian distributions. We define $p(\mathbf{x}^n|\mathbf{z}, \boldsymbol{\theta})$ through the following mapping from latent space to data space, together with a Gaussian prior $p(\mathbf{z})$ for the latent variables:

$$\mathbf{x} = \mathbf{W}\mathbf{z} + \boldsymbol{\mu} + \boldsymbol{\varepsilon} \qquad \text{with} \qquad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_\ell) \quad , \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{R}), \tag{2.18}$$

with model parameters $\mathbf{W}, \mathbf{R}$, and $\boldsymbol{\mu}$. The idea behind the model is illustrated in Figure 2.2. The prior distribution over the latent variables is a simple Gaussian ball (left-hand part of Figure 2.2) in latent space. A $d \times \ell$ generative or *factor loading* matrix $\mathbf{W}$ maps the latent space into data space. The effect is to stretch and translate the Gaussian ball in data space (right-hand part of Figure 2.2) resulting in a sort of ℓ-dimensional pancake in d-dimensional space; the pancake can also be translated over $\boldsymbol{\mu}$. To get a real manifold in data space, the pancake is finally convolved in data space with a Gaussian noise distribution $p(\boldsymbol{\varepsilon})$ with $d \times d$ covariance matrix $\mathbf{R}$ which is independent of $\mathbf{z}$ (Roweis and Ghahramani 1999). This can also be interpreted as a generative model in which the hidden variables model the causes for the observed data (Figure 2.3).

Due to the restrictions imposed upon the form of the mapping and the priors, everything stays entirely in the Gaussian domain. The conditional distribution of the latent variables given the observed variables is:[2]

$$\mathbf{x}|\mathbf{z} \sim \mathcal{N}(\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \mathbf{R}), \tag{2.19}$$

[2] With $\mathbf{x}|\mathbf{z} \sim \ldots$ as a shorthand for $p(\mathbf{x}|\mathbf{z}) = \ldots$

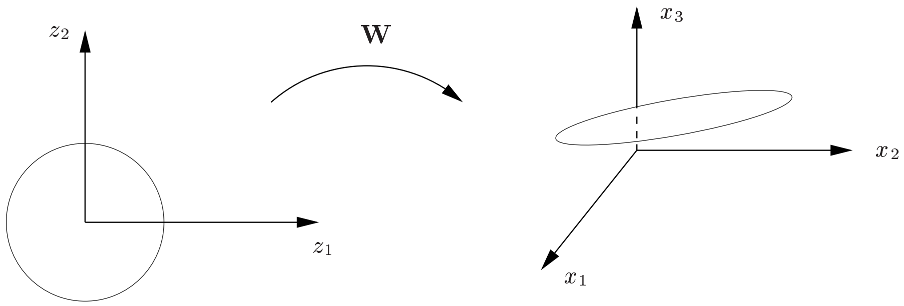


Figure 2.2: A generative model from a latent space of dimension 2 to a data space of dimension 3.

and its convolution (2.17) with the Gaussian prior $p(\mathbf{z})$ can be performed analytically. This gives the distribution of the observed data, which is also Gaussian:[3]

$$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{M}) \qquad \text{with} \qquad \mathbf{M} = \mathbf{R} + \mathbf{W}\mathbf{W}^T. \tag{2.20}$$

But for the moment, we have not gained anything, since the covariance matrix of the observation noise $\mathbf{R}$ is unrestricted and the number of parameters is still $O(d^2)$. All second-order information in the data could be modeled by choosing $\mathbf{R}$ equal to the sample covariance matrix. The simplest way of restricting $\mathbf{R}$ is to make the observed data independent given the latent data, that is (see (2.19)):

- $\mathbf{R} = \sigma^2 \mathbf{I}_d$: the latent variable model is called probabilistic principal component analysis (PPCA) (Tipping and Bishop 1999) or sensible principal component analysis (Roweis 1998; Roweis and Ghahramani 1999). This terminology has been chosen while with $\sigma^2 \to 0$ conventional PCA is recovered (and even a stronger statement can be made as we will see below).
- $\mathbf{R} \sim$ diagonal matrix: the latent variable model is standard factor analysis (Everitt 1984). In this case, the hidden variables $\mathbf{z}$ are often called *factors*.

These restrictions imply that a linear latent variable model can be viewed as a way of capturing the covariance structure of the d-dimensional observed data through $\mathbf{M}$ (2.20) with at most $d(\ell+1)$ parameters. This might be interpreted as a kind of *discrete* regularization in which one can tune the complexity of the model by choosing the dimension of latent space ℓ. In this way, one can cover the whole spectrum of a covariance matrix with $O(d)$ parameters to one with $O(d^2)$ in a more or less smooth manner. It is also interesting to note that even if one chooses $\ell = 1$, the resulting covariance matrix has a number of parameters of the same order as a Gaussian with a diagonal covariance matrix, but will still be able to capture correlations between the observed variables. A central issue in the use of linear latent variable models is that of choosing an appropriate value for ℓ. We will see that this problem can be addressed by a Bayesian treatment (section 2.7).

Estimation What about maximum likelihood estimation for linear latent variable models? The log-likelihood (2.2) of (2.20) is:

$$\mathcal{L}(\boldsymbol{\mu}, \mathbf{W}, \mathbf{R}) = \sum_n \ln[\mathcal{N}(\boldsymbol{\mu}, \mathbf{M})].$$

[3] This follows from the fact that the sum of two independent Gaussian distributed quantities is also Gaussian distributed with as mean the sum of the means and as covariance matrix the sum of the covariance matrices. $\mathbf{x}$ is the sum of $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ and $\mathbf{Wz} + \boldsymbol{\mu} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{W}\mathbf{W}^T)$ where the latter follows from (A.19).

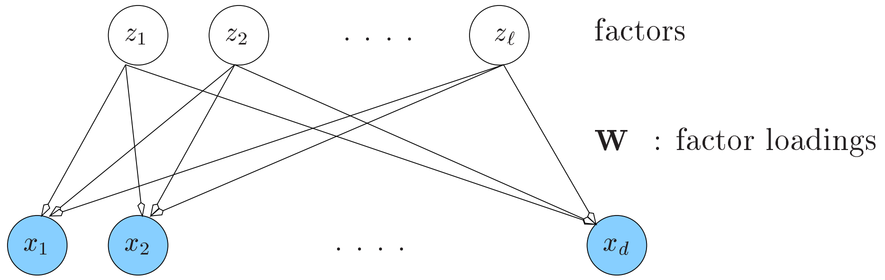


Figure 2.3: Network representation of probabilistic principal component analysis and factor analysis.

Using the definition of a multivariate Gaussian (2.15):

$$= \sum_n \ln \left[\frac{1}{|\mathbf{M}|^{1/2}(2\pi)^{d/2}} \exp\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \mathbf{M}^{-1}(\mathbf{x}-\boldsymbol{\mu})\} \right].$$

Distributing the logarithm over the product:

$$= -\frac{N}{2}\{d \ln(2\pi) + \ln|\mathbf{M}|\} - \frac{1}{2}\sum_n \{(\mathbf{x}-\boldsymbol{\mu})^T \mathbf{M}^{-1}(\mathbf{x}-\boldsymbol{\mu})\}.$$

The last term is scalar, thus one can take the trace of it and perform trace rotation (A.1):

$$\mathcal{L}(\boldsymbol{\mu}, \mathbf{W}, \mathbf{R}) = -\frac{N}{2}\{d \ln(2\pi) + \ln|\mathbf{M}| + \mathrm{tr}(\mathbf{M}^{-1}\mathbf{S})\}, \tag{2.21}$$

where $\mathbf{S}$ is the $d \times d$ sample covariance matrix:

$$\mathbf{S} = \frac{1}{N}\sum_n (\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^T. \tag{2.22}$$

The maximum likelihood estimate of the mean $\boldsymbol{\mu}$ is simply the sample mean of the data:

$$\boldsymbol{\mu}_{\mathrm{ML}} = \frac{1}{N}\sum_n \mathbf{x}^n. \tag{2.23}$$

The log-likelihood (2.21) with respect to the parameters $\mathbf{W}$ and $\mathbf{R}$ can be maximized by the EM algorithm. For factor analysis, this instantiation of EM was derived by Rubin and Thayer (1982) and we come back to this in the next section. The special case of PPCA with $\mathbf{R} = \sigma^2 \mathbf{I}_d$ was dealt with by Roweis (1998) and Tipping and Bishop (1999). Tipping and Bishop actually showed that for the PPCA model a closed-form solution for the global maximum of the likelihood exists and that this is the only stable maximum of the likelihood surface. This solution involves an eigendecomposition of the sample covariance matrix $\mathbf{S}$ and the maximum likelihood estimate for the generative matrix $\mathbf{W}$ spans the ℓ-dimensional *principal* subspace of the data:

$$\mathbf{W}_{\mathrm{ML}} = \mathbf{U}_\ell(\Lambda_\ell - \sigma^2 \mathbf{I}_\ell)^{1/2}\mathbf{V}, \tag{2.24}$$

where Λ_ℓ is a $\ell \times \ell$ diagonal matrix with the ℓ largest eigenvalues λ_i of $\mathbf{S}$ on the diagonal. $\mathbf{U}_\ell$ is a $d \times \ell$ matrix containing in its columns the corresponding *principal* eigenvectors of $\mathbf{S}$, and $\mathbf{V}$ is an arbitrary $\ell \times \ell$ rotation matrix ($\mathbf{V}^{-1} = \mathbf{V}^T$).[4]

[4]This arbitrary rotation implies that the number of free parameters for PPCA is $d\ell + 1 - \ell(\ell-1)/2$.

This clearly shows the strong relation with standard principal component analysis, a popular technique for dimensionality reduction (Jollife 1986). PCA is based on a linear projection onto the principal subspace spanned by the (orthonormal) principal eigenvectors $\mathbf{U}_\ell$ of the sample covariance matrix $\mathbf{S}$. The new coordinate values are called the *principal components.* The principal component projection is the orthogonal projection which minimizes the squared reconstruction error:

$$\sum_n ||\mathbf{x}^n - \mathbf{U}_\ell \mathbf{U}_\ell^T (\mathbf{x}^n - \boldsymbol{\mu}_{\mathrm{ML}}) + \boldsymbol{\mu}_{\mathrm{ML}}||^2. \tag{2.25}$$

Note that projection and reconstruction boil down to simple matrix multiplications due to the orthonormality of the eigenvectors. An equivalent characterization of PCA is as the linear projection which maximizes the projection variances. Typical applications of PCA are data compression and denoising.

The maximum likelihood estimate for the observation noise variance $\mathbf{R} = \sigma^2 \mathbf{I}_d$ can also be found in closed form (Tipping and Bishop 1999):

$$\sigma^2_{\mathrm{ML}} = \frac{1}{d-\ell} \sum_{i=\ell+1}^{d} \lambda_i, \tag{2.26}$$

and this has now the interpretation of the average variance lost per dimension which has been left out.

Some Remarks The probabilistic model of PPCA has several advantages with respect to standard PCA such as the availability of a proper density model with likelihood scores for model comparison. For standard PCA one can evaluate the reconstruction cost (2.25) of a point but this measure is insensitive to arbitrary translations within the principal subspace. We are mainly interested in another advantage of PPCA, viz. the ease of incorporating it into a mixture of constrained Gaussians. In section 2.4, we address the extension to mixtures of latent variable models and their maximum likelihood estimation via the EM algorithm in detail.

The reader might wonder about the consequences of the seemingly small difference in the choice of the noise model $\mathbf{R}$ which is spherical for PPCA and diagonal for FA. It is easy to see from the generative model (2.18) that factor analysis is insensitive to rescaling the coordinates of the data.[5] Scaling factors can namely be incorporated in the corresponding rows of $\mathbf{W}$ and the corresponding entry of the diagonal $\mathbf{R}$. This transforms ML solutions for the original data into ML solutions for the rescaled data. At the same time, FA is sensitive to the choice of the coordinate system in data space: an orthogonal transformation of the data cannot be incorporated into $\mathbf{R}$ because of the diagonality constraint. The constraint corresponds to the assumption that the components of the noise are independent in this particular coordinate system (like a sort of sensor noise). For probabilistic PCA, it is exactly the other way around. PPCA is insensitive to a rotation of the data which can be incorporated by left multiplying $\mathbf{W}$ by the same rotation. However, scaling the data can only be handled if all the coordinates are scaled by the same factor because of sphericality constraint on $\mathbf{R}$.

PPCA (like standard PCA) can have difficulties in separating information from noise as illustrated by the toy example of Figure 2.4. In this example, the second coordinate x_2 is much noisier than the others. Factor analysis with its diagonal noise covariance $\mathbf{R}$ succeeds in modeling this noisy component and finds the correlation between the third and the fourth coordinate x_3, x_4. PPCA on the other hand, is confused by the high variance on the second coordinate x_2 and finds a correlation between dimensions 2–4 and an estimate of the noise that averages over the remaining noise in the

[5] This argument and the next ones are done by "handwaving". A precise statement can be made by considering the effect of a linear transformation $\mathbf{x} \to \mathbf{A}\mathbf{x}$ on the log-likelihood (2.21). This a nice exercise and all the details are in (Tipping and Bishop 1999).

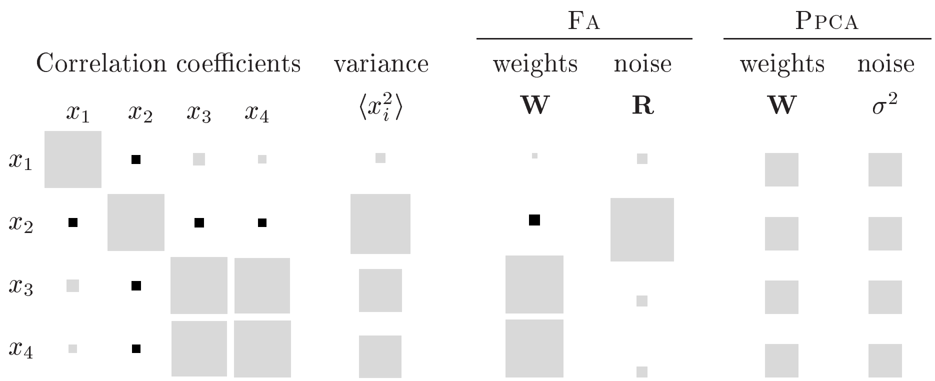


Figure 2.4: A toy data set of 1000 patterns in four dimensions (x_1, x_2, x_3, x_4). The first dimension has been generated from a Gaussian having small variance, while the second dimension has been generated from a Gaussian with high variance. The first two dimensions are independent of the other dimensions. The third and fourth dimension are highly correlated and have been generated from the same values (Gaussian with medium-valued variance) corrupted by small Gaussian noise. This can be seen from the Hinton diagrams with gray for positive and black for negative values and the size of the square proportional to the value of the corresponding parameter. The first two diagrams represent the correlation coefficients and the variance of the data. This data was used to train a FA and a PPCA model with one factor ($\ell = 1$). The other Hinton diagrams shown correspond to the estimated 4×1 factor loading $\mathbf{W}$ and the 4×1 diagonal of the estimated noise covariance $\mathbf{R}$ for both models.

data. Loosely speaking, principal component analysis pays attention to both variance and covariance, whereas factor analysis looks only at covariance (Neal and Dayan 1997).

The reader is referred to (Roweis and Ghahramani 1999) and (Bishop 1999b) for a more general discussion of latent variable models which are non-linear (such as the generative topographic mapping) and dynamic in time (such as hidden Markov models and Kalman filters).

2.4 Mixtures of Factor Analysis and PCA

One important advantage of the linear latent variable models described in section 2.3 is that they define a proper probability model which can be extended to a mixture model. The mixture model (2.8) is then a linear combination of component distributions (2.20):

$$p_j(\mathbf{x}|\boldsymbol{\mu}_j, \mathbf{R}_j, \mathbf{W}_j) \sim \mathcal{N}(\boldsymbol{\mu}_j, \mathbf{R}_j + \mathbf{W}_j \mathbf{W}_j^T), \tag{2.27}$$

with separate parameters for each of the component distributions. With spherical noise covariance $\mathbf{R}_j$, the model is called a mixture of principal component analyzers (Tipping and Bishop 1999) and with a diagonal $\mathbf{R}_j$, it is called a mixture of factor analyzers (Ghahramani and Hinton 1996). These mixtures can be interpreted as a mixture of constrained Gaussians in which the number of parameters can be controlled through the dimension of the latent space ℓ without putting too strong constraints on the flexibility of the model, that is, on the form of the covariance matrix.

In this section, I present a detailed derivation of an EM algorithm for mixtures of factor analyzers and then specialize it to the EM algorithm for MPCAs derived by Tipping and Bishop (1999). An EM algorithm for MFAs was originally presented by Ghahramani and Hinton (1996) but the one presented here gives a nicer separation of concerns by staying close to Tipping and Bishop's algorithm for MPCAs. The reader who does not want to go into all the details of this derivation can skip this part and have a look at the final algorithms and a discussion of their computational complexity on page 34-35 right away. For those who dare to continue reading, it might be helpful to keep an eye on the matrix and probability identities which are listed in Appendix A which will be used extensively.

Derivation of EM Algorithm for Mixtures of Latent Variable Models

A mixture of latent variable models takes the following form:

$$p(\mathbf{x}) = \sum_{j=1}^{m} \alpha_j p_j(\mathbf{x}), \qquad \text{with} \qquad p_j(\mathbf{x}) \sim \mathcal{N}(\boldsymbol{\mu}_j, \mathbf{R}_j + \mathbf{W}_j\mathbf{W}_j^T), \tag{2.28}$$

where $p_j(\mathbf{x})$ is a single latent variable model (2.20) and the α_j are the mixing coefficients. The error function to be minimized is the negative log-likelihood (2.10):

$$E(\boldsymbol{\theta}) = -\sum_n \ln \sum_{j=1}^{m} \alpha_j p_j(\mathbf{x}^n|\boldsymbol{\mu}_j, \mathbf{R}_j, \mathbf{W}_j).$$

The maximum likelihood estimates for the parameters of this mixture model can be determined with a two-stage EM procedure (Tipping and Bishop 1999). The idea of this two-stage approach is to take into account the hidden variables indicating the component labels first and consider the latent variables $\{\mathbf{z}^n\}$ only in the second stage. This allows us to easily build upon the general framework for mixture models described in section 2.2.

First stage: E-step I recall that the E-step for general mixture models involves estimating the responsibilities of component j for each data point (2.11):

$$h_j(\mathbf{x}^n) = \frac{\alpha_j p_j(\mathbf{x}^n)}{\sum\limits_{i=1}^{m} \alpha_i p_i(\mathbf{x}^n)}. \tag{2.29}$$

First stage: M-step The M-step then consists of minimizing the expected complete error function (2.13) with respect to the parameters of the mixture model (with $'$ denoting the new parameter values):

$$\mathcal{E}(E_c) = -\sum_n \sum_{j=1}^{m} h_j(\mathbf{x}^n) \ln \left\{ \alpha'_j p_j(\mathbf{x}^n|\boldsymbol{\mu}'_j, \mathbf{R}'_j, \mathbf{W}'_j) \right\}. \tag{2.30}$$

As we know, the updates of the mixing coefficients are independent of the choice of the component densities (2.14):

$$\alpha'_j = \frac{1}{N} \sum_n h_j(\mathbf{x}^n). \tag{2.31}$$

In this first stage, we will only update the centers $\boldsymbol{\mu}_j$, the other parameters are dealt with in the second stage. Using (2.28) and the definition of a multivariate Gaussian (2.15), the complete error function (2.30) can be written as:

$$\mathcal{E}(E_c) = -\sum_n \sum_{j=1}^{m} h_j(\mathbf{x}^n) \ln \left[\alpha'_j (2\pi)^{-d/2} |\mathbf{M}'_j|^{-1/2} \exp \left\{ -\frac{1}{2} (\mathbf{x}^n - \boldsymbol{\mu}'_j)^T (\mathbf{M}'_j)^{-1} (\mathbf{x}^n - \boldsymbol{\mu}'_j) \right\} \right],$$

where the covariance matrix for the latent variable model is $\mathbf{M}'_j = \mathbf{R}'_j + \mathbf{W}'_j(\mathbf{W}'_j)^T$. For the centers of the component densities, the partial derivative of this complete error function is (using (A.2) and (A.3)):

$$\partial\left[-\sum_n\sum_{i=1}^m h_i(\mathbf{x}^n)\left\{-\frac{1}{2}(\mathbf{x}^n-\boldsymbol{\mu}'_i)^T(\mathbf{M}'_i)^{-1}(\mathbf{x}^n-\boldsymbol{\mu}'_i)\right\}\right]/\partial\boldsymbol{\mu}'_j = -\sum_n h_j(\mathbf{x}^n)(\mathbf{M}'_j)^{-1}(\mathbf{x}^n-\boldsymbol{\mu}'_j).$$

Setting this partial derivative to zero, we obtain the new estimate for the means:

$$\boldsymbol{\mu}'_j = \frac{\sum_n h_j(\mathbf{x}^n)\mathbf{x}^n}{\sum_n h_j(\mathbf{x}^n)}. \tag{2.32}$$

This concludes the first stage of the EM algorithm for a mixture of latent variable models. What still needs to be done is to find the updates of the parameters in $\mathbf{M}_j$, that is the weight matrices $\mathbf{W}_j$ and the local noise models $\mathbf{R}_j$. This is done with a second stage of the EM algorithm within the M-step of the first stage. As shown in section 2.1, decreasing and not minimizing the expected complete error function (2.30) in the M-step is sufficient for convergence to a local minimum. We will do exactly that in this second stage by introducing the latent variables $\mathbf{z}$ in the complete error function of the first stage (2.30) and performing one iteration of the EM algorithm to update $\mathbf{W}_j$ and $\mathbf{R}_j$. This is guaranteed to decrease $\mathcal{E}(E_c)$ and will, therefore, also increase the likelihood of the entire mixture model.

Second stage: E-step We start by introducing the continuous latent variables $\mathbf{z}_j$ in (2.30):

$$\mathcal{E}(E_c) = -\sum_n\sum_{j=1}^m h_j(\mathbf{x}^n)\ln\left[\int_{\mathbf{z}}\alpha'_j p_j(\mathbf{x}^n,\mathbf{z}|\boldsymbol{\mu}'_j,\mathbf{R}_j,\mathbf{W}_j)\,d\mathbf{z}\right]. \tag{2.33}$$

From the previous sections, we know that EM can take care of the integral inside the logarithm through estimation of the posterior of the latent variables $p_j(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta})$. The posterior can be found by observing that the joint distribution of the latent and observed variables for component j is also Gaussian distributed (according to the definition of the latent variable model (2.18) and (2.27)):

$$\begin{bmatrix}\mathbf{z}\\ \mathbf{x}\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\mathbf{0}\\ \boldsymbol{\mu}'_j\end{bmatrix},\begin{bmatrix}\mathbf{I}_\ell & \mathbf{W}_j^T\\ \mathbf{W}_j & \mathbf{R}_j+\mathbf{W}_j\mathbf{W}_j^T\end{bmatrix}\right).$$

The posterior can be determined directly from this joint distribution using (A.24) and is also Gaussian:

$$p_j(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\mu}'_j,\mathbf{R}_j,\mathbf{W}_j) = \mathcal{N}(\mathbf{W}_j^T\mathbf{M}_j^{-1}(\mathbf{x}^n-\boldsymbol{\mu}'_j),\mathbf{I}-\mathbf{W}_j^T\mathbf{M}_j^{-1}\mathbf{W}_j). \tag{2.34}$$

The Gaussian posterior is fully characterized by its first and second moments:

$$\langle\,\mathbf{z}_j^n\,\rangle = \mathbf{W}_j^T\mathbf{M}_j^{-1}(\mathbf{x}^n-\boldsymbol{\mu}'_j) \tag{2.35}$$

$$\langle\,\mathbf{z}_j^n(\mathbf{z}_j^n)^T\,\rangle = \mathbf{I}-\mathbf{W}_j^T\mathbf{M}_j^{-1}\mathbf{W}_j+\langle\,\mathbf{z}_j^n\,\rangle\langle\,\mathbf{z}_j^n\,\rangle^T. \tag{2.36}$$

These two equations seem to require the inversion of the $d\times d$ matrix $\mathbf{M}_j$, but they can be simplified using the matrix inversion lemma (A.8) (leaving out the indices):

$$\mathbf{M}^{-1} = (\mathbf{W}\mathbf{W}^T+\mathbf{R})^{-1} = \mathbf{R}^{-1}-\mathbf{R}^{-1}\mathbf{W}(\mathbf{I}_\ell+\mathbf{W}^T\mathbf{R}^{-1}\mathbf{W})^{-1}\mathbf{W}^T\mathbf{R}^{-1} = \mathbf{R}^{-1}-\mathbf{R}^{-1}\mathbf{W}\mathbf{N}^{-1}\mathbf{W}^T\mathbf{R}^{-1}, \tag{2.37}$$

with $\ell\times\ell$ matrix:

$$\mathbf{N} = \mathbf{I}_\ell+\mathbf{W}^T\mathbf{R}^{-1}\mathbf{W}. \tag{2.38}$$

We see that inverting $\mathbf{M}$ actually only requires the inversion of a $\ell \times \ell$ matrix $\mathbf{N}$ and taking the inverse of the noise covariance matrix $\mathbf{R}$ which is easy because of the diagonality constraint. Further simplification is possible with the following straightforward rewriting:

$$\mathbf{I}_\ell = \mathbf{N}\mathbf{N}^{-1} = \begin{array}{l} \nearrow \mathbf{N}^{-1} + \mathbf{W}^T\mathbf{R}^{-1}\mathbf{W}\mathbf{N}^{-1} \\ \searrow \mathbf{N}^{-1} + \mathbf{N}^{-1}\mathbf{W}^T\mathbf{R}^{-1}\mathbf{W} \end{array} \tag{2.39}$$

Thus, we can transform the first two factors in (2.35) using (2.37) and (2.39):

$$\mathbf{W}^T\mathbf{M}^{-1} = \mathbf{W}^T\mathbf{R}^{-1} - \mathbf{W}^T\mathbf{R}^{-1}\mathbf{W}\mathbf{N}^{-1}\mathbf{W}^T\mathbf{R}^{-1} = (\mathbf{I}_\ell - \mathbf{W}^T\mathbf{R}^{-1}\mathbf{W}\mathbf{N}^{-1})\mathbf{W}^T\mathbf{R}^{-1} = \mathbf{N}^{-1}\mathbf{W}^T\mathbf{R}^{-1}. \tag{2.40}$$

The first two terms of (2.36) can be written as (using (2.39)):

$$\mathbf{I}_\ell - \mathbf{W}^T\mathbf{M}^{-1}\mathbf{W} = \mathbf{I}_\ell - \mathbf{N}^{-1}\mathbf{W}^T\mathbf{R}^{-1}\mathbf{W} = \mathbf{N}^{-1}. \tag{2.41}$$

Substituting (2.40) and (2.41) in the moments (2.35) and (2.36) gives:

$$\langle\, \mathbf{z}_j^n \,\rangle = \mathbf{N}_j^{-1}\mathbf{W}_j^T\mathbf{R}_j^{-1}(\mathbf{x}^n - \boldsymbol{\mu}'_j) \tag{2.42}$$

$$\langle\, \mathbf{z}_j^n(\mathbf{z}_j^n)^T \,\rangle = \mathbf{N}_j^{-1} + \langle\, \mathbf{z}_j^n \,\rangle\langle\, \mathbf{z}_j^n \,\rangle^T. \tag{2.43}$$

Having fully characterized the posterior of the latent variables, this terminates the E-step of the second stage.

Second stage: M-step For the second-stage M-step, the expected (with respect to the posteriors of the latent variables) complete error function corresponding to (2.33) is:

$$\mathcal{E}(\hat{E}_c) = -\sum_n \sum_{j=1}^m h_j(\mathbf{x}^n) \int_{\mathbf{z}} p_j(\mathbf{z}|\mathbf{x}^n, \boldsymbol{\theta}) \left\{ \ln \alpha'_j p_j(\mathbf{x}^n, \mathbf{z}|\boldsymbol{\mu}'_j, \mathbf{R}'_j, \mathbf{W}'_j) \right\} d\mathbf{z}.$$

Given that the posterior is Gaussian and is fully characterized by its first and second moments, this can be written in a more compact way:

$$\mathcal{E}(\hat{E}_c) = -\sum_n \sum_{j=1}^m h_j(\mathbf{x}^n) \langle\, \ln \alpha'_j p_j(\mathbf{x}^n, \mathbf{z}|\boldsymbol{\mu}'_j, \mathbf{R}'_j, \mathbf{W}'_j) \,\rangle, \tag{2.44}$$

where $\langle\, . \,\rangle$ denotes the expectation with respect to the posterior distribution of $\mathbf{z}$. The joint probability distribution is (using (2.18) and the definition of a multivariate Gaussian (2.15)):

$$p_j(\mathbf{x}^n, \mathbf{z}) = p_j(\mathbf{x}^n|\mathbf{z})p_j(\mathbf{z}) = (2\pi)^{-d/2}|\mathbf{R}'_j|^{-1/2} \exp\left\{ -\frac{1}{2}(a_j^n)^T(\mathbf{R}'_j)^{-1}a_j^n \right\} (2\pi)^{-\ell/2} \exp\left\{ -\frac{1}{2}\mathbf{z}^T\mathbf{z} \right\},$$

where

$$a_j^n = \mathbf{x}^n - \mathbf{W}'_j\mathbf{z} - \boldsymbol{\mu}'_j.$$

Rewriting (2.44) using the above expression for the joint distribution and collecting irrelevant constants which do not depend on $\mathbf{W}'_j$ and $\mathbf{R}'_j$, gives:

$$\begin{aligned} \mathcal{E}(\hat{E}_c) = & -\sum_n \sum_{j=1}^m h_j(\mathbf{x}^n) \Big\{ -\frac{1}{2} \ln |\mathbf{R}'_j| - \frac{1}{2} \langle\, (\mathbf{x}^n - \mathbf{W}'_j\mathbf{z}_j^n - \boldsymbol{\mu}'_j)^T(\mathbf{R}'_j)^{-1}(\mathbf{x}^n - \mathbf{W}'_j\mathbf{z}_j^n - \boldsymbol{\mu}'_j) \,\rangle \Big\} \\ & + \text{constant}. \end{aligned} \tag{2.45}$$

Factoring out the last term on the first line (with the trace operator "tr" to get the appropriate moments):

$$\begin{aligned}
& -\frac{1}{2}\Big\{ (\mathbf{x}^n - \boldsymbol{\mu}'_j)^T (\mathbf{R}'_j)^{-1} (\mathbf{x}^n - \boldsymbol{\mu}'_j) - 2(\mathbf{x}^n - \boldsymbol{\mu}'_j)^T (\mathbf{R}'_j)^{-1} \mathbf{W}'_j \langle\, \mathbf{z}^n_j \,\rangle \\
& + \langle\, (\mathbf{z}^n_j)^T (\mathbf{W}'_j)^T (\mathbf{R}'_j)^{-1} \mathbf{W}'_j \mathbf{z}^n_j \,\rangle \Big\} \\
= \; & -\frac{1}{2}\Big[(\mathbf{x}^n - \boldsymbol{\mu}'_j)^T (\mathbf{R}'_j)^{-1} (\mathbf{x}^n - \boldsymbol{\mu}'_j) - 2(\mathbf{x}^n - \boldsymbol{\mu}'_j)^T (\mathbf{R}'_j)^{-1} \mathbf{W}'_j \langle\, \mathbf{z}^n_j \,\rangle \\
& + \mathrm{tr}\left\{ (\mathbf{W}'_j)^T (\mathbf{R}'_j)^{-1} \mathbf{W}'_j \langle\, \mathbf{z}^n_j (\mathbf{z}^n_j)^T \,\rangle \right\} \Big].
\end{aligned}$$

The second stage M-step then requires minimizing the complete error function (substituting the above expression in (2.45)):

$$\begin{aligned}
\mathcal{E}(\hat{E}_c) \; = \; & -\sum_n \sum_{j=1}^{m} h_j(\mathbf{x}^n) \Big[-\frac{1}{2} \ln |\mathbf{R}'_j| - \frac{1}{2} (\mathbf{x}^n - \boldsymbol{\mu}'_j)^T (\mathbf{R}'_j)^{-1} (\mathbf{x}^n - \boldsymbol{\mu}'_j) \\
& + (\mathbf{x}^n - \boldsymbol{\mu}'_j)^T (\mathbf{R}'_j)^{-1} \mathbf{W}'_j \langle\, \mathbf{z}^n_j \,\rangle - \frac{1}{2} \mathrm{tr}\left\{ (\mathbf{W}'_j)^T (\mathbf{R}'_j)^{-1} \mathbf{W}'_j \langle\, \mathbf{z}^n_j (\mathbf{z}^n_j)^T \,\rangle \right\} \Big]
\end{aligned} \tag{2.46}$$

with respect to the parameters which have not yet been optimized in the first stage of the EM algorithm: the generative matrices $\mathbf{W}_j$ and the noise covariance $\mathbf{R}_j$. For $\mathbf{W}_j$, the partial derivative is (and this is left as an exercise to the reader, it requires some derivatives of traces and scalar forms listed in Appendix A):

$$\frac{\partial \mathcal{E}(\hat{E}_c)}{\partial \mathbf{W}'_j} = -\sum_n h_j(\mathbf{x}^n) \left\{ (\mathbf{R}'_j)^{-1} (\mathbf{x}^n - \boldsymbol{\mu}'_j) \langle\, \mathbf{z}^n_j \,\rangle^T - (\mathbf{R}'_j)^{-1} \mathbf{W}'_j \langle\, \mathbf{z}^n_j (\mathbf{z}^n_j)^T \,\rangle \right\} = 0, \tag{2.47}$$

that has the following solution:

$$\mathbf{W}'_j = \left[\sum_n h_j(\mathbf{x}^n) (\mathbf{x}^n - \boldsymbol{\mu}'_j) \langle\, \mathbf{z}^n_j \,\rangle^T \right] \left[\sum_n h_j(\mathbf{x}^n) \langle\, \mathbf{z}^n_j (\mathbf{z}^n_j)^T \,\rangle \right]^{-1}. \tag{2.48}$$

For noise covariance $\mathbf{R}_j$, the partial derivative of the complete error function (2.46) with respect to its inverse is (yet another small exercise for the reader):

$$\begin{aligned}
\frac{\partial \mathcal{E}(\hat{E}_c)}{\partial (\mathbf{R}'_j)^{-1}} \; = \; & -\sum_n h_j(\mathbf{x}^n) \Big\{ \frac{1}{2} \mathbf{R}'_j - \frac{1}{2} (\mathbf{x}^n - \boldsymbol{\mu}'_j)(\mathbf{x}^n - \boldsymbol{\mu}'_j)^T + \mathbf{W}'_j \langle\, \mathbf{z}^n_j \,\rangle (\mathbf{x} - \boldsymbol{\mu}'_j)^T \\
& - \frac{1}{2} \mathbf{W}'_j \langle\, \mathbf{z}^n_j (\mathbf{z}^n_j)^T \,\rangle (\mathbf{W}'_j)^T \Big\} = 0.
\end{aligned}$$

Using the new estimate (2.48) $\mathbf{W}'_j$ and the diagonal constraint on the noise model $\mathbf{R}_j$ (Rubin and Thayer 1982) we obtain:

$$\mathbf{R}'_j = \frac{1}{\sum_n h_j(\mathbf{x}^n)} \mathrm{diag} \left[\sum_n h_j(\mathbf{x}^n) \left\{ (\mathbf{x}^n - \boldsymbol{\mu}'_j) - \mathbf{W}'_j \langle\, \mathbf{z}^n_j \,\rangle \right\} (\mathbf{x}^n - \boldsymbol{\mu}'_j)^T \right]. \tag{2.49}$$

And this ends the derivation of the EM algorithm for MFAs (Algorithm 3 on the following page). The EM algorithm for a mixture of latent variable models thus consists of iteratively performing:

→ first stage E-step: (2.29);

→ first stage M-step: reestimate α_j (2.31) and centers $\boldsymbol{\mu}_j$ (2.32);

→ second stage E-step: compute the posteriors (2.42) and (2.43) using the new estimates for $\boldsymbol{\mu}_j$;

→ second stage M-step: reestimate $\mathbf{W}_j$ (2.48) and $\mathbf{R}_j$ (2.49).

Algorithm 3 EM algorithm for MFAs

Require:

- A training set of N examples: $\{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$.
- Error function: $E = -\sum_n \ln \sum_{j=1}^m \alpha_j p_j(\mathbf{x}^n)$ with $p_j(\mathbf{x}) \sim \mathcal{N}(\boldsymbol{\mu}_j, \mathbf{R}_j + \mathbf{W}_j\mathbf{W}_j^T)$
- $\mathbf{W}_j : d\times\ell$ $\mathbf{R}_j : d\times d$ and diagonal $\mathbf{N}_j : \ell\times\ell$ $\langle \mathbf{z}_j^n \rangle : \ell\times 1$ $\langle \mathbf{z}_j^n(\mathbf{z}_j^n)^T \rangle : \ell\times\ell$

loop
  {First stage: E-step}
  for $j := 1$ to m **do**
    for $n := 1$ to N **do**
      $h_j(\mathbf{x}^n) := \frac{\alpha_j p_j(\mathbf{x}^n)}{\sum_{i=1}^m \alpha_i p_i(\mathbf{x}^n)}$
    end for
  end for
  {First stage: M-step for $\boldsymbol{\alpha}$ and $\boldsymbol{\mu}$}
  for $j := 1$ to m **do**
    $\alpha_j := \frac{1}{N}\sum_n h_j(\mathbf{x}^n)$
    $\boldsymbol{\mu}_j := \frac{\sum_n h_j(\mathbf{x}^n)\mathbf{x}^n}{\sum_n h_j(\mathbf{x}^n)}$
  end for
  {Second stage: E-step}
  for $j := 1$ to m **do**
    $\mathbf{N}_j := \mathbf{I}_\ell + \mathbf{W}_j^T\mathbf{R}_j^{-1}\mathbf{W}_j$
    for $n := 1$ to N **do**
      $\langle \mathbf{z}_j^n \rangle := \mathbf{N}_j^{-1}\mathbf{W}_j^T\mathbf{R}_j^{-1}(\mathbf{x}^n - \boldsymbol{\mu}_j)$
      $\langle \mathbf{z}_j^n(\mathbf{z}_j^n)^T \rangle := \mathbf{N}_j^{-1} + \langle \mathbf{z}_j^n \rangle\langle \mathbf{z}_j^n \rangle^T$
    end for
  end for
  {Second stage: M-step for $\mathbf{W}$ and $\mathbf{R}$}
  for $j := 1$ to m **do**
    $\mathbf{W}_j := \left[\sum_n h_j(\mathbf{x}^n)(\mathbf{x}^n - \boldsymbol{\mu}_j)\langle \mathbf{z}_j^n \rangle^T\right]\left[\sum_n h_j(\mathbf{x}^n)\langle \mathbf{z}_j^n(\mathbf{z}_j^n)^T \rangle\right]^{-1}$
    $\mathbf{R}_j := \frac{1}{\sum_n h_j(\mathbf{x}^n)}\mathrm{diag}\left[\sum_n h_j(\mathbf{x}^n)\left\{(\mathbf{x}^n - \boldsymbol{\mu}_j) - \mathbf{W}_j\langle \mathbf{z}_j^n \rangle\right\}(\mathbf{x}^n - \boldsymbol{\mu}_j)^T\right]$
  end for
  {Stage 2 minimizes $\mathcal{E}(\hat{E}_c)$ and, therefore decreases $\mathcal{E}(E_c)$ which decreases the error function E}
end loop

We already know that a mixture of latent variable models offers an efficient alternative to full GMMs in terms of the number of parameters, but what about its computational complexity? Let us analyze the complexity of the algorithm step by step. The most intricate part is the calculation of the posteriors $h_j(\mathbf{x}^n)$, which requires the evaluation of the component densities:

$$p_j(\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}_j, \mathbf{R}_j + \mathbf{W}_j\mathbf{W}_j^T) = \frac{1}{|\mathbf{M}_j|^{1/2}(2\pi)^{d/2}} \exp\left\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_j)^T\mathbf{M}_j^{-1}(\mathbf{x} - \boldsymbol{\mu}_j)\right\}. \qquad (2.50)$$

This seems to involve calculating the inverse and determinant of the $d \times d$ matrix $\mathbf{M}_j$. However, as before, $\mathbf{M}_j^{-1}$ can be rewritten using the matrix inversion lemma (see (2.37)):

$$\mathbf{M}_j^{-1} = \mathbf{R}_j^{-1} - \mathbf{R}_j^{-1}\mathbf{W}_j\mathbf{N}_j^{-1}\mathbf{W}_j^T\mathbf{R}_j^{-1},$$

which only requires the inversion of a $\ell \times \ell$ matrix which is $O(\ell^3)$ and the easy inversion of the diagonal matrix $\mathbf{R}_j$. Actually, just calculating $\mathbf{N}_j = \mathbf{I}_\ell + \mathbf{W}_j^T\mathbf{R}_j^{-1}\mathbf{W}_j$ is a more complex operation of $O(\ell^2 d)$. When substituting this result in (2.50) care should be taken in choosing an efficient parsing for the multiplications:

$$(\mathbf{x}-\boldsymbol{\mu}_j)^T\mathbf{M}_j^{-1}(\mathbf{x}-\boldsymbol{\mu}_j) = (\mathbf{x}-\boldsymbol{\mu}_j)^T\mathbf{R}_j^{-1}(\mathbf{x}-\boldsymbol{\mu}_j) - \left\{(\mathbf{x}-\boldsymbol{\mu}_j)^T\mathbf{R}_j^{-1}\mathbf{W}_j\right\}\mathbf{N}_j^{-1}\left\{\mathbf{W}_j^T\mathbf{R}_j^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)\right\}.$$

This is an $O(m\ell dN)$ operation since it has to be done for each data point and each mixture component. Furthermore, the determinant $|\mathbf{M}_j|$ can also be simplified with the determinant factoring lemma (A.9):

$$|\mathbf{M}_j| = |\mathbf{R}_j + \mathbf{W}_j\mathbf{W}_j^T| = |\mathbf{R}_j||\mathbf{I}_\ell + \mathbf{W}_j^T\mathbf{R}_j^{-1}\mathbf{W}_j| = |\mathbf{R}_j||\mathbf{N}_j|,$$

which only necessitates a determinant of a $\ell \times \ell$ matrix, which is $O(\ell^3)$, and the easy determinant of the diagonal matrix $\mathbf{R}_j$.

It is easy to verify that the other operations are of the same order of complexity as the ones considered until now. This is especially so for the update of $\mathbf{R}_j$ which does not require the calculation of the sample covariance but only the variance for each coordinate, because of the "diag" operator. The EM algorithm for MFAs with m components, therefore, has computational complexity of $O(m\ell^2 d) + O(m\ell dN)$ in each iteration. Recall from section 2.2.1 that the complexity of EM for full GMMs is $O(md^3) + O(md^2N)$. MFAs give a speed-up of $(d/\ell)^2$ on the first term and (d/ℓ) on the second term. Both in terms of the number of parameters and of computational complexity, MFAs smoothly cover the range between diagonal and full covariance matrices in a GMM.

A Few Remarks on Mixtures of Principal Component Analyzers

The transformation of the algorithm for MFAs to one for MPCAs is quite easy: substituting $\mathbf{R}_j = \sigma_j^2$ in Algorithm 3 and some algebra leads to Algorithm 4 on the next page which is identical to the one of Tipping and Bishop (1999). The interested reader can find the details of this transformation in Appendix E. The computational complexity of this algorithm is also $O(m\ell^2 d) + O(m\ell dN)$. This is not that straightforward to see since I formulated the algorithm in terms of a weighted local covariance matrix $\mathbf{S}_j$ for each component. Calculating these covariance matrices would be an $O(d^2N)$ operation, but in this case it can be done more efficiently: $\mathbf{S}_j$ is used only in $\mathbf{S}_j\mathbf{W}_j$ and in $\mathrm{tr}(\mathbf{S}_j)$. The latter is a straightforward $O(dN)$ operation and the calculation of $\mathbf{S}_j\mathbf{W}_j$ can be done in $O(\ell dN)$ with a smarter parsing:

$$\mathbf{S}_j\mathbf{W}_j = \sum_n (\mathbf{x}^n - \boldsymbol{\mu}_j)\left\{(\mathbf{x}^n-\boldsymbol{\mu}_j)^T\mathbf{W}_j\right\}.$$

The advantage of the explicit formulation in terms of local covariance matrices is that it can be shown that each mixture component performs a local PCA, where each data point is weighted by the responsibility of that component (Tipping and Bishop 1999). This has motivated the use of MPCAs as a model for local linear dimensionality reduction and visualization (Bishop and Tipping 1998).

2.5 Incremental EM Algorithm for PCA

Let us now go back to PPCA, that is MPCA with just one component density. As discussed in section 2.3, Tipping and Bishop (1999) have shown that for the PPCA model a closed-form solution

Algorithm 4 EM algorithm for MPCAs.

Require:

- see Algorithm 3 but with $\mathbf{R}_j = \sigma_j^2 \mathbf{I}_d$

loop
  {First stage: E-step}
  for $j := 1$ to m **do**
    $h_j(\mathbf{x}^n) := \frac{\alpha_j p_j(\mathbf{x}^n)}{\sum_{i=1}^{m} \alpha_i p_i(\mathbf{x}^n)}$
  end for
  {First stage: M-step}
  for $j := 1$ to m **do**
    $\alpha_j := \frac{1}{N} \sum_n h_j(\mathbf{x}^n)$
    $\boldsymbol{\mu}_j := \frac{\sum_n h_j(\mathbf{x}^n)\mathbf{x}^n}{\sum_n h_j(\mathbf{x}^n)}$
  end for
  {Second stage}
  for $j := 1$ to m **do**
    {Local covariance matrix}
    $\mathbf{S}_j := \frac{1}{\alpha_j N} \sum_n h_j(\mathbf{x}^n)(\mathbf{x}^n - \boldsymbol{\mu}_j)(\mathbf{x}^n - \boldsymbol{\mu}_j)^T$
    $\mathbf{N}_j := \sigma_j^2 \mathbf{I}_\ell + \mathbf{W}_j^T \mathbf{W}_j$
    $\mathbf{W}_j, \mathbf{W}_{\text{old},j} := \mathbf{S}_j \mathbf{W}_j (\sigma_j^2 \mathbf{I}_\ell + \mathbf{N}_j^{-1} \mathbf{W}_j^T \mathbf{S}_j \mathbf{W}_j)^{-1}, \mathbf{W}_j$
    $\sigma_j^2 := \frac{1}{d} \mathrm{tr}(\mathbf{S}_j - \mathbf{W}_j \mathbf{N}_j^{-1} \mathbf{W}_{\text{old},j}^T \mathbf{S}_j)$
  end for
end loop

exists for the global maximum of the likelihood and that this is the only stable maximum of the likelihood surface. The closed-form solution (2.24) involves computing the sample covariance matrix, which is $O(d^2 N)$, and its eigendecomposition, which is $O(d^3)$. One might prefer the EM algorithm for PPCA with its computational complexity of $O(\ell d N) + O(\ell^2 d)$ and which also converges to the global maximum of the likelihood. The EM algorithm can offer a considerable speed-up over the closed-form solution if it does not take too many iterations to converge or if $\ell \ll d$.

PPCA trained with EM can also be an attractive way of doing standard PCA, for example for feature extraction (Roweis 1998). Of course, a host of other approaches exists for doing PCA. A general technique is to first compute the sample covariance matrix, which is $O(d^2 N)$, followed by any general method which solves a symmetric eigenvalue problem (Golub and Van Loan 1996). The more advanced methods are able to extract a specified number ℓ of principal eigenvectors and are in general $O(\ell d^2)$. One can avoid computing the sample covariance matrix and its typical problems with a small amount of data in a high-dimensional space, by computing the singular value decomposition of the $N \times d$ matrix containing the training data (Ripley 1996). A disadvantage of these techniques is that, in general, they require storing the entire covariance matrix or the entire data matrix. This has motivated the development of incremental techniques from the field of neural networks. A first approach is closely related to singular value decomposition and inspired by the fact that PCA minimizes the reconstruction cost (2.25). This can be mimicked by an auto-associative linear neural network with ℓ hidden neurons minimizing the sum-of-squares error function in an incremental way (Diamantaras and Kung 1996). Another class of methods use some form of Hebbian learning on a one-layer neural network to find the principal subspace (Diamantaras and Kung 1996).

The EM algorithm for PPCA combines several of the advantages of the above methods. It does not require computing the covariance matrix and is only $O(\ell dN)+O(\ell^2 d)$ when extracting the ℓ principal eigenvectors. The span of the generative matrix $\mathbf{W}$ is guaranteed to converge to the ℓ-dimensional principal subspace (see (2.24)). Moreover, the algorithm can be performed incrementally using only a single data point at a time as we will see below.

Since we are now interested in feature extraction and not in density modeling, we can make the noise variance infinitesimal $\sigma \to 0$ and end up with a concise EM algorithm for PCA (Roweis 1998). The reduction of the EM algorithm for MPCAs (Algorithm 4) in the single component case ($m=1$) and with infinitely small noise variance ($\sigma^2 \to 0$) is as follows. The first stage EM-step is simple and reduces to the sample mean (2.23) as estimate for $\boldsymbol{\mu}$ since $h_j(\mathbf{x}^n)=1$ for all n in the one-component case. The second stage requires a little more attention and I copy the equations needed here (leaving out the component indices):

$$\mathbf{N} = \sigma^2\mathbf{I}_\ell + \mathbf{W}^T\mathbf{W} \qquad\qquad \mathbf{W}' = \mathbf{SW}(\sigma^2\mathbf{I}_\ell + \mathbf{N}^{-1}\mathbf{W}^T\mathbf{SW})^{-1}.$$

For $\sigma^2 \to 0$ this can be simplified:

$$\mathbf{N} = \mathbf{W}^T\mathbf{W} \qquad\qquad \mathbf{W}' = \mathbf{SW}(\mathbf{N}^{-1}\mathbf{W}^T\mathbf{SW})^{-1}.$$

We start with the update of the generative matrix:

$$\mathbf{W}' = \mathbf{SW}(\mathbf{N}^{-1}\mathbf{W}^T\mathbf{SW})^{-1}.$$

Substituting the sample covariance matrix $\mathbf{S}$ (2.22) and using $\mathbf{N}=\mathbf{W}^T\mathbf{W}$:

$$= \Big[\sum_n(\mathbf{x}^n-\boldsymbol{\mu})(\mathbf{x}^n-\boldsymbol{\mu})^T\Big]\mathbf{W}\Big[(\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T\{\sum_n(\mathbf{x}^n-\boldsymbol{\mu})(\mathbf{x}^n-\boldsymbol{\mu})^T\}\mathbf{W}\Big]^{-1}.$$

Adding $\mathbf{W}^T\mathbf{W}$ and its inverse as factors (using that $(\mathbf{AB})^{-1}=\mathbf{B}^{-1}\mathbf{A}^{-1}$):

$$\mathbf{W}' = \Big[\sum_n(\mathbf{x}^n-\boldsymbol{\mu})(\mathbf{x}^n-\boldsymbol{\mu})^T\Big]\mathbf{W}(\mathbf{W}^T\mathbf{W})^{-1}\Big[(\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T\{\sum_n(\mathbf{x}^n-\boldsymbol{\mu})(\mathbf{x}^n-\boldsymbol{\mu})^T\}\mathbf{W}(\mathbf{W}^T\mathbf{W})^{-1}\Big]^{-1}.$$

Defining $\langle\,\mathbf{z}^n\,\rangle = (\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T(\mathbf{x}^n-\boldsymbol{\mu})$ (this really is (2.42) for $\mathbf{R}_j=\sigma_j^2\mathbf{I}_d\to 0$ in the E-step for training a MFA) gives the update of the generative matrix:

$$\mathbf{W}' = \Big[\sum_n(\mathbf{x}^n-\boldsymbol{\mu})\langle\,\mathbf{z}^n\,\rangle^T\Big]\Big[\sum_n\langle\,\mathbf{z}^n\,\rangle\langle\,\mathbf{z}^n\,\rangle^T\Big]^{-1}. \tag{2.51}$$

This gives us Algorithm 5 for PCA (the batch version in the left-hand column). The EM algorithm for PCA has an intuitive interpretation. The E-step can be seen as a linear projection of the data $\mathbf{x}^n$ onto our current guess for the principal subspace; this gives us the values of the latent variables $\mathbf{z}^n$. The M-step is the solution of a least-squares problem and minimizes the reconstruction cost of the data.

It is straightforward to transform the batch version of Algorithm 5 into an equivalent incremental[6] version in the right-hand column (Roweis 1998). The incremental variant does not need to store the entire data set and its storage requirements are only $O(\ell d)+O(\ell^2)$. Hebbian learning on a one-layer neural network has similar computational and storage complexity as EM for PCA. The EM algorithm has the advantage that it does not require choosing a suitable learning rate and that at each iteration the cost is minimized. The Hebbian rules have the advantage of being really on-line and can thus be applied when data varies over time. We will come back to this and other issues in section 2.8 where EM for PCA is applied to so-called *kernel* PCA. This is a non-linear form of PCA based on the kernel trick illustrated in Chapter 1 and has to solve an eigenvalue problem on a matrix of size $N\times N$. The EM algorithm for PCA can be used to do this without having to store the entire matrix.

[6]Note that I use a relatively weak definition of “incremental” since the parameters are updated only after having seen the whole data set.

Algorithm 5 EM algorithm for PCA (batch and incremental).

Require:

- A training set of N examples: $\{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$.

{Batch Algorithm}
$\boldsymbol{\mu} := \sum_n \mathbf{x}^n / N$
loop
  {E-step}
  for $n := 1$ to N **do**
    $\langle \mathbf{z}^n \rangle := (\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T(\mathbf{x}^n - \boldsymbol{\mu})$
  end for
  {M-step}
  $\mathbf{W} := \left[\sum_n (\mathbf{x}^n - \boldsymbol{\mu})\langle \mathbf{z}^n \rangle^T\right] \left[\sum_n \langle \mathbf{z}^n \rangle \langle \mathbf{z}^n \rangle^T\right]^{-1}$
end loop

{Incremental Algorithm}
$\boldsymbol{\mu} := \sum_n \mathbf{x}^n / N$
loop
  {E-step}
  $\mathbf{V} := (\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T$
  $\mathbf{A} := \mathbf{0}_{(d\times\ell)}; \mathbf{B} := \mathbf{0}_{(\ell\times\ell)}$
  for $n := 1$ to N **do**
    $\langle \mathbf{z}^n \rangle := \mathbf{V}(\mathbf{x}^n - \boldsymbol{\mu})$
    $\mathbf{a} := (\mathbf{x}^n - \boldsymbol{\mu})\langle \mathbf{z}^n \rangle^T; \mathbf{A} := \mathbf{A} + \mathbf{a}$
    $\mathbf{b} := \langle \mathbf{z}^n \rangle \langle \mathbf{z}^n \rangle^T; \mathbf{B} := \mathbf{B} + \mathbf{b}$
  end for
  {M-step}
  $\mathbf{W} := \mathbf{A}\mathbf{B}^{-1}$
end loop

2.6 Experiments: Density Estimation

We have come a long way and the presented models seem interesting but now time has come to let the data have its say. This section is dedicated to an experimental comparison of standard GMMs and mixtures of latent variable models and illustrates the merit of the latter in density modeling (Moerland 1999b).

Recent literature contains various experimental results for density modeling with mixtures of latent variable models, on isolated problems:

- a comparison of MPCAs and GMMs on an artificial data set (Tipping and Bishop 1999);
- a comparison of MFAs with various other methods (standard PCA, factor analysis, generative topographic mapping) on the specific problem of density modeling of electropalatographic data (Carreira-Perpiñán and Renals 1998);
- handwritten digit recognition MPCAs and MFAs with a mixture model for each class in a Bayes classifier (Hinton, Dayan, and Revow 1997; Tipping and Bishop 1999);
- face recognition with MFAs with a mixture model for each class in a Bayes classifier and the nearest neighbour rule (Frey, Colmenarez, and Huang 1998).

The goal of the experiments described in the rest of this section is to provide a more extensive comparison of GMMs and mixtures of latent variable models on a large set of benchmarks. Here, we consider only density estimation per se but we will come back to the inclusion of mixture models in Bayes classifiers later on (sections 3.1 and 4.3).

2.6.1 Experimental Set-Up

The density estimation experiments were mostly done with various data sets out of the Irvine repository (Blake, Keogh, and Merz 1998) and a subset of the digits "two" out of the NIST special database-3

Data set	# attributes		# classes	# examples		missing data	# attr.
	discr.	cont.		train	test		
banana	-	2	2	803	-		
cancer	-	10	2	699	-	•	10
dermatology	-	34	6	366	-	•	34
glass	-	9	6	214	-		
heart	-	13	2	303	-	•	13
ionosphere	-	34	2	351	-		
iris	-	4	3	150	-		
letter	-	16	26	20000	-		
NIST	-	256	10	15025	4975		
optical	-	64	10	3823	-		
pen	-	16	10	7494	-		
pima	-	8	2	768	-		
satimage	-	36	6	4435	2000		
segmentation	-	19	7	2310	-		
sonar	-	60	2	208	-		
soybean	35	-	19	683	-	•	134
twos	-	256	10	1948	-		
vowel	-	10	11	990	-		
waveform	-	21	3	600	-		
waveform-noise	-	40	3	600	-		

Table 2.1: Properties of the data sets used in the experiments. The last column gives the number of attributes after pre-processing missing data.

of handwritten digits (Garris and Wilkinson 1992). I limited myself to data sets for classification problems since that is what we will focus on in the following chapters and the same collection of benchmarks will be used there. Of course, for the current experiments only the input space of the data sets plays a role and the class labels are not taken into consideration. An overview of the main characteristics of the different data sets is given in Table 2.1. As can be seen from this table, the benchmarks largely differ in input dimension and number of patterns and cover a wide spectrum of data sets which one might encounter in practice. The reader is referred to Appendix C for more information on these and other benchmarks used in this thesis and on their availability.

The raw data has been pre-processed in various ways. This is not really necessary in the context of density estimation but it will be so in later chapters where neural networks are used. Thus, throughout my thesis data sets are always pre-processed in the same way. First of all, each of the ordinal and real-valued attributes has been normalized to have zero mean and unit standard deviation on the training data. For soybean, some of the attributes are categorical and these are mapped to a 1-of-c coding, thus increasing the number of attributes (see the last column of Table 2.1). Finally, for the data sets indicated with a •, some of the attributes are missing for some patterns. For ordinal and real-valued inputs, the missing value has been replaced by zero (the mean value after normalization) and for categorical inputs an extra bit was added to the 1-of-c coding to encode a missing value. Of course, this is not the best strategy to follow (especially for mixture models trained by EM) (Ghahramani and Jordan 1994): it was mainly chosen for its simplicity. The pre-processing of the NIST and twos data consisted of centering, normalizing, and smoothing the original patterns to obtain a 16×16 representation.

The first class of models compared are GMMs with the various choices of a covariance matrix described in section 2.2.1: spherical, diagonal, full, and full but tied across the mixture components. From the class of mixtures of latent variable models both MFAs and MPCAs were used. The training of all mixture models consisted of an initialization phase and 15 iterations of the EM algorithm. The initialization of each of the mixture models used k-means clustering (Bishop 1995) to determine the centers. The mixing coefficients were then computed from the proportion of examples belonging (closest) to each cluster.

Covariance matrices for the initialization of the GMMs were calculated based on the sample covariance of the points associated with (that is, closest to) the corresponding centers. The penalized likelihood approach described in section 2.2.1 was used for training full GMMs with a fixed (sub-optimal) value for $\beta = 0.05$.[7] Furthermore, a small threshold was imposed upon the singular values of the covariance matrices to avoid non-singularities and mixture components which collapse to covering only a single data point.

The initialization of the generative matrices $\mathbf{W}_j$ of the mixtures of latent variable models was done using (2.24) via a few iterations of EM for PCA on the points associated with the corresponding centers. The noise models $\mathbf{R}_j$ were initialized using the variance lost in the PCA projections found for each cluster using (2.26). For the MFAs, I decided to use a single noise model $\mathbf{R}$ which is tied across the mixture components. This single diagonal matrix corresponds to the intuitive idea behind factor analysis of data corrupted by sensor noise: such noise would be similarly distributed for all data and should not vary across the mixture components. A simple form of regularization was used by imposing a small threshold upon the values of $\mathbf{R}_j$.

The number of iterations of the EM algorithm was chosen to be 15 because this very often turned out to be sufficient for approaching a local minimum on the training set. This might be due to the initialization which already positions the mixture components rather well. Of course, there might be far better local minima and there are methods for escaping from a "bad" local minimum but that will not be our concern here.

The 5x2cv F test (Alpaydin 1999; Dietterich 1998a) has been used on all data sets for testing whether a difference in performance between the methods was statistically significant. In this test, five replications of twofold cross-validation are performed: each training set and each test set comprises 50% of the data. This was also done for the satimage data which, in principle, comes with a fixed division in a training and a test set (see Table 2.1). The test is explained in detail in Appendix B. In each round of cross-validation, one third of the training data was set aside for validation purposes (respecting the class distributions). The results on the validation sets were used to select the best model for each class of mixture models on each data set. The free parameters varied in each class of models were the number of components (at least 5 different values where used) and, for MPCAs and MFAs, the dimension of latent space.

2.6.2 Evaluation on Artificial Data

As a first test, experiments were performed on two often used artificial classification problems with code for generating the data at the Irvine repository (Blake, Keogh, and Merz 1998): waveform and waveform-noise (the last two rows of Table 2.1). The waveform data is generated according to:

$$\begin{aligned} x_i &= uh_1(i) + (1-u)h_2(i) + \varepsilon_i && \text{Class 1} \\ x_i &= uh_1(i) + (1-u)h_3(i) + \varepsilon_i && \text{Class 2} \\ x_i &= uh_2(i) + (1-u)h_3(i) + \varepsilon_i && \text{Class 3,} \end{aligned}$$

where $i = 1, 2, \ldots 21$, u is a random variable, uniformly distributed on $(0, 1)$, $\varepsilon_i \sim \mathcal{N}(0, \mathbf{I})$, and the h_i are shifted triangular waveforms: $h_1(i) = \max(6 - |i - 11|, 0)$, $h_2(i) = h_1(i-4)$, and $h_3(i) = h_1(i+4)$. For waveform-noise, the 19 additional attributes are all noise attributes with mean 0 and variance 1.

[7] Ideally different values should have been tried selecting the best value on validation data.

Mixture model	train	test	5x2cv
Full	22.4(0.18)	26.1(0.34)	
Spherical	25.4(0.08)	25.7(0.28)	
Diagonal	24.9(0.10)	25.3(0.26)	$<$
MPCA-1	24.6(0.14)	25.2(0.31)	
MPCA-3	24.0(0.15)	25.1(0.27)	
MFA-3	23.5(0.17)	24.4(0.24)	$<$
MFA-1	23.8(0.19)	24.3(0.25)	$<$

Table 2.2: Density modeling on waveform with 3 mixture components. Scores are in average negative log-likelihood (the smaller the better). The entries are the averages of the negative log-likelihood per data point over 10 cross-validated simulations with the standard deviation in parentheses. A $<$-sign indicates whether the score on the test set is significantly better (95%) than the one on the previous row. MFA-ℓ and MPCA-ℓ denote a mixture of latent variable models with ℓ factors.

Mixture model	train	test	5x2cv
Full	45.4(0.27)	60.1(0.69)	
MPCA-3	51.5(0.14)	53.7(0.50)	$<$
Spherical	52.8(0.10)	53.4(0.48)	$<$
MPCA-1	52.3(0.12)	53.4(0.48)	
Diagonal	51.5(0.13)	52.4(0.48)	$<$
MFA-3	50.0(0.19)	51.9(0.50)	$<$
MFA-1	50.7(0.19)	51.7(0.51)	$<$

Table 2.3: Density modeling on waveform-noise with three mixture components. Scores are in average negative log-likelihood.

The results on waveform and waveform-noise are in Tables 2.2 and 2.3. The GMM with a full covariance matrix obtains the best likelihood on the training set but the worst score on the test set: the model is overfitting the data due to its many parameters. For all the other models the score on the test is only slightly worse than the score on the training set which suggests the absence of overfitting. On waveform (Table 2.2), the best results are obtained with the mixtures of latent variable models. It is especially worth noting that the MFA model with only one factor performs much better than the GMM with a diagonal covariance matrix which has about the same number of free parameters. This shows that the axial alignment constraint of the diagonal model is not appropriate in this case. On waveform-noise (Table 2.3), the best results are again obtained with MFAs, but MPCAs are not performing that good. This is most likely due to the fact that the 19 additional inputs for this data set are just white noise and MFAs can separately model correlations and variance ($\mathbf{R}_j$ is diagonal), whereas MPCAs cannot ($\mathbf{R}_j$ is isotropic).

It is also interesting to investigate the influence of the number of parameters in a mixture model. Figure 2.5 illustrates that for a fixed number of parameters, GMMs with diagonal covariance matrices, varying the number of mixture components from 2 to 10, were always outperformed by MFAs, varying the number of factors and mixture components, on waveform-noise.

A second example is the noisy shrinking spiral data of Figure 2.6 which has been used in some recent papers on MFAs (Ghahramani and Beal 2000; Ueda, Nakano, Ghahramani, and Hinton 1999). This toy data set nicely illustrates some of the short-comings of the GMMs.

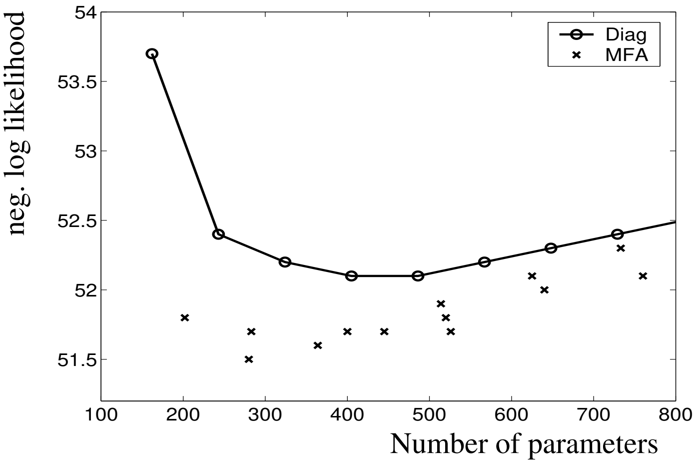
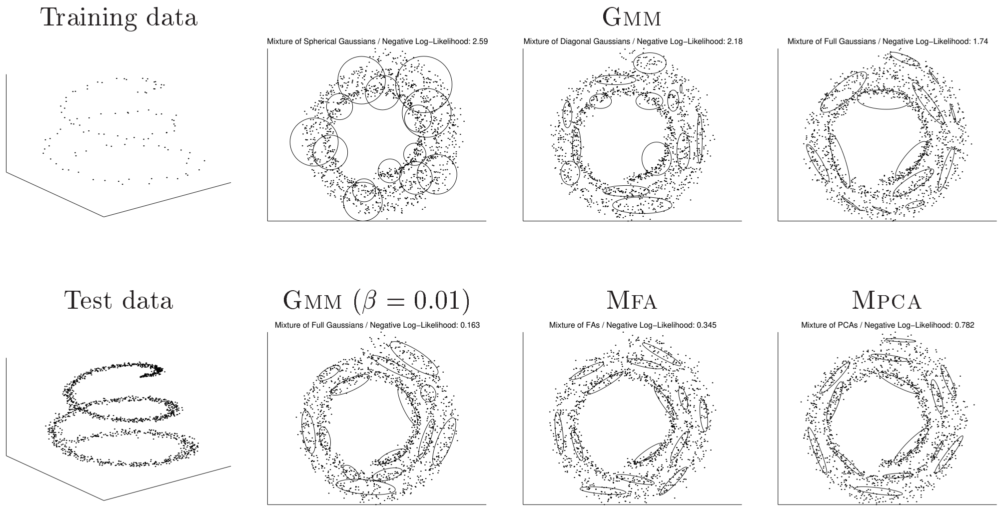


Figure 2.5: Comparison of a GMM with diagonal covariance matrices and a MFA on waveform-noise: the average negative log-likelihood on the test set versus the number of parameters.



	GMM					
	spherical	diagonal	full	full($\beta = 0.01$)	MFA	MPCA
Train	2.56	1.63	-0.92	-0.20	-0.19	-0.54
Test	2.69	2.14	1.99	0.21	0.41	0.85

Figure 2.6: Noisy shrinking spiral data with a training set of 100 examples and a test set of 1500 examples. All mixture models have 14 components. The dimension of latent space for the MPCA and the MFA is $\ell = 1$. The values in the table are the average negative log-likelihood of the test data. Results are the mean over 20 random initializations of the data. The mixture models are shown in the $x-y$ plane with ellipses at a distance of one standard deviation.

Spherical and diagonal covariance matrices are clearly too constraint. The introduction of more mixture components did not help either since this reduced the training error but not the test error. A full GMM has too many parameters for the small number of examples in the training set: this leads to the lowest training error but a high test error. A regularized full GMM ($\beta = 0.01$) performs much better and actually outperforms the MFA and MPCA models (although I did not test statistical significance here). Note that I selected the value of β by "cheating" since it was chosen based on its performance on the test set. MFAs and MPCAs show good performance with a dimension of latent space $\ell = 1$ without any further regularization.

2.6.3 Evaluation on Real-World Data

Do the good results for mixtures of latent variable models on both artificial data sets carry over to real-world data? To answer this question, experiments have been performed on the data sets listed in Table 2.1. The results are shown in Table 2.4, where the best method and the ones that are not significantly worse (95% on the 5x2cv F test) are set underlined. Note that on the dermatology and glass data, the results do not allow any conclusion about the relative performance of the different models. The results also show the power of using a *paired* test: in spite of sometimes large standard deviations, significant differences can still be detected.

For most data sets the number of mixture components tried, had at least five different values ranging from one to a value depending on the complexity of the problem but with a maximum of 20. The dimension of latent space for MPCAs and MFAs was varied in a similar way but with a maximum value of 15 (and of course upper-bounded by the dimension of data space).

The last row in Table 2.4 gives an immediate idea of the global performance of the different models by specifying the total number of underlined scores for each model class. This number of wins shows that MFAs provide the best density estimations. They are outperformed on only two data sets (cancer and ionosphere) out of 18. With respect to the spherical GMMs, the score of only 1 win illustrates that they are too constrained to model the data. From the results, one can also indirectly conclude that using full covariance matrices makes the model sensitive to overfitting: the best model has only one mixture component in about half of the cases. This effect can be reduced by tying the covariance matrices across the mixture components but this also considerably reduces modeling power: there is only one win for tied GMMs. When data is plentiful, however, full GMMs might still be among the best models. Diagonal GMMs are performing quite well on a few data sets; this is most impressive on the optical data which is hand-written digit data with discretized attributes. Some of its attributes are zero throughout the whole data set and a diagonal GMM is well-suited for capturing this kind of absence of noise. It might also explain why the scores of the MPCAs are not as good as with MFAs: the single variance parameter for each mixture component makes it more difficult for MPCAs to separate information and noise. This is illustrated by the mediocre performance on the optical data. MPCAs do not succeed in dealing with the varying amounts of noise in the different attributes.

We can conclude that mixtures of latent variable models and especially MFAs, are indeed a flexible alternative to standard Gaussian mixture models, the complexity of which can be tuned by varying the dimension of the latent space. Moreover, their computational complexity also scales favorably with the dimension of latent space and places them on a spectrum going from diagonal GMMs to full GMMs.

2.7 Bayesian Methods for Latent Variable Models

One of the limitations of the maximum likelihood framework to which we have adhered until now is that it does not take into account model complexity. Therefore, we selected the number of mixture components and factors by measuring the performance of each model on validation data. Actually,

Data	Gmm				Mpca	Mfa
	spherical	diagonal	tied	full		
banana	$2.7(0.04)^{6}$	$\underline{2.6}(0.10)^{6}$	$2.7(0.04)^{6}$	$\underline{2.6}(0.09)^{6}$	$\underline{2.6}(0.08)^{6,1}$	$\underline{2.6}(0.10)^{6,1}$
cancer	$7.3(1.09)^{6}$	$\underline{1.5}(0.97)^{6}$	$11.1(0.95)^{2}$	$5.5(1.12)^{6}$	$\underline{3.7}(1.43)^{6,5}$	$6.1(0.98)^{2,5}$
dermatology	$39.1(0.92)^{6}$	$36.5(4.94)^{4}$	$40.2(1.07)^{2}$	$40.7(1.25)^{1}$	$37.9(0.66)^{6,1}$	$33.4(9.42)^{4,5}$
glass	$10.1(1.79)^{4}$	$10.8(3.62)^{4}$	$11.5(4.39)^{1}$	$11.7(4.38)^{1}$	$11.1(3.62)^{4,1}$	$13.1(4.93)^{1,5}$
heart	$17.9(0.26)^{6}$	$\underline{16.4}(0.74)^{2}$	$\underline{15.4}(0.95)^{6}$	$18.2(0.21)^{1}$	$18.0(0.24)^{2,1}$	$\underline{14.5}(1.38)^{6,1}$
ionosphere	$31.9(2.20)^{8}$	$\underline{27.2}(1.53)^{8}$	$62.1(9.21)^{2}$	$65.3(6.17)^{1}$	$\underline{29.4}(2.20)^{8,1}$	$37.5(1.72)^{1,5}$
iris	$4.0(0.31)^{4}$	$3.4(0.23)^{2}$	$2.8(0.27)^{4}$	$\underline{2.5}(0.20)^{2}$	$\underline{2.6}(0.31)^{2,3}$	$\underline{2.7}(0.25)^{2,2}$
letter	$18.3(0.11)^{20}$	$16.0(0.18)^{20}$	$16.0(0.14)^{20}$	$\underline{11.4}(0.29)^{20}$	$12.0(0.14)^{20,15}$	$\underline{11.6}(0.50)^{20,15}$
optical	$66.3(1.01)^{20}$	$-\underline{11.6}(1.48)^{20}$	$53.0(3.12)^{20}$	$8.5(2.60)^{6}$	$46.1(6.55)^{20,15}$	$-\underline{12.5}(1.71)^{10,15}$
pen	$13.8(0.17)^{20}$	$7.7(0.28)^{20}$	$10.6(0.24)^{20}$	$3.3(0.47)^{20}$	$3.9(0.35)^{20,15}$	$-\underline{4.7}(0.74)^{20,15}$
satimage	$13.2(0.37)^{20}$	$11.5(0.24)^{20}$	$-0.6(0.14)^{20}$	$-\underline{4.6}(0.37)^{6}$	$-3.1(0.43)^{10,15}$	$-\underline{4.4}(0.36)^{6,15}$
segmentation	$13.3(0.65)^{10}$	$-6.0(2.36)^{10}$	$-7.2(1.35)^{8}$	$-14.2(2.92)^{8}$	$-\underline{7.6}(7.69)^{10,8}$	$-\underline{20.6}(4.66)^{10,8}$
sonar	$\underline{76.7}(1.92)^{4}$	$79.0(1.97)^{4}$	$195.5(9.93)^{1}$	$197.3(8.56)^{1}$	$\underline{73.5}(3.11)^{1,5}$	$\underline{68.9}(4.54)^{1,15}$
soybean	$16.2(2.23)^{6}$	$-167.0(9.76)^{6}$	$-173.5(5.84)^{6}$	$-\underline{174.7}(8.85)^{2}$	$-13.2(9.40)^{4,8}$	$-\underline{196.7}(6.85)^{6,8}$
twos	$-109.1(1.99)^{20}$	$-186.7(3.00)^{20}$	$-295.2(1.88)^{1}$	$-295.1(2.20)^{1}$	$-286.7(3.82)^{4,15}$	$-\underline{323.6}(7.19)^{4,15}$
vowel	$12.4(0.14)^{11}$	$12.3(0.14)^{11}$	$11.7(0.17)^{11}$	$\underline{11.5}(0.22)^{4}$	$\underline{11.5}(0.31)^{11,3}$	$\underline{11.5}(0.39)^{11,5}$
waveform	$25.3(0.30)^{6}$	$24.9(0.28)^{6}$	$24.9(0.22)^{2}$	$24.9(0.25)^{1}$	$24.8(0.27)^{1,10}$	$\underline{24.3}(0.25)^{4,1}$
waveform-noise	$53.5(0.49)^{4}$	$52.5(0.49)^{4}$	$54.1(0.54)^{1}$	$54.1(0.54)^{1}$	$53.5(0.46)^{2,1}$	$\underline{51.6}(0.51)^{1,3}$
wins	1	5	1	6	7	14

Table 2.4: Input density modeling with Gmms, Mfas, and Mpcas. Scores are in average negative log-likelihood on the test set and are the average over 10 experiments in a 5x2cv F test set-up. The standard deviation is given between parentheses. Each score corresponds to the best model out of the particular class of models as evaluated on a validation set. The first superscript indicates the number of mixture components in the "best". The second superscript for mixtures of latent variable models specifies the dimension of latent space. The underlined scores are the ones that do not pass the 5x2cv F test with 95% confidence when compared with the model having the best score.

using mixtures of latent variable models with the *same* dimension of latent space for each of the mixture components is simply a way of keeping this discrete model search more tractable. One would certainly expect to improve performance by allowing each of the mixture components to have its proper dimensionality. In this section, I present a Bayesian way of selecting these component dimensionalities automatically as proposed by Bishop (1999a) for Pca. The Bayesian approach is introduced by showing how regularization can be interpreted in probabilistic terms. The evidence framework of MacKay (1991) for keeping Bayesian inference tractable is also briefly explained. This approach is used to derive a novel Bayesian treatment of factor analysis which can automatically determine the effective dimension of latent space. The resulting algorithm is an extension of the previously derived Em algorithm. Bishop's (1999a) Bayesian Pca is derived as a special case of Bayesian Fa. It is shown that Bayesian Fa is computationally more complex than Bayesian Pca but that it remains tractable when implemented carefully. Both methods can be readily extended to mixtures of Fa and Pca. I conclude this section with a series of experiments on toy data and with the first experimental results on a large set of real-world benchmarks. These experiments show the viability of the Bayesian approach as a way of avoiding overfitting and selecting the dimensionality (or dimensionalities when dealing with mixture models) of latent space.

2.7.1 From Regularization to the Evidence Framework

An approach to control the effective complexity of a model is the use of *regularization* which involves the addition of a term to the error function in order to encourage simpler mappings. This can be used as a means to still obtain good generalization with models that are actually too complex for the data. The regularized error function thus consists of a data term which we assume to be the negative log-likelihood as before and a penalty term E_P:

$$E = -\ln p(D|\boldsymbol{\theta}) + \gamma E_P(\boldsymbol{\theta}). \tag{2.52}$$

The parameter γ controls the influence of the penalty term on the total error function. A well-known example of regularization is known as *weight decay* or *ridge regression*: $E_P(\boldsymbol{\theta}) = \frac{1}{2}||\boldsymbol{\theta}||^2$. This regularizer favors a smoother mapping by penalizing large parameter values which are often an indication for overfitting.

The inclusion of such a regularizer simplifies the problem of model selection since we can now choose a rather complex model and rely on the penalty term to control the effective complexity. The complexity control is now embodied by the single regularization parameter γ. An appropriate value for γ could again be found by using validation data.[8] This becomes unwieldy when more complex regularizers are used with several regularization parameters. Such a complex regularizer is exactly what we need to control the dimension of latent space in PPCA and FA. Remember that each dimension of latent space, with $\ell = d-1$, corresponds to a column of the generative matrix $\mathbf{W} = (\mathbf{w}_1 \ \ldots \ \mathbf{w}_{d-1})$. Bishop (1999a), therefore, proposes to use a separate regularization term of the weight decay type for each column of $\mathbf{W}$:

$$\frac{1}{2}\sum_{i=1}^{d-1} \gamma_i ||\mathbf{w}_i||^2, \tag{2.53}$$

with regularization parameters $\boldsymbol{\gamma} = (\gamma_1 \ldots \gamma_{d-1})$. This means that the value of γ_i controls the norm of column $\mathbf{w}_i$ and is even able to switch it off entirely by forcing the corresponding weights to zero. This regularization term is motivated by the framework of *automatic relevance determination* (ARD) as proposed by MacKay and Neal (see, for example, MacKay 1994b). They used it for controlling the relevance of inputs of a multi-layer perceptron (MLP).

This ARD term involves $d-1$ regularization parameters and as already observed before, selecting their values using validation data would be cumbersome. A principled way of handling regularization on the training data only is the application of Bayesian inference techniques (Bishop 1995; MacKay 1991). The essence of Bayesian inference is that instead of a single set of values for the parameters as in maximum likelihood estimation, a probability distribution over the parameter space is considered. This can be motivated by interpreting (2.52) entirely in probabilistic terms using Bayes' rule (A.18):

$$E = -\ln p(\boldsymbol{\theta}|D, \boldsymbol{\gamma}) = -\ln \left[\frac{p(D|\boldsymbol{\theta})p(\boldsymbol{\theta}|\boldsymbol{\gamma})}{p(D|\boldsymbol{\gamma})}\right] \propto -\ln p(D|\boldsymbol{\theta}) - \ln p(\boldsymbol{\theta}|\boldsymbol{\gamma}). \tag{2.54}$$

The weight decay regularizer $\frac{1}{2}\gamma||\boldsymbol{\theta}||^2$ then corresponds directly to a so-called *prior* distribution for the parameters:

$$p(\boldsymbol{\theta}|\boldsymbol{\gamma}) = \mathcal{N}(\boldsymbol{\theta}|\mathbf{0}, 1/\gamma) \propto \exp(-\frac{\gamma||\boldsymbol{\theta}||^2}{2}).$$

Thus, minimization of the regularized error function (2.52) is equivalent to finding the *most probable* parameter values $\boldsymbol{\theta}_{\mathrm{MP}}$ maximizing the posterior distribution $p(\boldsymbol{\theta}|D, \boldsymbol{\gamma})$. This is a nice probabilistic interpretation of the error function but the real strength of the Bayesian framework only emerges when going to a higher level of inference.

[8] Of course, it cannot be determined on the training data only since the optimum value would be zero.

At the second level of inference, we do not consider the regularization parameters (or *hyperparameters*) fixed but also try to adapt them within the Bayesian framework. We start with the posterior distribution of the hyperparameters and turn the Bayesian crank:

$$p(\boldsymbol{\gamma}|D) = \frac{p(D|\boldsymbol{\gamma})p(\boldsymbol{\gamma})}{p(D)}. \tag{2.55}$$

The data-dependent term $p(D|\boldsymbol{\gamma})$ is the normalizing constant of the posterior $p(\boldsymbol{\theta}|D,\boldsymbol{\gamma})$ on the previous level of inference (2.54) and it is called the *evidence* for the hyperparameters $\boldsymbol{\gamma}$. It is assumed that we only have very weak prior knowledge about the hyperparameters and that we do not consider the *hyperprior* $p(\boldsymbol{\gamma})$. We now take the evidence (MacKay 1991) or type-II maximum likelihood (Berger 1985, page 99) approach and determine the most probable values for $\boldsymbol{\gamma}$ maximizing the posterior $p(\boldsymbol{\gamma}|D)$.[9] Ignoring the hyperprior and the denominator of (2.55) which does not depend on $\boldsymbol{\gamma}$, this boils down to maximizing the evidence $p(D|\boldsymbol{\gamma})$.

Maximizing the evidence can be done by expressing it in terms of the already defined likelihood and prior (using the sum and the product rule):

$$\text{evidence} = p(D|\boldsymbol{\gamma}) = \int p(D|\boldsymbol{\theta},\boldsymbol{\gamma})p(\boldsymbol{\theta}|\boldsymbol{\gamma})d\boldsymbol{\theta} = \int p(D|\boldsymbol{\theta})p(\boldsymbol{\theta}|\boldsymbol{\gamma})d\boldsymbol{\theta}.$$

The above integral can sometimes be handled analytically, for example when both likelihood and prior are Gaussian and quadratic in the parameters $\boldsymbol{\theta}$. In general, however, it will require Monte Carlo methods (Neal 1996) or well-chosen approximations. Again we follow in the footsteps of MacKay (1991) and approximate the integrand by a Gaussian distribution around the most probable parameter values $\boldsymbol{\theta}_{\mathrm{MP}}$. This leads to an analytical solution for the most probable hyperparameters maximizing the evidence. In the case of a single weight decay regularizer it gives the following solution (MacKay 1991):

$$\gamma := \frac{\delta}{||\boldsymbol{\theta}_{\mathrm{MP}}||^2} \qquad \text{with} \qquad \delta = k - \gamma_i \mathrm{tr}(\mathbf{H}^{-1}),$$

where k is the total number of parameters in $\boldsymbol{\theta}$ and $\mathbf{H}$ is the Hessian matrix given by the second derivatives of $-\ln p(\boldsymbol{\theta}|D)$ (evaluated at $\boldsymbol{\theta}_{\mathrm{MP}}$); δ is generally referred to as the number of well-determined parameters and $0 \le \delta \le k$. In what follows, we will assume that all parameters are well-determined by the data and replace δ with the number of parameters k. This way we do not need to calculate and store the Hessian matrix which can become cumbersome when having hundreds of parameters (as is often our case). This approximation is called quick and dirty (MacKay 1991, Chapter 3) but also cheap and cheerful[10] (MacKay 1994a).

The above is readily generalized from the case of a single weight decay regularizer to the more complicated Ard regularizer (2.53). This regularizer can also be interpreted as the logarithm of a prior distribution on the parameters with each hyperparameter γ_i controlling one of the columns of $\mathbf{W}$:

$$p(\mathbf{W}|\boldsymbol{\gamma}) = \prod_{i=1}^{d-1} \left(\frac{\gamma_i}{2\pi}\right)^{d/2} \exp\left(-\frac{1}{2}\gamma_i ||\mathbf{w}_i||^2\right). \tag{2.56}$$

Each of the regularization parameters γ_i can be re-estimated separately (MacKay 1991) and using the cheap and cheerful approximation this gives:

$$\gamma_i := \frac{d}{||\mathbf{w}_i||^2}, \tag{2.57}$$

[9] A fully Bayesian treatment would use the entire distribution and not only the most probable hyperparameters; the distribution should then be integrated over in later stages.

[10] It is like with a glass of beer which is half full or half empty: the degree of fullness is in the eye of the beholder. I will refer to it as the cheap and cheerful approximation.

where we have used that each $\mathbf{w}_i$ is a vector of d parameters. The consequence of the above re-estimation formula is that columns $\mathbf{w}_i$, for which there is insufficient support from the data will be driven to zero, with the corresponding $\gamma_i \to \infty$. The un-used dimensions are switched off completely. The effective or underlying dimensionality of the model is defined as the number of columns $\mathbf{w}_i$ whose value remain non-zero.

A practical implementation of the evidence framework using an ARD regularizer (2.53), (2.56) consists of iteratively performing:

1 Minimize a regularized error function (2.54): $E = -\ln p(D|\mathbf{W}) + \frac{1}{2}\sum_{i=1}^{d-1} \gamma_i ||\mathbf{w}_i||^2$.

2 Re-estimate the regularization parameters $\boldsymbol{\gamma}$ using (2.57): $\gamma_i := d/||\mathbf{w}_i||^2$.

Actually, in practice one often does not perform step 1 until convergence to a minimum; just a few iterations of the optimization algorithm suffice.

We now apply this framework to FA and PPCA. Due to the approximation made, step 2 is straightforward. The regularization term in step 1, however, requires changing the EM algorithm for these models.

2.7.2 Bayesian Factor Analysis and PCA

We limit ourselves to the case of a single latent variable model in order to simplify the notation, but the extension to mixture models is easy. As before, we start with factor analysis and PPCA is dealt with as a special case.

Factor Analysis The log-likelihood maximized by the EM algorithm for factor analysis in the single component case was (2.21):

$$\mathcal{L}(\boldsymbol{\mu}, \mathbf{W}, \mathbf{R}) = -\frac{N}{2}\{d\ln(2\pi) + \ln|\mathbf{M}| + \mathrm{tr}(\mathbf{M}^{-1}\mathbf{S})\}.$$

The error function to be minimized including the ARD regularization term then becomes:

$$E = -\mathcal{L}(\boldsymbol{\mu}, \mathbf{W}, \mathbf{R}) + \frac{1}{2}\sum_{i=1}^{d-1} \gamma_i ||\mathbf{w}_i||^2.$$

As one can see by checking the various steps of the EM algorithm for MFAs derived in section 2.4, the only part that changes is the estimation of $\mathbf{W}$ in the second stage M-step. All we need is the expected complete error function at that stage (2.46), while at the same time restricting ourselves to just one mixture component and adding the regularization term:

$$\begin{aligned}\mathcal{E}(\hat{E}_c) \quad = \quad & -\sum_n \Big[-\frac{1}{2}\ln|\mathbf{R}| - \frac{1}{2}(\mathbf{x}^n - \boldsymbol{\mu})^T\mathbf{R}^{-1}(\mathbf{x}^n - \boldsymbol{\mu}) + (\mathbf{x}^n - \boldsymbol{\mu})^T\mathbf{R}^{-1}\mathbf{W}\langle\, \mathbf{z}^n \,\rangle \\ & -\frac{1}{2}\mathrm{tr}\left\{\mathbf{W}^T\mathbf{R}^{-1}\mathbf{W}\langle\, \mathbf{z}^n(\mathbf{z}^n)^T \,\rangle\right\}\Big] + \frac{1}{2}\sum_{i=1}^{d-1}\gamma_i||\mathbf{w}_i||^2. \end{aligned} \tag{2.58}$$

As before, we take the partial derivative with respect to $\mathbf{W}$ and put it to zero (which is just (2.47) plus the regularization term):

$$\frac{\partial\mathcal{E}(\hat{E}_c)}{\partial\mathbf{W}} = -\sum_n \left\{\mathbf{R}^{-1}(\mathbf{x}^n - \boldsymbol{\mu})\langle\, \mathbf{z}^n \,\rangle^T - \mathbf{R}^{-1}\mathbf{W}\langle\, \mathbf{z}^n(\mathbf{z}^n)^T \,\rangle\right\} + \frac{1}{2}\frac{\partial \sum_{i=1}^{d-1}\gamma_i||\mathbf{w}_i||^2}{\partial\mathbf{W}} = 0. \tag{2.59}$$

The last term can be written in matrix form with $\mathbf{\Gamma} = \text{diag}(\boldsymbol{\gamma})$; using trace rotation (A.4) and the diagonality of $\mathbf{\Gamma}$, we have:

$$\frac{\partial \sum_{i=1}^{d-1} \gamma_i ||\mathbf{w}_i||^2}{\partial \mathbf{W}} = \frac{\partial \text{tr}\left(\mathbf{\Gamma W}^T \mathbf{W}\right)}{\partial \mathbf{W}} = 2\mathbf{W\Gamma}.$$

Substituting this equation in (2.59) and introducing the shorthands for the sufficient statistics $\mathbf{X} = \sum_n (\mathbf{x}^n - \boldsymbol{\mu})\langle\, \mathbf{z}^n \,\rangle^T$ and $\mathbf{Z} = \sum_n \langle\, \mathbf{z}^n (\mathbf{z}^n)^T \,\rangle$ gives:

$$0 = -\mathbf{R}^{-1}\mathbf{X} + \mathbf{R}^{-1}\mathbf{WZ} + \mathbf{W\Gamma}.$$

Multiplying both sides by $\mathbf{R}$:

$$\mathbf{X} = \mathbf{WZ} + \mathbf{RW\Gamma}. \tag{2.60}$$

The additional term due to regularization does complicate things, since there is no straightforward way of solving this system as before (where we had (2.48) $\mathbf{W} = \mathbf{XZ}^{-1}$). Our goal is to factor out $\mathbf{W}$ and this requires some further manipulation. We consider the equivalent system using the "vec" operator (A.10) which stacks the columns of a matrix and applying it to both sides:

$$\text{vec}(\mathbf{X}) = \text{vec}(\mathbf{WZ}) + \text{vec}(\mathbf{RW\Gamma}).$$

Using (A.13) (instantiation $\mathbf{A}, \mathbf{B}, \mathbf{C} := \mathbf{I}_d, \mathbf{W}, \mathbf{Z}$) vec($\mathbf{W}$) can be factored out of the first term:

$$\text{vec}(\mathbf{X}) = (\mathbf{Z}^T \otimes \mathbf{I}_d)\text{vec}(\mathbf{W}) + \text{vec}(\mathbf{RW\Gamma}),$$

where $\otimes$ denotes the Kronecker product (A.12). Both $\mathbf{R}$ and $\mathbf{\Gamma} = \text{diag}(\boldsymbol{\gamma})$ are diagonal and defining $\mathbf{r}$ as the main diagonal of $\mathbf{R}$:

$$\text{vec}(\mathbf{X}) = (\mathbf{Z}^T \otimes \mathbf{I}_d)\text{vec}(\mathbf{W}) + \text{vec}\{(\mathbf{r}\boldsymbol{\gamma}^T) \circ \mathbf{W}\},$$

where $\circ$ denotes the element-wise matrix product (A.14). With (A.15), the element-wise product can be eliminated:

$$\text{vec}(\mathbf{X}) = [\mathbf{Z}^T \otimes \mathbf{I}_d + \text{diag}\{\text{vec}(\mathbf{r}\boldsymbol{\gamma}^T)\}]\text{vec}(\mathbf{W}). \tag{2.61}$$

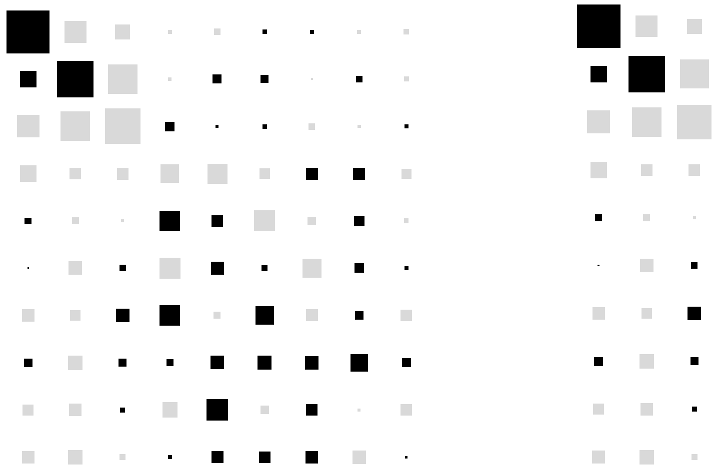
At last, we have obtained a linear system for which we can find $\mathbf{W}$ with standard numerical methods. However, the matrix between square brackets is of size $d\ell \times d\ell$ and seemingly solving the system is $O(d^3\ell^3)$. Luckily, because of the sparseness of the identity matrix $\mathbf{I}_d$, it has at most $d\ell^2$ non-zero elements: the matrix is tiled with ℓ^2 diagonal matrices of size $d \times d$. Permuting rows and columns, it can actually be transformed into a matrix with d matrices of size $\ell \times \ell$ on the main diagonal. Such *banded* systems can be solved with complexity $O(d\ell^3)$ (Golub and Van Loan 1996, page 153).

Principal Component Analysis For PPCA the inclusion of the regularization term requires only a simple modification of the standard EM algorithm. We go back to the system we started with (2.60) and substitute the spherical noise covariance matrix of PPCA $\mathbf{R} = \sigma^2\mathbf{I}_d$:

$$\mathbf{X} = \mathbf{WZ} + \sigma^2\mathbf{W\Gamma}.$$

Now it is straightforward to factor out $\mathbf{W}$:

$$\mathbf{X} = \mathbf{W}(\mathbf{Z} + \sigma^2\mathbf{\Gamma})$$



(a) ML estimation of $\mathbf{W}$ (b) Bayesian estimation of $\mathbf{W}$

Figure 2.7: Comparison of ML and Bayesian estimation for PPCA on a toy data set. The data set consists of 100 10-dimensional points where the first three coordinates are normally distributed with a variance of 1 and the other 7 normally distributed coordinates with a variance of 1/4. The Hinton diagrams of $\mathbf{W}$ show that the Bayesian model finds the appropriate dimension of latent space, while the ML model does not.

and substituting the definitions of $\mathbf{X}$ and $\mathbf{Z}$, the estimate for $\mathbf{W}$ becomes (Bishop 1999a):

$$\mathbf{W} = \left[\sum_n (\mathbf{x}^n - \boldsymbol{\mu}) \langle \mathbf{z}^n \rangle^T \right] \left[\sum_n \langle \mathbf{z}^n (\mathbf{z}^n)^T \rangle + \sigma^2 \boldsymbol{\Gamma} \right]^{-1} . \tag{2.62}$$

Comparing this with the update for $\mathbf{W}$ in Algorithm 3 for MFAs, we see that this only introduced an additional term $\sigma^2\boldsymbol{\Gamma}$. This means that for PPCA and MPCA, the regularization term does not change computational complexity and storage requirements.

2.7.3 Experiments: Toy Data

As a first test of Bayesian PCA, I took an example from the paper by Bishop (1999a) (in this way, it also offered the possibility of cross-checking my implementation). This example consists of 100 10-dimensional data points from a Gaussian distribution with a variance of 1 in the first three coordinates and a variance of a 1/4 in the remaining seven coordinates. Thus, the underlying dimensionality of the data is equal to three and the variance in the other seven directions can be modeled with the single parameter σ^2 of the spherical noise covariance matrix. However, since we only have a small sample of 100 points, we expect maximum likelihood estimation to have difficulties in separating sample and signal noise. This is confirmed by the 10×9 $\mathbf{W}$ matrix found by ML estimation of which the Hinton diagram is shown in Figure 2.7a. We see that it did discover the three principal components; however, the last six columns have non-zero elements due to the small sample size.

The 10×9 $\mathbf{W}$ matrix found by Bayesian PCA has only three non-zero columns as is shown by its Hinton diagram in Figure 2.7b. It does indeed discover the underlying dimensionality of the data and explains the remaining variance as noise by putting the other columns to zero.

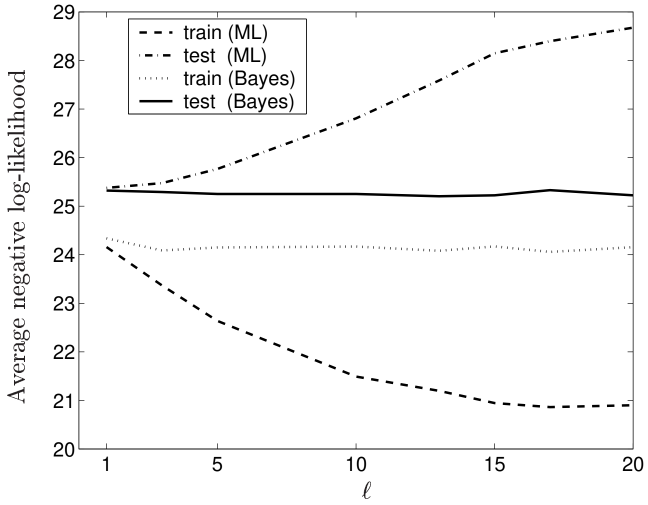


Figure 2.8: Comparison of a MPCA trained with maximum likelihood and a MPCA trained in a Bayesian framework on waveform: the average negative log-likelihood versus the dimension of latent space ℓ with a mixture of 4 PCAs. Results are the average of a 5x2cv experiment.

In a second experiment, we compare ML and Bayesian estimation for MPCAs. For this purpose, I performed density estimation on the waveform data of section 2.6.2 with a 4-component MPCA. As is to be expected a mixture of maximum likelihood PCAs starts overfitting the training data when the dimension of latent space is increased (Figure 2.8). A Bayesian mixture of PCAs resolves this problem and both training and test error stay on the same level when increasing ℓ. This can be explained by the fact that Bayesian estimation successfully determines the underlying dimensionality for each of the mixture components. In this experiment, in average 2-3 dimensions per component are retained and the other columns of the weight matrices are put to zero.

2.7.4 Experiments: Real-World Data

We now go back to the real-world data sets considered in section 2.6.3. A series of experiments has been performed in which the best MPCAs and MFAs found with maximum likelihood (Table 2.4) are compared with their Bayesian cousins. The set-up of the experiments for training the Bayesian MPCAs and MFAs is almost the same as the framework described in section 2.6.1. The difference lies in the estimation of the parameters for which 40 iterations of EM were performed with the new updates for the weight matrices: (2.62) for MPCAs and (2.61) for MFAs. After each fifth iteration of the EM algorithm, the hyperparameters γ were re-estimated (2.57).

The results for MPCAs are in Table 2.5. For convenience, the results for the best ML MPCAs found in Table 2.4 have been copied here together with the number of mixture components and the dimension of latent space. The Bayesian MPCAs had the same number of mixture components as their ML counterparts and a latent dimension of $\ell = d-1$ (except for the soybean data to save some computing time: $\ell = 50$ instead of 133). When comparing the negative log-likelihood scores of the ML and the Bayesian models, one observes that in most cases Bayesian estimation gives better results, even if the difference is often quite small and likely to be not significant. There are three data sets for which the difference is more important: optical, segmentation, and soybean. This is due to the fact that in the ML experiments, I did not include values of ℓ which are high enough. Choosing the dimension of latent space equal to the average value found by Bayesian MPCA, the scores obtained

Data	# components	ML MPCA likelihood	ML MPCA # factors	Bayesian MPCA likelihood	Bayesian MPCA # factors
banana	6	2.6(0.08)	1	2.6(0.09)	0.6/1
cancer	6	3.7(1.43)	5	3.0(1.31)	3.7/9
dermatology	6	37.9(0.66)	1	36.5(1.49)	3.9/33
glass	4	11.1(3.62)	1	10.1(2.56)	4.5/8
heart	2	18.0(0.24)	1	17.6(0.90)	3.5/12
ionosphere	8	29.4(2.20)	1	26.2(1.42)	2.2/33
iris	2	2.6(0.31)	3	2.8(0.52)	2.5/3
letter	20	12.0(0.14)	15	11.7(0.16)	12.2/15
optical	20	46.1(6.55)	15	26.8(5.72)	45.6/63
pen	20	3.9(0.35)	15	3.8(0.38)	11.4/15
satimage	10	−3.1(0.43)	15	−3.6(0.23)	14.8/35
segmentation	10	−7.6(7.69)	8	−17.9(5.33)	10.4/18
sonar	1	73.5(3.11)	5	71.2(3.38)	16.4/59
soybean	4	−13.2(9.40)	8	−116.0(17.83)	21.3/50
vowel	11	11.5(0.31)	3	11.2(0.25)	3.8/9
waveform	1	24.8(0.27)	10	24.9(0.34)	7.9/20
waveform-noise	2	53.5(0.46)	1	53.5(0.47)	3.9/39

Table 2.5: A comparison of input density modeling with MPCAs trained with maximum likelihood and in the Bayesian framework. Scores are in average negative log-likelihood on the test set and are the average over 10 experiments in a 5x2cv F test set-up. The standard deviation is given between parentheses. The ML score corresponds to the best MPCA model as evaluated on a validation set. The second column (# components) indicates the number of mixture components in the "best" model. The fourth column specifies the dimension of latent space of the best model. The Bayesian approach has been applied to a MPCA with the same number of components (as indicated by the second column) and with a number of factors $\ell = d-1$. The final column gives the average number of factors selected by the ARD prior in the Bayesian approach versus ℓ.

with ML come closer to the ones for Bayesian estimation although they are still slightly worse.

The average selected dimensionality over all mixture components with Bayesian MPCAs is given in the last column of Table 2.5. This value is often close to the value found on validation data by ML given in the fourth column. Bayesian estimation is a useful alternative to cross-validation indeed. Moreover, for MPCAs it almost comes for free when using the cheap and cheerful approximation to the evidence scheme.

The results for MFAs are in Table 2.6. The set-up was as with the Bayesian MPCAs and again the ML scores have been copied from Table 2.4. The likelihood scores are also quite comparable, although there are two exceptions for which Bayesian MFAs perform considerably worse: optical and soybean. A possible explanation might be that in this particular case, the cheap and cheerful approximation breaks down due to the rather high dimension of data space and latent space. An indication is that scores comparable to the ML score are obtained when estimating Bayesian MFAs for optical and soybean with $\ell = 20$. The average selected dimensionality over all mixture components with Bayesian MFAs in the last column of Table 2.6 is again quite close to the value selected on validation data by ML given in the fourth column. Thus, also for MFAs the Bayesian approach is a viable alternative to costly cross-validation in a ML framework.

Data	# components	ML MFA likelihood	ML MFA # factors	Bayesian MFA likelihood	Bayesian MFA # factors
banana	6	2.6(0.10)	1	2.7(0.05)	0.5/1
cancer	2	6.1(0.98)	5	4.7(1.11)	5.3/9
dermatology	4	33.4(9.42)	5	30.8(6.25)	6.4/33
glass	1	13.1(4.93)	5	11.3(4.01)	8.0/8
heart	6	14.5(1.38)	1	13.3(3.58)	2.1/12
ionosphere	1	37.5(1.72)	5	40.6(3.59)	21.0/33
iris	2	2.7(0.25)	2	2.6(0.28)	2.0/3
letter	20	11.6(0.50)	15	12.1(0.14)	11.8/15
optical	10	−12.5(1.71)	15	1.3(2.52)	52.1/63
pen	20	−4.7(0.74)	15	−4.4(0.48)	11.7/15
satimage	6	−4.4(0.36)	15	−4.2(0.24)	16.5/35
segmentation	10	−20.6(4.66)	8	−20.5(4.38)	8.5/18
sonar	1	68.9(4.54)	15	67.8(2.51)	15.0/59
soybean	6	−196.7(6.85)	8	−160.4(9.22)	5.6/50
vowel	11	11.5(0.39)	5	11.2(0.27)	3.6/9
waveform	4	24.3(0.25)	1	24.4(0.24)	1.1/20
waveform-noise	1	51.6(0.51)	3	51.6(0.53)	2.3/39

Table 2.6: A comparison of input density modeling with MFAs trained with maximum likelihood and in the Bayesian framework. See Table 2.5 for additional details.

2.7.5 Discussion

The price one has to pay for the Bayesian approach for FA is that even when using the cheap and cheerful approximation, it is computationally more expensive than the standard EM algorithm. Solving the linear system (2.61) is $O(d\ell^3)$, which is cumbersome for high-dimensional data and when choosing $\ell = d-1$. Luckily, this is in general also the case in which we do not expect a high effective dimensionality of latent space. Thus, it would not harm to follow the Bayesian approach but with a lower value of ℓ. This might also resolve some of the problems of the cheap and cheerful approximation since many parameters are likely to be not well-determined for high values of ℓ.

Recently alternative Bayesian treatments for PCA (Bishop 1999c) and MFAs (Ghahramani and Beal 2000) have been proposed. Both take a *variational* approach to Bayesian inference. This exploits the same idea that we encountered in the derivation of the generalized EM Algorithm (Algorithm 1) in order to approximate the posterior of the hidden variables. For Bayesian inference the parameters $\boldsymbol{\theta}$ are considered hidden and the posterior distribution is $p(\boldsymbol{\theta}|D)$. As we saw above, integrating over this distribution is in general not feasible and we used a Gaussian approximation around the most probable parameter values $\boldsymbol{\theta}_{\mathrm{MP}}$. The choice of a Gaussian is quite arbitrary and mainly motivated by analytical simplicity.[11] The variational approach introduces an approximating distribution $Q(\boldsymbol{\theta}|D)$ in order to make the integration tractable. Following the same reasoning as in the derivation of Algorithm 1, this leads to the idea of maximizing a lower bound of the evidence $p(D)$ (Jordan, Ghahramani, Jaakkola, and Saul 1999). It turns out that in many cases one only needs to assume that the approximating Q factorizes in a specific way, to keep the maximization tractable. The variational maximization determines the specific functional form of Q given this factorization and the priors on the parameters $\boldsymbol{\theta}$.

[11] Although in the large sample limit, the approximation becomes good.

Variational PCA (Bishop 1999c) not only includes the ARD prior on the weights $\mathbf{W}$ but also priors for the other parameters of the model. The first moments of the optimal Q distributions for W and the hyperparameters $\boldsymbol{\gamma}$ turn out to be very similar to the ones found above with both the Gaussian and the cheap and cheerful approximations. Ghahramani and Beal (2000) went one step further and proposed to use variational inference for MFAs. Their procedure cannot only automatically determine the dimensionality for each of the mixture components but also the number of components. While pruning components fits nicely in the variational framework, component birth is done in a heuristic way by splitting existing components. Performance on a number of toy examples is promising. The first moments of the optimal Q distributions for $\mathbf{W}$ and the hyperparameters $\boldsymbol{\gamma}$ are again almost identical to the ones found above.

It has also been proposed to use an annealing scheme for performing model selection with MPCAs (Meinicke and Ritter 1999). The disadvantage of this approach is that at its heart it repeatedly uses an EM algorithm on full GMMs and an eigendecomposition of weighted covariance matrices. This becomes computationally expensive for high-dimensional data and mixtures of latent variable models were introduced to avoid exactly this!

In section 4.3 it is shown that Bayesian MPCAs and MFAs also give good results when used for estimating class-conditional densities in a Bayes classifier.

2.8 Incremental and On-Line Kernel PCA

Until now we have mainly addressed the problem of density estimation. This section deals with another form of unsupervised learning, viz. feature extraction. As we saw already in section 2.3, PCA is one of the most popular techniques for feature extraction and latent variable models provide us with a probabilistic reformulation. When using PCA for feature extraction it amounts to a *linear* projection of the data onto the principal subspace. This has naturally motivated the development of *non-linear* variants of PCA. One of the motivations for the work of Tipping and Bishop (1999) was exactly this. Their MPCA provides a way of combining a number of local PCA models in a non-linear way. A related approach is to partition the data using vector quantization and then to find the principal subspace within each partition (Kambhatla and Leen 1997). This can be interpreted as a "winner-take-all" variant of the MPCA model. Various global non-linear approaches have also been developed such as auto-associative multi-layer perceptrons minimizing the reconstruction error (Kramer 1991; Diamantaras and Kung 1996) and principal curves (Hastie and Stuetzle 1989). The disadvantage of both the mixture and the global approaches is that they require non-linear optimization techniques. Moreover, their objective functions are often littered with local minima.

In this section, we pursue a different approach of making a model non-linear, viz. by *kernel* functions $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ on pairs of points in data space. If these kernel functions satisfy a certain condition (Mercer's condition), they correspond to non-linearly mapping the data in a high-dimensional *feature* space F by a map $\Phi : \mathbb{R}^d \to F$ and taking the dot product in this space (Vapnik 1995):

$$k(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}). \tag{2.63}$$

The simple polynomial kernel used on the toy data set in Chapter 1 is an example of such a kernel function.

This means that *any* linear algorithm in which the data only appears in the form of dot products $\mathbf{x}_i \cdot \mathbf{x}_j$, can be made non-linear by replacing the dot product by a kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$ and doing all the other calculations as before. The crux is that it enables us to work in feature space without having to map the data into it. The best-known example using this idea is the support vector machine (SVM) in which a linear classification method based on hyperplanes is transformed into a powerful non-linear method by kernel functions. Some examples of valid (that is, satisfying Mercer's condition) kernels are:

Polynomial kernels:	$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^p$
Radial basis function (RBF) kernels:	$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left[-\|\mathbf{x}_i - \mathbf{x}_j\|/(2\sigma^2)\right]$
Sigmoid kernels:	$k(\mathbf{x}_i, \mathbf{x}_j) = \tanh\left[a(\mathbf{x}_i \cdot \mathbf{x}_j)\right]$,

which all correspond to a dot product in a high-dimensional feature space. For a polynomial kernel of degree p, for example, the feature space consists of all products of entries up to order p: a quantity which grows like d^p. Recently, the kernel trick has also been applied to Fisher discriminant analysis (Mika, Rätsch, Weston, Schölkopf, and Müller 1999) and PCA (Schölkopf, Smola, and Müller 1998). The latter is coined kernel PCA and is based on a formulation of PCA in terms of the dot product matrix instead of the covariance matrix. This makes it possible to extract non-linear features by solving an eigenvalue problem like for PCA.

After a brief recall of the main ideas of kernel PCA, I show how we can profit from the work done in the previous sections and use the EM algorithm for PCA (Algorithm 5) to solve the eigenvalue problem and make kernel PCA more tractable on large data sets. This involves an adaptation of the formulation of kernel PCA which I developed independently of a similar approach by Mika (1998) described in section 2.8.2. The main contribution of this section consists of the application of the incremental version of Algorithm 5 in the context of kernel PCA. This alleviates the need for storing a so-called kernel matrix of size $N \times N$ and enables us to obtain the first results with kernel PCA on data sets with $N > 3,000$ in section 2.8.3. The usefulness of kernel PCA is illustrated by using the extracted non-linear features and training simple linear classifiers on them. Results on several data sets are among the best obtained with various other models described in this thesis. I conclude this section by the development of an on-line EM algorithm for PCA which takes EM-steps on part of the data only. This builds on a general approach of Neal and Hinton (1999) for on-line EM but the application to PCA is novel. Experiments in section 2.8.4 illustrate that this on-line algorithm can give further speed-ups of a factor 2 to 4.

2.8.1 Kernel PCA

The standard formulation of PCA is as the eigendecomposition of the covariance matrix of the data (section 2.3). We will see that PCA can also be carried out on the dot product matrix, a well-known fact in the literature (Kirby and Sirovich 1990; Schölkopf, Smola, and Müller 1998) but I think this proof is more elegant (and more objectively, shorter) than previous ones.

Let $\{\mathbf{x}^n\}$ be a data set with N examples of dimension d. We also suppose the data set to be centered: $\sum_n \mathbf{x}^n = \mathbf{0}_d$. The $d \times N$ matrix $\mathbf{X} = (\mathbf{x}^1\, \mathbf{x}^2 \, \ldots \, \mathbf{x}^N)$ represents the data in a compact way. Standard PCA is based on finding the eigenvalues and orthonormal eigenvectors of the (sample) covariance matrix of size $d \times d$:

$$\mathbf{C} = \frac{1}{N}\mathbf{X}\mathbf{X}^T. \tag{2.64}$$

The matrix in terms of dot products we are interested in is the dot product matrix of size $N \times N$:

$$\mathbf{K} = \frac{1}{N}\mathbf{X}^T\mathbf{X}. \tag{2.65}$$

With a slight abuse of language, I will refer to the eigenvectors with a corresponding eigenvalue different from zero, as the *non-zero eigenvectors*.

THEOREM 1

There is a one-to-one correspondence between the non-zero eigenvectors $\{\mathbf{v}^k\}$ of $\mathbf{C}$ and the non-zero eigenvectors $\{\mathbf{u}^k\}$ of $\mathbf{K}$ and they have the same eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_p$ (of course, $p \leq \min(d, N)$):

$$\mathbf{v}^k = \mathbf{X}\mathbf{u}^k / \sqrt{\lambda_k} \tag{2.66}$$

$$\mathbf{u}^k = \mathbf{X}^T\mathbf{v}^k / \sqrt{\lambda_k}, \tag{2.67}$$

where the scaling by $\sqrt{\lambda_k}$ normalizes the eigenvectors.

Proof Let $\mathbf{v}$ be an eigenvector of the covariance matrix $\mathbf{C}$ with eigenvalue λ: $\mathbf{X}\mathbf{X}^T\mathbf{v} = \lambda\mathbf{v}$. Then:

$$\mathbf{X}^T\mathbf{X}(\mathbf{X}^T\mathbf{v}) = \mathbf{X}^T(\mathbf{X}\mathbf{X}^T\mathbf{v}) = \lambda\mathbf{X}^T\mathbf{v},$$

so λ is also an eigenvalue of the dot matrix $\mathbf{X}^T\mathbf{X}$ with corresponding eigenvector $\mathbf{X}^T\mathbf{v}$ provided $\mathbf{X}^T\mathbf{v} \neq \mathbf{0}_N$ which follows from:

$$\lambda \neq 0 \quad \Rightarrow \quad \lambda\mathbf{v} \neq \mathbf{0}_d \quad \Leftrightarrow \quad \mathbf{X}\mathbf{X}^T\mathbf{v} \neq \mathbf{0}_d \quad \Rightarrow \quad \mathbf{X}^T\mathbf{v} \neq \mathbf{0}_N.$$

So we only have to take the non-zero eigenvectors into account. By symmetry (in $\mathbf{X}$ and $\mathbf{X}^T$), we can also conclude that each non-zero eigenvector of the dot product matrix $\mathbf{X}^T\mathbf{X}\mathbf{u} = \lambda\mathbf{u}$ corresponds to a non-zero eigenvector $\mathbf{X}\mathbf{u}$ of the covariance matrix with eigenvalue λ. The one-to-one correspondence as stated in theorem follows after a straightforward normalization of the eigenvectors. Given normalized eigenvectors $\mathbf{u}$ for the dot product matrix, one can normalize the eigenvectors for the covariance matrix:

$$\mathrm{norm}(\mathbf{v}) = \mathbf{v}^T\mathbf{v} = \mathbf{u}^T\mathbf{X}^T\mathbf{X}\mathbf{u} = \lambda\mathbf{u}^T\mathbf{u} = \lambda,$$

and the other way round. **End of** Proof

A direct consequence of the theorem is that one can perform PCA feature extraction entirely in terms of dot products by calculating the dot product matrix $\mathbf{K}$, determining its orthonormal eigenvectors $\mathbf{u}^k$ and its eigenvalues λ_k and projecting a point $\mathbf{x} \in \mathbb{R}^d$ onto the principal eigenvectors $\mathbf{v}^k$ in data space as defined by (2.66):

$$\mathbf{x}^T\mathbf{v}^k = \mathbf{x}^T\mathbf{X}\mathbf{u}^k / \sqrt{\lambda_k} = \left[\sum_{i=1}^{N} u_i^k (\mathbf{x} \cdot \mathbf{x}^i)\right] / \sqrt{\lambda_k}, \tag{2.68}$$

in which the data also appears only in a dot product. This means that we can map the data points into a high-dimensional feature space F by $\Phi : \mathbb{R}^d \to F$ and still perform PCA feature extraction in F without explicitly performing this map using the kernel trick.

When mapping the data into feature space, the dot product matrix becomes the so-called kernel matrix $\mathbf{K}$ (of size $N \times N$):

$$K_{ij} = \Phi(\mathbf{x}^i) \cdot \Phi(\mathbf{x}^j) = k(\mathbf{x}^i, \mathbf{x}^j), \tag{2.69}$$

where we used the kernel trick (2.63). Let the non-zero eigenvectors of $\mathbf{K}$ be $\{\mathbf{u}^k\}$, as before, with corresponding eigenvalues $\{\lambda^k\}$. The principal eigenvectors of the covariance matrix of the mapped data lie in the span of the Φ-images of the training data (similar to 2.66):

$$\mathbf{v}^k = \left[\sum_{i=1}^{N} u_i^k \Phi(\mathbf{x}^i)\right] / \sqrt{\lambda_k}. \tag{2.70}$$

Feature extraction can again be done by projecting a point $\Phi(\mathbf{x})$ onto the principal eigenvectors $\mathbf{v}^k$ in feature space (similar to (2.68)):

$$\Phi(\mathbf{x}) \cdot \mathbf{v}^k = \left[\sum_{i=1}^{N} u_i^k \{\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}^i)\}\right] / \sqrt{\lambda_k} = \left[\sum_{i=1}^{N} u_i^k k(\mathbf{x}, \mathbf{x}^i)\right] / \sqrt{\lambda_k}. \tag{2.71}$$

Algorithm 6 Kernel PCA

Require:

- A matrix of training examples: $\mathbf{X} = (\mathbf{x}^1\ \mathbf{x}^2\ \ldots\ \mathbf{x}^N)$.
- A kernel function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

for $i := 1$ to N **do**
  for $j := 1$ to N **do**
    $\mathbf{K}_{ij} := k(\mathbf{x}^i, \mathbf{x}^j)$ {kernel matrix}
  end for
end for
$\mathbf{J} := \text{ones}(N, N)$
$\mathbf{K} := (\mathbf{J} - \frac{1}{N}\mathbf{J})\mathbf{K}(\mathbf{J} - \frac{1}{N}\mathbf{J})$ {Centering the training data ($\star$)}
Determine eigenvectors $\{\mathbf{u}^k\}$ and eigenvalues $\{\lambda_k\}$ of $\mathbf{K}/N$.
for $k : \mathbf{u}^k$ is a non-zero eigenvector **do**
  $\mathbf{u}^k := \mathbf{u}^k / \sqrt{\lambda_k}$
end for
{Feature extraction of the first ℓ non-linear principal components of test data $\mathbf{t}_1, \ldots, \mathbf{t}_L$. Assume $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_N$}
for $i := 1$ to L **do**
  for $j := 1$ to N **do**
    $\mathbf{K}^{\text{test}}_{ij} := k(\mathbf{t}^i, \mathbf{x}^j)$
  end for
end for
$\mathcal{J} := \text{ones}(N, L)/N$
$\mathbf{K}^{\text{test}} := \mathbf{K}^{\text{test}} - \mathcal{J}'\mathbf{K} - \mathbf{K}^{\text{test}}\mathbf{J} + \mathcal{J}'\mathbf{K}\mathbf{J}$ {Centering the test data ($\star$)}
$\mathbf{A} := (\mathbf{u}^1\ \mathbf{u}^2\ \ldots\ \mathbf{u}^\ell)$
$\mathbf{Z} := \mathbf{K}^{\text{test}}\mathbf{A}$ { (2.71) in matrix notation}
{$\mathbf{Z}$ is the $L \times \ell$ matrix of the first ℓ non-linear principal components of $\mathbf{t}_1, \ldots, \mathbf{t}_L$}

This leads to Algorithm 6 which is formulated entirely in terms of the kernel function. The core of kernel PCA constitutes an eigenvalue problem on the kernel matrix $\mathbf{K}$. The two transformations (marked by $\star$) of the kernel matrix $\mathbf{K}$ in Algorithm 6 are necessary to have data that is centered in feature space as was assumed in theorem 1. Schölkopf et al. (1998) developed this as a way to center the data without having to calculate explicitly the mean in F (which would be impossible).

Kernel PCA corresponds exactly to linear PCA in the high-dimensional feature space F and, therefore, has all the properties of PCA in F. Because of the non-linearity of the map Φ, the features are extracted in data space in a non-linear way: the contour lines of constant projections onto a principal eigenvector are non-linear in data space. This is illustrated on a toy example which appears in the original paper of Schölkopf et al. (1998) and consists of three Gaussian clusters in $\mathbb{R}^2$, see Figure 2.9. Here I used a polynomial kernel of degree 1 to 5 for extracting the first four principal components. While for linear PCA (a polynomial kernel of degree 1) the contour lines are straight lines, they are non-linear for higher degrees. They also give a better idea of the structure of the data by focusing on regions where data points can be found. Notice also the similarity between the features extracted with kernels of the same parity (of their degree). Given the fact that the feature space of polynomial kernels corresponds to all ordered products of degree p, this does not come as a surprise. Also a RBF kernel was used for extracting principal components from the same toy data, see Figure 2.10.

Another interesting property of kernel PCA is that the number of principal components is at

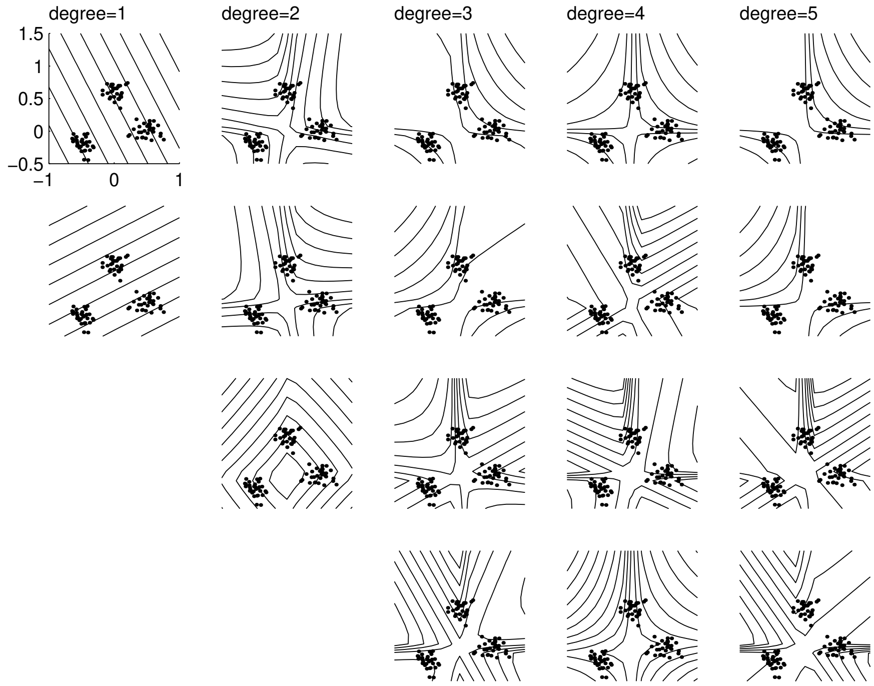


Figure 2.9: Toy example with three Gaussian clusters of 30 data points each. The figure shows the first 4 principal components (in order of decreasing eigenvalue size) extracted with the EM algorithm for kernel PCA using a polynomial kernel $(\mathbf{x}\cdot\mathbf{y})^p$ with degree $p=1\ldots5$. Contour lines indicate levels of constant principal component value. Toy data is the same as in (Schölkopf, Smola, and Müller 1998) and a script for generating it can be found on `http://svm.first.gmd.de/software/kpca-toy.m`. For linear PCA (degree 1) the contour lines are orthogonal to the eigenvectors.

most $\min(\dim(F), N)$ while for linear PCA it is at most $\min(d, N)$. This means that if the number of examples N exceeds the input dimension d, kernel PCA can, in general, extract more features. Remember from Chapter 1 that the dimension of F for a polynomial kernel of degree p is C_p^{d+p-1}. Since in this toy example $d=2$, only $p+1$ features are extracted (see Figure 2.9).

Kernel PCA has been applied by Schölkopf et al. (1998) to a character recognition problem in which non-linear features were extracted and used as input for a *linear* SVM, obtaining results which are competitive with the best results obtained with a *non-linear* SVM. This illustrates that kernel PCA is capable of extracting interesting non-linear features.

However, kernel PCA also has several disadvantages. Firstly, non-linear principal component extraction based on (2.71) involves evaluating the kernel function N times for each principal component of a new pattern while for standard PCA only the dot product of two d-dimensional vectors is needed. For large N, so-called *reduced set* methods have been proposed to approximate each eigenvector (2.70) with a smaller expansion (Schölkopf et al. 1998) $\mathbf{v}' = \sum_{i=1}^{M} \alpha_i \Phi(\mathbf{z}_i)$, where $M < N$ and $\mathbf{z}_i \in \mathbb{R}^d$. This is typically done by minimizing the squared difference between an eigenvector $\mathbf{v}$ and its approximation $\mathbf{v}'$. Gradient descent can then be used to find values for α_i and $\mathbf{z}_i$.

Secondly, a nice property of linear PCA is that one can approximately reconstruct a point in data space from its principal components. With kernel PCA, we can reconstruct a point in feature space but the problem of finding an approximate *pre-image* in data space is quite intricate (Schölkopf et al. 1999).

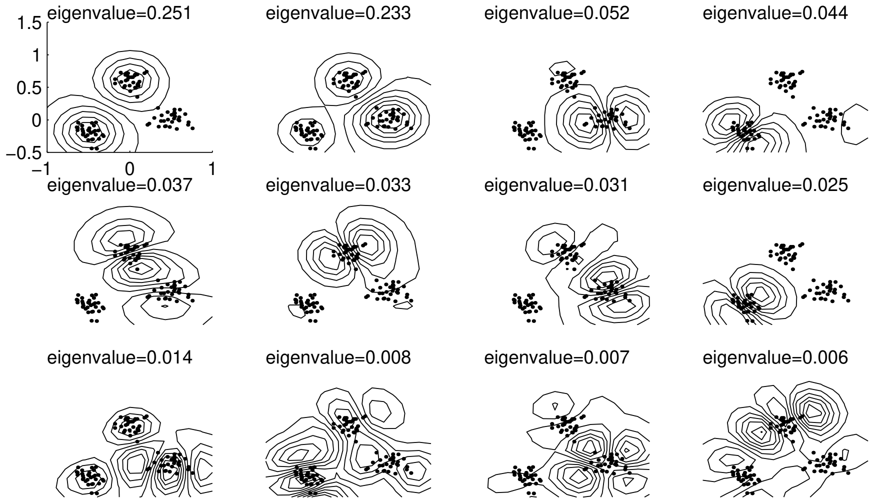


Figure 2.10: Toy example of Figure 2.9 using a RBF kernel: $\exp(-\|\mathbf{x}-\mathbf{y}\|^2/0.1)$. Contour lines indicate levels of constant principal component value. As observed by Schölkopf et al. (1998), the first two non-linear principal components (upper left) separate the three clusters. The next three split up each of the clusters. The components 6-8 also split up the clusters but are orthogonal to the previous three splits. Note that in this example, we could have extracted up to 90 (the number of data points) features from the two-dimensional data since the feature space of a RBF kernel is of infinite dimension. The eigenvalues, however, decay quite rapidly.

Thirdly, standard methods for solving eigenvalue problems need to store the entire $N \times N$ kernel matrix which can become infeasible for a large number of examples N. In the rest of this section, it is described how the incremental EM algorithm for PPCA (Algorithm 5) can be used to deal with this and make kernel PCA more tractable on large data sets.

2.8.2 Adapting Kernel PCA

A prerequisite of the EM algorithm for PCA is that is has to be applied on the data set (and not on the covariance matrix) but in the case of kernel PCA this data set $\{\Phi(\mathbf{x}^n)\}$ is high-dimensional and can most of the time not even be calculated explicitly. That is why the kernel trick was so useful in the first place! But there is a way around this problem which uses the idea of an *empirical kernel map* $\Psi_N : \mathbb{R}^d \to \mathbb{R}^N$ (Mika 1998; Schölkopf et al. 1999):

$$\begin{aligned}\Psi_N(\mathbf{x}) &= [\Phi(\mathbf{x}^1)\cdot\Phi(\mathbf{x}),\ldots,\Phi(\mathbf{x}^N)\cdot\Phi(\mathbf{x})]^T \\ &= [k(\mathbf{x}^1,\mathbf{x}),\ldots,k(\mathbf{x}^N,\mathbf{x})]^T.\end{aligned} \tag{2.72}$$

This empirical map does not map the data into feature space but into a space of size N. This is motivated by the fact that in (2.70) the eigenvectors lie in the span of the mapped training data (like the weight vector in a SVM). The empirical kernel map projects each data point onto the subspace spanned by the $\{\Phi(\mathbf{x}^n)\}$ and enables to do all calculations in the relevant subspace of F. Since the $\{\Psi_N(\mathbf{x}^n)\}$ do not form an *orthonormal* basis in $\mathbb{R}^N$, the dot product in this space is not the ordinary dot product $\langle\mathbf{x},\mathbf{y}\rangle = \sum_i \mathbf{x}_i\mathbf{y}_i$. The nice thing is that in the case of kernel PCA we can ignore this as the following argument shows.

The idea is that we have to perform linear PCA on the $\{\Psi_N(\mathbf{x}^n)\}$ from the empirical kernel map and diagonalize its covariance matrix. Let the $N \times N$ matrix $\mathbf{\Psi} = [\Psi_N(\mathbf{x}^1)\,\Psi_N(\mathbf{x}^2)\,\ldots\,\Psi_N(\mathbf{x}^N)]$, then

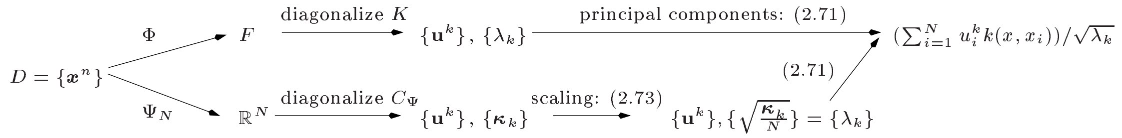


Figure 2.11: Schematic overview of kernel PCA. The upper halve describes standard kernel PCA of section 2.8.1, the lower halve describes the empirical map solution of section 2.8.2.

from (2.72) and the definition of the kernel matrix (2.69): $\mathbf{\Psi} = N\mathbf{K}$. This means that the covariance matrix of the empirically mapped data is:

$$\mathbf{C}_\Psi = \frac{1}{N}\mathbf{\Psi}\mathbf{\Psi}^T = N\mathbf{K}\mathbf{K}^T = N\mathbf{K}^2.$$

So we actually diagonalize $N\mathbf{K}^2$ instead of $\mathbf{K}$ as in kernel PCA but one can show (see (Mika 1998) for a proof) that the two matrices have the same eigenvectors $\{\mathbf{u}^k\}$. The eigenvalues $\{\lambda_k\}$ of $\mathbf{K}$ are related to the eigenvalues $\{\kappa_k\}$ of $N\mathbf{K}^2$ by:

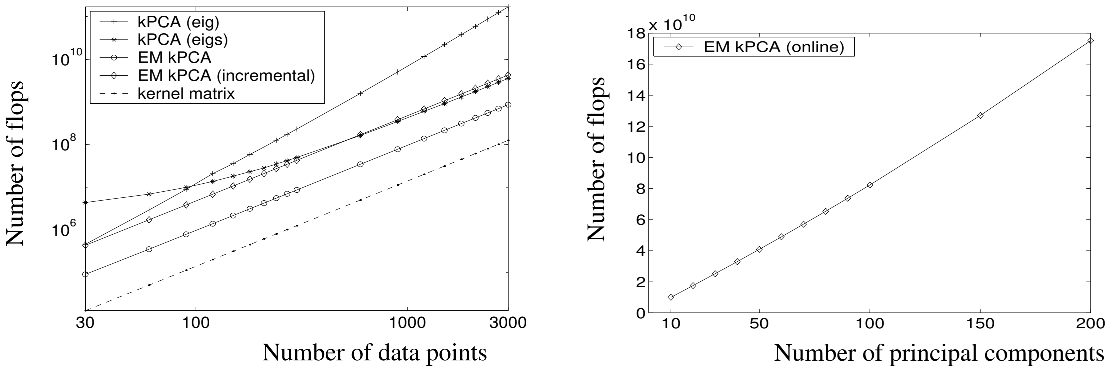
$$\lambda_k = \sqrt{\frac{\kappa_k}{N}} \tag{2.73}$$

and as before one can normalize the eigenvectors $\{\mathbf{v}^k\}$ for the covariance matrix $\mathbf{C}$ of the data by dividing each $\mathbf{u}^k$ by $\sqrt{\lambda_k}$. Figure 2.11 illustrates the two different ways of doing kernel PCA. Instead of actually diagonalizing the covariance matrix $\mathbf{C}_\Psi$, the incremental EM algorithm for PCA is applied directly on the mapped data $\mathbf{\Psi} = N\mathbf{K}$. It is now relatively easy to adapt Algorithm 6 such that it also correctly takes into account the centering of the data in an incremental way. This means that we only need to apply the empirical map to one data point at a time and do not need to store the $N \times N$ kernel matrix. The computational complexity of incremental EM for kernel PCA when extracting ℓ non-linear principal components is $O(\ell N^2)$ to which one should add $O(dN^2)$ for calculating the mapped data $\mathbf{\Psi}$ in each iteration. This should be compared to standard kernel PCA when extracting all principal components which is $O(N^3)$. If one wants to extract only a limited number of components for huge data sets, the gain in computational complexity can also be considerable.

2.8.3 Experiments

Toy Data

As a first example, we compare the various algorithms for kernel PCA on the toy data set of Figure 2.9 and 2.10. The experiment with this toy data set consisted of investigating how the complexity of the various methods for kernel PCA scales with the size of the data set (see Figure 2.12a and its caption). Complexity has been measured in floating point operations with the Matlab `flops` function. The number of data points was gradually increased from 30 to 3,000 for the four methods. One can see that standard kernel PCA scales cubicly with the number of data points due to diagonalization of an $N \times N$ matrix. This can be considered as an upper bound for all methods since if one wants to extract all N non-linear components also the other methods become cubic in N. The other methods scale quadratically with N with a similar complexity for incremental EM and implicitly restarted Arnoldi. The batch EM algorithm is computationally more efficient than its incremental variant because the kernel matrix is calculated only once, while for the incremental method each time an example is presented, its empirical kernel map has to be calculated. The big advantage of incremental EM with respect to all other methods, however, is the reduction in storage requirements. For the other



(a) # of flops vs. # of data points

(b) # of flops vs. # of principal components

Figure 2.12: Comparison of various methods for kernel PCA on the toy data of Figure 2.10 varying the number of data points and the number of extracted (non-linear) principal components. For figure (a), the y-axis indicates the number of flops for calculating the first eigenvector and eigenvalue. The methods compared are standard kernel PCA with the Matlab `eig` function (extracting all principal components), standard kernel PCA with the Matlab `eigs` function (extracting one principal component with an implicitly restarted Arnoldi method), the EM algorithm for kernel PCA and its incremental variant (both extracting one principal component and with 20 EM iterations). The calculation of the kernel matrix is also included as a lower bound for all methods. For figure (b), the on-line EM algorithm has been used to extract an increasing number of (non-linear) principal components from the toy data of Figure 2.10 with 1000 points in each of the three clusters.

methods the $N \times N$ kernel matrix has to be stored, while for incremental EM storage requirements are $O(\ell N) + O(\ell^2)$ ($\mathbf{A}, \mathbf{B}, \mathbf{W}$ in Algorithm 5).

A last concern which one might have is with respect to the convergence of the EM algorithm for kernel PCA. We know that it will converge to a global minimum but the number of iterations could increase with the number of data points. In the example of Figure 2.12a, I monitored the first principal eigenvalues and eigenvectors. Eigenvalues found by the four methods almost always agree within 99%. Also the dot product between their eigenvectors is almost always 0.99 or more, as we would expect for orthonormal vectors. This fast convergence is also its principal advantage with respect to other methods for incremental PCA based on neural network models (Oja 1982; Sanger 1989), which converge more slowly and where a learning rate has to be specified. Finally, Figure 2.12b illustrates how the complexity of incremental EM scales with the number of extracted non-linear components. Since the complexity is of $O(\ell N^2)$, it is more or less linear in $\ell \to N$.

Real-World Data

Typical applications of standard PCA include denoising, data compression and reconstruction. The application of kernel PCA to these problems is an active research topic in itself (Schölkopf et al. 1999). This is indeed a non-trivial problem as the outcomes of kernel PCA feature extraction lie in feature space and need not have a pre-image in data space: each eigenvector lies in the span of the mapped training data (2.70). Thus, techniques are required for finding approximate pre-images of those expansions in feature space.

We will, therefore, limit ourselves to a simpler (but indirect) way of evaluating the usefulness

		Linear	RBF kernel			Polynomial kernel		
Data	d	%	%	$2\sigma^2$	#	%	degree	#
banana	2	57.0	<u>92.8</u>	10	30	69.6	4	30
letter	16	76.8	<u>91.8</u>	10	512	84.5	3	256
optical	64	94.6	96.8	64	128	<u>97.4</u>	2	512
pen	16	95.5	<u>99.3</u>	10	200	98.1	2	200
sonar	60	66.9	<u>83.1</u>	30	60	75.4	4	60
vowel	10	62.4	<u>85.8</u>	5	50	69.8	2	100

Table 2.7: Results of the experiments with kernel PCA features used for classification with a perceptron. Scores are in percentage of correct classification on the test set and are the averages over 10 experiments in a 5x2cv F test framework. Each of the scores corresponds to the best model out of the particular class of models as evaluated on a validation set. The underlined scores are the ones that do not pass the 5x2cv F test test with 90% confidence when compared with the model having the best score. The column labeled "Linear" gives the percentage of correctly classified patterns when linear PCA is used. Columns labeled "#" give the number of features extracted by kernel PCA.

	Degree (polynomial kernel)			
#	1	2	3	4
16	90.3	91.0	90.4	89.9
32	93.0	94.7	94.5	94.6
64	92.9	95.7	96.1	95.7
128	92.8	97.0	96.9	96.8
256	92.6	97.4	97.5	97.1
512	-	97.8	<u>98.3</u>	97.7

Table 2.8: Percentage of correct classification on the NIST test set for a perceptron trained on non-linear features extracted with kernel PCA. A polynomial kernel $(\mathbf{x} \cdot \mathbf{y})^p$ with degree $p = 1 \cdots 4$ has been used. The underlined scores are best with 90% confidence using McNemar's test.

of the non-linear features extracted by kernel PCA, viz. as a preprocessing step for classification algorithms. For this purpose, kernel PCA features were used for training a simple linear classifier. I restricted myself to a subset of the data sets given in Table 2.1 and described in Appendix C (see Table 2.7 and 2.8 for the data sets used). Pre-processing of the attribute values was done as described in section 2.6.1. The desired outputs are based on the 1-of-c coding scheme with one output for each class. The NIST data comes with a fixed division in a training and a test set (see Table 2.1) and McNemar's test was used to determine whether the difference in performance between two methods was statistically significant (Dietterich 1998a). This test involves only a single training run and is described in detail in Appendix B. For all other data sets the 5x2cv F test was used with five replications of twofold cross-validation. The incremental EM algorithm for PCA has been used within the framework of section 2.8.2 for performing kernel PCA on the training data. While for some of the data sets standard batch methods for the eigendecomposition of the kernel matrix could have been used, this is clearly out of the question for letter (10,000 training examples) and NIST (15,025 training examples). The number of iterations of the incremental EM algorithm was chosen to be 20. For each data set both polynomial and RBF kernels were used. The degree of the polynomial kernel was varied from 1 to 4 and for the RBF kernel several values for the variance σ^2 were tried; also the number of extracted features was varied.

The linear classifier trained on the extracted features was a simple 1-layer neural network with a softmax activation function (3.14). Early stopping on a validation set, which contains one third of the training data and has balanced classes, was used to avoid overfitting. The validation data set aside for early stopping was also used to select the best model for each type of kernel function on each data set. The 1-layer neural network was trained with the scaled conjugate gradient algorithm (Møller 1993).

The results of the experiments for all data sets except the NIST data are in Table 2.7. As a basis of comparison the results of a perceptron with a polynomial kernel of degree 1 using all d features are given in the column labeled "Linear". This is just standard PCA and boils down to a rotation of the original data. What is not shown in Table 2.7 is that, in general, the performance of the linear classifier trained on non-linear principal components is better than for the same number of linear principal components. For example on the pen data, for a RBF kernel with $\sigma^2 = 5$ and 16 non-linear features already 97% of the test data is correctly classified. Moreover, kernel PCA makes it possible to extract many more components than the dimension of data space and improve results even further. This is clearly illustrated by the results listed in Table 2.7 in which the number of non-linear features used is up to 35 times as high. It is interesting to notice that a RBF kernel is often the safe choice for a kernel function and that the polynomial kernel seems mainly suitable for image data (such as optical and NIST).

The results on the NIST data are given in Table 2.8. Again the non-linear principal components systematically lead to better results than when using the same number of linear principal components. The best result obtained is with a polynomial kernel of degree 3 and 512 non-linear features (remember that $d = 256$). This is actually the best result I have obtained on the NIST data. The results on the data sets of Table 2.7 are also often as good as the best scores I obtained with mixtures of experts and Bayes classifiers using mixtures of latent variable models (see section 4.3 and 4.4.3).

A brute force approach for handling large data sets has been proposed by Schölkopf et al. (1998). Their idea is to perform kernel PCA only on $M < N$ of the training examples (2.71):

$$\Phi(\mathbf{x}) \cdot \mathbf{v}^k = \left[\sum_{i=1}^{M} u_i^k k(\mathbf{x}, \mathbf{x}^i) \right] / \sqrt{\lambda_k},$$

while extracting the principal components from all examples for subsequent training of the linear classifier. To compare this approach with full-blown kernel PCA, I performed some experiments on the NIST data with a polynomial kernel of degree 3 and 256 non-linear features. When all 15,025 training examples are used, 97.5% of the test data is classified correctly (see Table 2.8). Using only part of the data for kernel PCA this score is equaled (McNemar's test with 90% confidence) only when choosing M as high as 10,000 examples. However, already surprisingly good results are obtained with $M = 300$: the resulting perceptron correctly classifies 97.2% of the test data. It would be interesting to perform a more systematic comparison with the brute force method on various data sets. A possibility for improving this method would be not to select the subset of the data randomly (as I did here) but to start with "interesting" data points found by a clustering procedure, for example.

2.8.4 On-Line EM for PCA and Kernel PCA

Algorithm 5 for incremental PCA has the advantage of not having to store the entire data set. However, it is not really an *on-line*[12] algorithm since first all data is used in the E-step and only then parameters are updated in the M-step. One would expect the algorithm to converge faster if it was allowed to take EM-steps based on a part of the data only. This can actually be done in a sound way as was shown by Neal and Hinton (1999) for the generalized EM algorithm (Algorithm 1).

[12] My definition of the notion "on-line" is relatively weak since I still assume that we have a fixed training set which does not change over time.

Algorithm 7 On-line EM algorithm: general and for PCA.

Require:

- A training set of N examples: $\{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$.

{On-Line EM Algorithm}
loop
  E-step:
  Choose a data point $\mathbf{x}^n$
  $Q(\mathbf{z}|\mathbf{x}^n, \boldsymbol{\theta}) := p(\mathbf{z}|\mathbf{x}^n, \boldsymbol{\theta})$
  M-step:
  Update $\boldsymbol{\theta}$ to maximize the free energy $\mathcal{L}(Q, \boldsymbol{\theta})$ keeping Q fixed.
end loop

{On-Line PCA Algorithm}
$\mathbf{W} := \mathrm{rand}(d, \ell)$
$\boldsymbol{\mu} := \sum_n \mathbf{x}^n / N$
{Initialization of sufficient statistics}
$\mathbf{A} := \mathbf{0}_{(d\times\ell)}; \mathbf{B} := \mathbf{0}_{(\ell\times\ell)}$
for $n := 1$ to N **do**
  $\mathbf{s}^n := \mathbf{0}_{(d\times\ell)}; \mathbf{t}^n := \mathbf{0}_{(\ell\times\ell)}$
end for
$i := 0$
loop
  if $i = 0$ **then** $\mathbf{V} := (\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T$ **end if**
  for $n := 1$ to N **do**
    if $i > 0$ **then** $\mathbf{V} := (\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T$ **end if**
    $\langle \mathbf{z}^n \rangle := \mathbf{V}(\mathbf{x}^n - \boldsymbol{\mu})$
    $\mathbf{a} := (\mathbf{x}^n - \boldsymbol{\mu})\langle \mathbf{z}^n \rangle^T$
    $\mathbf{b} := \langle \mathbf{z}^n \rangle \langle \mathbf{z}^n \rangle^T$
    $\mathbf{A} := \mathbf{A} + \mathbf{a} - \mathbf{s}^n; \mathbf{B} := \mathbf{B} + \mathbf{b} - \mathbf{t}^n$
    if $i > 0$ **then** $\mathbf{W} := \mathbf{A}\mathbf{B}^{-1}$ **end if**
    $\mathbf{s}^n := \mathbf{a}; \mathbf{t}^n := \mathbf{b}$
  end for
  if $i = 0$ **then** $\mathbf{W} := \mathbf{A}\mathbf{B}^{-1}$ **end if**
  $i := i + 1$
end loop

This general on-line EM algorithm is given on the left-hand side in Algorithm 7. The *partial* E-step consists of choosing a pattern $\mathbf{x}^n$ and determining the posterior of the hidden variables. This on-line version is still guaranteed to converge to a local maximum of the data likelihood if each data point is regularly chosen in the E-step. A simple way to guarantee that it is indeed the case is by selecting the data point for the E-step in a cyclic way. However, the M-step still requires having the whole data set available. As Neal and Hinton (1999) pointed out, also a partial M-step suffices if the E-step can be summarized by sufficient statistics. This can be illustrated on the EM algorithm for PCA (Algorithm 5) where the M-step depends entirely on *sufficient statistics* $s = \sum_n s^n$, viz.:

$$\mathbf{A} = \sum_n (\mathbf{x}^n - \boldsymbol{\mu})\langle \mathbf{z}^n \rangle^T$$
$$\mathbf{B} = \sum_n \langle \mathbf{z}^n \rangle \langle \mathbf{z}^n \rangle^T.$$

In the incremental version of the EM algorithm for PCA, we already exploited the fact that these quantities can be calculated incrementally. We can go one step further, however, by storing the sufficient statistics s_i^m for each data point $\mathbf{x}^m$ at iteration i and using it to obtain the sufficient statistics after a partial E-step on data point $\mathbf{x}^n$ at iteration $i+1$:

$$s_{i+1} = \left(\sum_m s_i^m\right) + s_{i+1}^n - s_i^n,$$

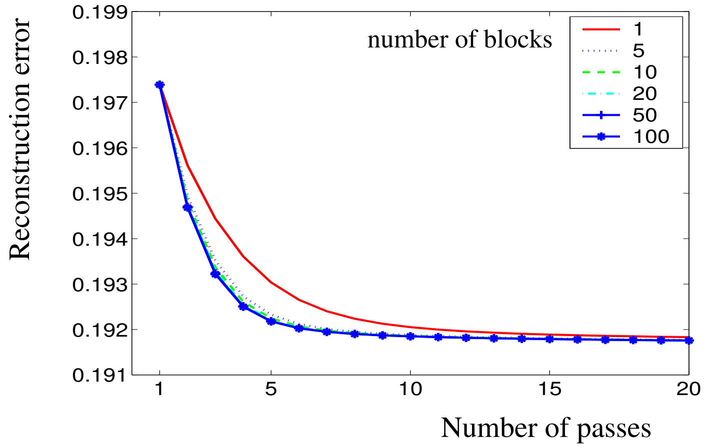


Figure 2.13: Illustration of the influence of the number of blocks when doing kernel PCA with on-line EM on the toy data of figure 2.9 (with 100 data points in each of the three clusters). Results are the average over 10 experiments.

which simply replaces the old sufficient statistics with the newly computed ones for the data point selected in the E-step. The M-step then uses the current sufficient statistics to update the parameters.

I applied this idea to the EM algorithm for PCA to transform it into the on-line algorithm on the right-hand side of Algorithm 7. The matrices $\mathbf{s}^n$ and $\mathbf{t}^n$ play the role of the sufficient statistics for each data point as calculated in the previous iteration. A first iteration of plain incremental PCA is done to obtain good initial values for the sufficient statistics. This led to more stable results; a similar problem is also pointed out by Thiesson, Meek, and Heckerman (1999) with on-line EM for GMMs. Note that, as written, the algorithm cycles through the data set in a fixed way.

The reader can immediately notice two drawbacks of the on-line algorithm. The first one is that the on-line algorithm requires storing the old sufficient statistics $\mathbf{s}^n$ and $\mathbf{t}^n$. This is not always onerous but in this case it is: it requires storing N matrices of size $d \times \ell$ and N matrices of size $\ell \times \ell$. The second one is that the assignments $\mathbf{V} := (\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T$ and $\mathbf{W} := \mathbf{A}\mathbf{B}^{-1}$ are computationally expensive if performed for each data point separately. Both drawbacks can be handled by cycling through *blocks* of data points instead of individual points.

Similar ideas have been applied to training Gaussian mixture models with an on-line EM algorithm in (Neal and Hinton 1999; Thiesson, Meek, and Heckerman 1999). They have demonstrated that it can lead to speed-ups of a factor three with respect to batch EM on some data sets. Here I describe a small experiment with the on-line EM algorithm for PCA when used to perform kernel PCA. For this purpose, I used the toy data of Figure 2.9 with 100 data points in each of the three clusters; four principal components were extracted. The on-line algorithm was used while varying the number of blocks in which the 300 data points were partitioned. The number of blocks varied from 1, which corresponds to incremental EM since all data points are used, to 100, with 3 data points per block. Results are shown in Figure 2.13 in which the reconstruction error[13] on the training set is given as a function of the number of passes through the entire data set. This illustrates that already when partitioning the data in only five blocks, a significantly lower number of passes is needed for convergence. Splitting the

[13]The reconstruction error is defined as $||\Phi(\mathbf{x}^n) - \mathrm{P}_\ell\Phi(\mathbf{x}^n)||^2$: the squared distance between the Φ-image of $\mathbf{x}^n$ and its reconstruction when projected onto the first ℓ principal components $\mathrm{P}_\ell\Phi(\mathbf{x}^n) = [\sum_{k=1}^{\ell} \Phi(\mathbf{x}^n) \cdot \mathbf{v}^k]\mathbf{v}^k$. It can be entirely expressed in terms of the kernel function and can actually be calculated in an incremental way. Thus, it is again not necessary to store the entire kernel matrix (proof omitted).

data in more blocks does not make a difference. Experiments on some other data sets led to similar results with the on-line algorithm with 5-10 blocks converging 2 to 4 times faster than incremental EM. Whether this is really worth the effort depends on the additional computation time of the on-line algorithm. In the specific context of kernel PCA where $d = N$, the cost of the incremental EM algorithm is $O(N^2\ell)$. The additional cost of the on-line algorithm is due to the repeated calculation of $\mathbf{V}$ and $\mathbf{W}$ which is $O(N\ell^2)$ for each block of data. This means that especially for $\ell \ll N$ while partitioning the data in few blocks, the computational cost of the M-step is negligible.

2.8.5 Discussion

The on-line EM algorithm for PCA described in the previous section can easily be extended to FA, MFAs, and MPCAs. Also in this case the M-step depends completely on sufficient statistics, as can be seen by inspecting Algorithm 3. It would be interesting to use these on-line variants to speed up training of mixtures of latent variable models on large data sets.

Another possibility for extending these on-line EM algorithms could be to use a stochastic variant as proposed in (Neal and Hinton 1999; Nowlan 1991). This method does not maintain strictly accurate sufficient statistics but is based on an exponentially decaying average of the statistics of recently visited data points. This can be interpreted as a way of forgetting data which was seen a long time ago and it is especially interesting for very large data sets. Although convergence to a local optimum is not guaranteed, this variant seems to work well and fast in practice for appropriate values of the decay parameter.

Real on-line algorithms for PCA and FA which can handle data that changes over time do exist. For PCA these were already mentioned in section 2.5. For FA real on-line algorithms are more difficult to come by since the E-step has to be approximated. A first approach uses a wake-sleep algorithm in which linear generative and recognition networks are jointly trained using the delta rule (Neal and Dayan 1997). The algorithm is simple and requires only local computations but it is slow since sampling is used and convergence to a local maximum of the data likelihood is not guaranteed. Another recent approach uses iterative probability propagation on the FA network of Figure 2.3 to approximate the E-step (Frey 1999). The M-step is again realized with a simple delta rule. Experimental results indicate that although iterative probability propagation is not guaranteed to converge, in practice it often does. A disadvantage of almost all these approaches using a Hebbian or delta rule is that a learning rate has to be chosen. The EM algorithm does not have such user-defined parameters and convergence to a local optimum is guaranteed; it is considerably less local though.

A disadvantage of kernel PCA already outlined in section 2.8.1 is that principal component extraction requires calculating a dense expansion in terms of kernels:

$$\Phi(\mathbf{x}) \cdot \mathbf{v} = \left[\sum_{i=1}^{N} u_i k(\mathbf{x}, \mathbf{x}^i) \right]. \tag{2.74}$$

Reduced set methods are one way to obtain smaller expansions over $M < N$ patterns but they are computationally quite expensive. Another approach would be to sparsify the expansion such that many of the u_i^k are equal to zero and the corresponding kernels can be pruned. Remember that the $\mathbf{u}^k$ are the orthonormal eigenvectors of the kernel matrix $\mathbf{K}$. Is there a way to obtain a sparser (approximate) representation of the eigenvectors? I tried an approach which was remotely inspired by Tipping's (2000) relevance vector machine and the Bayesian method for PCA of section 2.7. The idea is to use probabilistic PCA (Algorithm 4 with $m = 1$) for kernel PCA and define a suitable prior on the weight matrix $\mathbf{W}$. Instead of the ARD prior on the columns of $\mathbf{W}$ used in section 2.7, we can define an ARD prior on all the weights:

$$p(\mathbf{W}|\mathbf{\Gamma}) = \prod_{i=1}^{d}\prod_{j=1}^{\ell}\left(\frac{\gamma_{ij}}{2\pi}\right)^{d/2}\exp\left(-\frac{1}{2}\gamma_{ij}w_{ij}^2\right),$$

with $\mathbf{\Gamma}$ a $d\times\ell$ matrix of hyperparameters. The γ_{ij} control the inverse variance of w_{ij} and for large values of γ_{ij}, the corresponding w_{ij} will be driven to zero and lead to a sparser $\mathbf{W}$. Like in section 2.7, the EM algorithm for PPCA can be adapted to incorporate the ARD regularizer and to re-estimate hyperparameters γ_{ij}. The complexity of the adapted EM algorithm is similar to the one we derived for Bayesian FA and can also profit from sparse matrix computations. I did a few small experiments with this ARD regularizer and it did lead to sparser eigenvectors on the kernel matrix in most cases. However, the eigenvectors remained quite dense with at least 50% of the coordinates different from zero.

A more promising approach has been proposed recently by Smola et al. (1999). Their sparse kernel feature analysis takes a different path in that it puts an ℓ_1 penalty on the expansion coefficients in (2.74): $\sum_{i=1}^{N}|u_i| \leq 1$. This leads to very sparse expansions which require only m kernel functions to be computed for extracting the first m features. The basic algorithm still needs to store the kernel matrix, however. They also propose an approximate variant of their algorithm which is guaranteed to find feature extractors which are among the best ones. This algorithm works on random subsamples of size c of the data. This reduces memory requirements to $O(cN)$ and still extracts meaningful features on the toy data of Figure 2.9.

CHAPTER 3

Supervised Learning: Mixture Models

While the previous chapter dealt with modeling unlabeled data $\{\mathbf{x}^n\}$, we now consider the problem of *supervised* learning of labeled data given a training set:

$$D = \{(\mathbf{x}^1, \mathbf{t}^1), \ldots, (\mathbf{x}^N, \mathbf{t}^N)\},$$

with patterns $\mathbf{x}^n = (x_1^n, \ldots, x_d^n)^T \in \mathbb{R}^d$ and *target* outputs $\mathbf{t}^n$. The goal of supervised learning is to find a model which predicts as accurately as possible the label $\mathbf{t}$ of a given pattern $\mathbf{x}$. It is natural to distinguish between two main types of supervised learning. In the first category, the labels are real-valued $\mathbf{t} \in \mathbb{R}^C$ and the task is referred to as a *regression problem*. The labels $\mathbf{t}$ can also be categorical and represent a discrete class label $\mathcal{C}_k$ in a C-class *classification* problem. While I focus mainly on classification problems in the oncoming chapters, the discussion in this chapter will be kept as general as possible.

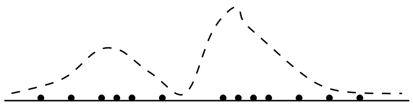
We assume that all examples are drawn independently from a joint density function $p(\mathbf{x}, \mathbf{t})$ and that the training set D is a sample from it. Adhering to the principle not to solve a more complex problem than the one we are interested in, one might say that modeling this joint density is too general a problem since it can be decomposed using the product rule (A.17):

$$p(\mathbf{x}, \mathbf{t}) = p(\mathbf{t}|\mathbf{x})p(\mathbf{x}).$$

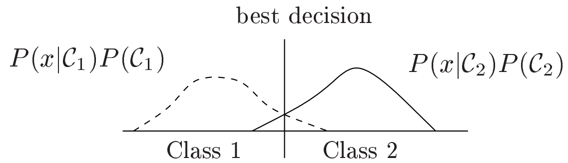
When doing supervised learning, it is the conditional density $p(\mathbf{t}|\mathbf{x})$ of $\mathbf{t}$ given $\mathbf{x}$ that we really want to model. Directly modeling this conditional density is the approach which has dominated most of neural network research.

Roadmap

This chapter describes two ways of modeling the conditional density in a maximum likelihood framework. It is mainly of tutorial nature and serves as an introduction for the extensions and experiments described in Chapter 4. We start with a brief recall of the Bayes classifiers already encountered in Chapter 1. These are of interest since the mixture models of the previous chapter can be readily plugged into Bayes classifiers as class-conditional densities. While Bayes classifiers do try to model the conditional density, this is done in an indirect way. The rest of this chapter, therefore, deals with more direct ways of modeling the conditional density. First, it is outlined how the well-known sum-of-squares error function comes about as a direct consequence of assuming a Gaussian conditional density. This leads to the fact that, in the limit of infinite data, the estimator which minimizes the sum-of-squares function is the conditional mean of the target outputs. As was the case for unsupervised learning, such a unimodal representation is often not flexible enough. This motivates the



(a) Density modeling



(b) Optimal Bayes decision

Figure 3.1: Bayes classification consists of modeling the density for each class separately (a) and classifying points according to Bayes' decision rule (b).

idea of applying mixture models also in the context of supervised learning in the form of a *conditional* mixture model. An excellent reference for this material is (Bishop 1995) and section 3.2 largely follows his presentation. The rest of the chapter studies a specific choice of a conditional mixture model, viz. mixtures of experts (MEs) (Jacobs, Jordan, Nowlan, and Hinton 1991). The modular architecture of a ME is described in detail (section 3.3) followed by a review of the literature on various ways of training MEs. The methods discussed are gradient-based optimization and, not surprising for mixture models, the EM algorithm (section 3.4). The last section contains an original contribution consisting of a generalization of the aforementioned result on the conditional mean as an optimal estimator for the sum-of-squares error. An identical result also holds for the mixture of experts error function; for a classification problem this means that MEs can in principle estimate posterior probabilities of class membership (Moerland 1997a).

3.1 Bayes Classifiers

The classic approach for a classification problem with C classes $\{\mathcal{C}_k\}$ is based on rewriting the conditional density with Bayes' rule (A.18) (extension of the two-class case of (1.3)):

$$P(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)P(\mathcal{C}_k)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|\mathcal{C}_k)P(\mathcal{C}_k)}{\sum_j p(\mathbf{x}|\mathcal{C}_j)P(\mathcal{C}_j)} = \frac{\text{class-conditional} \times \text{prior}}{\text{normalization}} \tag{3.1}$$

and modeling the *prior* $P(\mathcal{C}_k)$ and the *class-conditional* densities $p(\mathbf{x}|\mathcal{C}_k)$. The prior represents the probability that an arbitrary example out of our data belongs to class $\mathcal{C}_k$. The class-conditional distribution models the density of the data belonging to class $\mathcal{C}_k$. Before describing in some more detail how these quantities can be estimated, it is explained how (3.1) can be used to classify a pattern.

It is easy to show (see, for example (Duda and Hart 1973)) and intuitively clear that the probability of making an error when classifying an example $\mathbf{x}$ is minimized by Bayes' decision rule of assigning it to the class with the largest posterior probability (Figure 3.1):

$$\mathbf{x} \text{ is assigned to } \mathcal{C}_k \quad \Leftrightarrow \quad P(\mathcal{C}_k|\mathbf{x}) \geq P(\mathcal{C}_j|\mathbf{x}) \quad \text{for all } j \neq k.$$

This can be rewritten using the denominator of (3.1):

$$\mathbf{x} \text{ is assigned to } \mathcal{C}_k \quad \Leftrightarrow \quad p(\mathbf{x}|\mathcal{C}_k)P(\mathcal{C}_k) \geq p(\mathbf{x}|\mathcal{C}_j)P(\mathcal{C}_j) \quad \text{for all } j \neq k. \tag{3.2}$$

This means that we would have an optimal classifier if we could perfectly estimate the priors and the class-conditional densities. Of course, the most trivial problems excepted, this is not possible in practice and one needs to find approximate estimates of these quantities on a finite set of training data $\{(\mathbf{x}^n, \mathbf{t}^n)\}$. Priors $P(\mathcal{C}_k)$ are often estimated on the basis of the training set as the proportion of samples of class $\mathcal{C}_k$ or using a priori knowledge. The class-conditional densities can be modeled on the training data with non-parametric methods like histograms or nearest-neighbors or parametric methods such as the mixture models described in Chapter 2. Note that the estimation of the class-conditional densities involves K subproblems in which each of the $p(\mathbf{x}|\mathcal{C}_k)$ is estimated based on the data belonging to class $\mathcal{C}_k$ only. As a consequence it is straightforward to introduce new classes without having to reestimate the whole model: imagine that we would exchange our decimal system for the duodecimal system and would have invented new symbols representing the numbers "ten" and "eleven". A Bayes classifier for hand-written number recognition would only require estimating the class-conditional densities for the two new numbers, while a classifier which directly estimates the posterior probabilities (for example, a multi-layer perceptron) should be completely retrained.

A possible criticism of Bayes classifiers is that in a sense they are modeling too much: for each class many aspects of the data are modeled which may or may not play a role in discriminating between classes. Often Bayes classifiers also require more parameters and more computation during recall since when a new example is presented, the posterior probabilities of all classes need to be calculated. Advantages of the Bayes classifier are its ease of training: estimating the class-conditional densities is often far less computationally demanding than training the complex non-linear models described in the rest of this chapter. Moreover, as argued by Hinton et al. (1997), Bayes classifiers are more resistant to overfitting the training data, because the input data contains much more information than just the class label.

A last remark concerning the Bayes decision rule is that in practice, one often uses a numerically more stable variant of (3.2) by taking the logarithm of both sides (which is equivalent since the logarithm is monotonic):

$$\mathbf{x} \text{ is assigned to } \mathcal{C}_k \qquad \Leftrightarrow \qquad \ln p(\mathbf{x}|\mathcal{C}_k) + \ln P(\mathcal{C}_k) \geq \ln p(\mathbf{x}|\mathcal{C}_j) + \ln P(\mathcal{C}_j) \quad \text{for all } j \neq k. \tag{3.3}$$

We will come back to the Bayes classifier in section 4.3 where the GMMs, MPCAs, and MFAs of chapter 2 are used to model class-conditional densities.

3.2 Conditional Mixture Models

As said in the introduction to this chapter, a logical approach to supervised learning is to model directly the conditional density $p(\mathbf{t}|\mathbf{x})$ of target output $\mathbf{t}$ given pattern $\mathbf{x}$. Like in section 2.1 for unsupervised learning, this can be done by choosing a particular parameterization of $p(\mathbf{t}|\mathbf{x}, \boldsymbol{\theta})$ and estimating the parameters in a maximum likelihood framework for (independently distributed) training data $\{(\mathbf{x}^n, \mathbf{t}^n)\}$:

$$\mathcal{L}(\boldsymbol{\theta}) = p(\{\mathbf{t}^n\}|\{\mathbf{x}^n\}, \boldsymbol{\theta}) = \prod_n p(\mathbf{t}^n|\mathbf{x}^n, \boldsymbol{\theta}).$$

If one wants to interpret it as an error function to be minimized, we take the negative log-likelihood:

$$E(\boldsymbol{\theta}) = -\sum_n \ln p(\mathbf{t}^n|\mathbf{x}^n, \boldsymbol{\theta}), \tag{3.4}$$

where a parameterized model $\mathbf{y}(\mathbf{x}, \boldsymbol{\theta})$ is going to describe certain aspects of the conditional density. To make this a little bit more concrete, let us assume that we want to solve a regression problem with one-dimensional output $\{(\mathbf{x}^n, t^n)\}$ and that we take the conditional density to be Gaussian:

$$p(t|\mathbf{x}, \boldsymbol{\theta}) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left[-\frac{\{y(\mathbf{x}, \boldsymbol{\theta}) - t\}^2}{2\sigma^2}\right], \tag{3.5}$$

where I introduced a parameterized model $y(\mathbf{x}, \boldsymbol{\theta})$ for the mean of the Gaussian, so that $p(t|\mathbf{x}, \boldsymbol{\theta})$ can be interpreted as Gaussian noise around the mean. Substituting this in the error function (3.4) gives, leaving out the constant terms:

$$E(\boldsymbol{\theta}) = \frac{1}{2} \sum_n \{y(\mathbf{x}^n, \boldsymbol{\theta}) - t\}^2, \tag{3.6}$$

i.e. the familiar sum-of-squares error function. In the same vein, it can be shown that a Bernoulli conditional distribution gives rise to the cross-entropy error function for two-class problems and a multinomial distribution to cross-entropy for multi-class problems (see (Bishop 1995, chapter 6) for an excellent discussion).

Thus, we see that the maximum likelihood framework can offer a motivation for well-known error functions. It is important to realize that the use of a particular error function does not assume the conditional density $p(t|\mathbf{x}, \boldsymbol{\theta})$ to be really of this form. Let us have another look at the Gaussian case which motivated the sum-of-squares error function. In the limit of infinite data, the estimate $\mathbf{y}(\mathbf{x}, \boldsymbol{\theta}^{\text{best}})$ which minimizes the error function (3.6) is the conditional mean of the target outputs (Papoulis 1991; Bishop 1995):

$$y(\mathbf{x}, \boldsymbol{\theta}^{\text{best}}) = \langle t|\mathbf{x}\rangle := \int t p(t|\mathbf{x})\, dt, \tag{3.7}$$

and this also determines a global variance given by the value of the sum-of-squares error function at its minimum $y(\mathbf{x}, \boldsymbol{\theta}^{\text{best}})$. This implies that the sum-of-squares error function cannot distinguish between the true conditional distribution and a Gaussian distribution with the same conditional mean and global averaged variance. This can be particularly troublesome when the target data for the same input is multi-modal: minimizing a sum-of-squares error function can only learn the conditional mean of the data which is clearly a poor description of the data. This situation is comparable to the one in a parametric approach to unsupervised learning using simple densities and our solution throughout Chapter 2 has been to use the more flexible mixture models. Can we also do this in the context of supervised learning or, in other words, what would be the supervised counterpart of (2.9)? The obvious choice is a mixture model of m conditional probability densities $\phi_j(\mathbf{t}|\mathbf{x}, \boldsymbol{\theta})$:

$$p(\mathbf{t}|\mathbf{x}, \boldsymbol{\theta}) = \sum_{j=1}^{m} p(\mathbf{t}, z=j|\mathbf{x}, \boldsymbol{\theta}) = \sum_{j=1}^{m} p(z=j|\mathbf{x}, \boldsymbol{\theta}) p(\mathbf{t}|\mathbf{x}, z=j, \boldsymbol{\theta}) := \sum_{j=1}^{m} g_j(\mathbf{x}, \boldsymbol{\theta}) \phi_j(\mathbf{t}|\mathbf{x}, \boldsymbol{\theta}), \tag{3.8}$$

with input-dependent mixing coefficients $g_j(\mathbf{x}, \boldsymbol{\theta})$ which are non-negative and sum to one. This guarantees that the mixture distribution is normalized: $\int p(\mathbf{t}|\mathbf{x})\, d\mathbf{t} = 1$. The corresponding error function in the maximum likelihood framework is, using (3.4):

$$E(\boldsymbol{\theta}) = -\sum_n \ln p(\mathbf{t}^n|\mathbf{x}^n, \boldsymbol{\theta}) = -\sum_n \ln \sum_{j=1}^{m} g_j(\mathbf{x}^n, \boldsymbol{\theta}) \phi_j(\mathbf{t}^n|\mathbf{x}^n, \boldsymbol{\theta}) \tag{3.9}$$

and it is this mixture error function which forms the basis of the rest of this chapter. It is also the error function which underpins Bishop's (1995, section 6.4) mixture density network which he applied to learning the conditional density of multi-valued problems I discussed earlier.

Until now, I have motivated the mixture error function (3.9) by its capacity of dealing with multi-modality in regression problems. Is there a similar motivation for classification problems? Well, formally there is not: it is well-known (Duda and Hart 1973, section 5.8.3; Bishop 1995, section 6.6) that for a multi-class problem with 1-of-C coding[1] of the targets, the conditional mean (3.7) is equal

[1] For a classification problem with C classes, the target outputs are coded as $\mathbf{t} \in \{0,1\}^C$ such that for a pattern from class $\mathcal{C}_c$, output $\mathbf{t}$ has a one on its c-th coordinate and zeros everywhere else: $t_k^n = \delta_{kc}$.

to the posterior probability:

$$y_k(\mathbf{x}, \boldsymbol{\theta}^{\text{best}}) = P(\mathcal{C}_k|\mathbf{x}).$$

We know from the discussion on Bayes classifiers in the previous section that this is optimal and thus, there seems to be no reason to introduce the mixture error function. However, the above discussion on optimal estimators is biased for several reasons (Bishop 1995):

- It only holds in the limit of an infinite data set as is evidenced by the integral in (3.7). On finite data, issues such as the bias-variance dilemma and model complexity play an important role.
- At the same time, it supposes that the model $\mathbf{y}(\mathbf{x}, \boldsymbol{\theta})$ is sufficiently general to approximate the conditional mean $\langle t|\mathbf{x}\rangle$ accurately.
- The conditional mean corresponds to a *global* minimum of the error function but the error surface of a sufficiently general model $\mathbf{y}(\mathbf{x}, \boldsymbol{\theta})$ is likely to have many *local* minima in a complex non-linear optimization problem.

It is in this context that the mixture error function (3.9) can also be motivated less formally, as a way of replacing a complex global model $\mathbf{y}(\mathbf{x}, \boldsymbol{\theta})$ by a mixture of less complex local models. This is particularly relevant if the problem can be decomposed into simpler subproblems in different regions of data space. The conditional mixture model (3.8) can then be interpreted as a probabilistic subdivision of the input region by the mixing coefficients $g_j(\mathbf{x}, \boldsymbol{\theta})$ which weight the component means $\mathbf{y}_j(\mathbf{x}, \boldsymbol{\theta})$ of conditional densities $\phi_j(\mathbf{t}|\mathbf{x}, \boldsymbol{\theta})$.

Note that until now we have assumed that the mixing coefficients and the mixture components of (3.8) shared their parameters $\boldsymbol{\theta}$. To take the divide-and-conquer principle of handling a mixture of simple local models even further, it seems logical to split up the parameter vector between the different constituent components of the mixture model $\boldsymbol{\theta} = (\boldsymbol{\theta}_g, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_m)$:

$$p(\mathbf{t}|\mathbf{x}, \boldsymbol{\theta}) = \sum_{j=1}^{m} p(\mathbf{t}, z = j|\mathbf{x}, \boldsymbol{\theta}) = \sum_{j=1}^{m} p(z = j|\mathbf{x}, \boldsymbol{\theta}_g) p(\mathbf{t}|\mathbf{x}, z = j, \boldsymbol{\theta}_j) = \sum_{j=1}^{m} g_j(\mathbf{x}, \boldsymbol{\theta}_g) \phi_j(\mathbf{t}|\mathbf{x}, \boldsymbol{\theta}_j). \tag{3.10}$$

The best-known example of such a supervised modular mixture model is the *mixture of experts* of Jacobs et al. (1991) which I will discuss in more detail in the next section. The divide-and-conquer approach of mixtures of experts has shown particularly useful in discovering different regimes in piece-wise stationary time series (Weigend et al. 1995), modeling discontinuities in the input-output mapping (Nowlan 1991), and regression and classification problems in general (Waterhouse 1997).

3.3 Mixtures of Experts

A mixture of experts embodies the idea of a modular supervised mixture model described in the previous section. It does this by associating specific models with the constituent components of (3.10) in a way described in Figure 3.2. The *gating network* (or *gate*) is a neural network with m outputs and corresponds to the input-dependent mixture coefficients $g_j(\mathbf{x}, \boldsymbol{\theta}_g)$. In order to ensure a probabilistic interpretation, the $g_j(\mathbf{x})$ are related via the *softmax* function (Bridle 1990). This gives for the j-th output of the gating network:

$$g_j(\mathbf{x}) = \frac{\exp[a_j(\mathbf{x})]}{\sum_{i=1}^{m} \exp[a_i(\mathbf{x})]}, \tag{3.11}$$

where the a_i are the gating network outputs before the softmax thresholding.[2] The softmax function makes the gating network outputs sum to one and non-negative.

[2] In neural network lingo this is called an *activation function*.

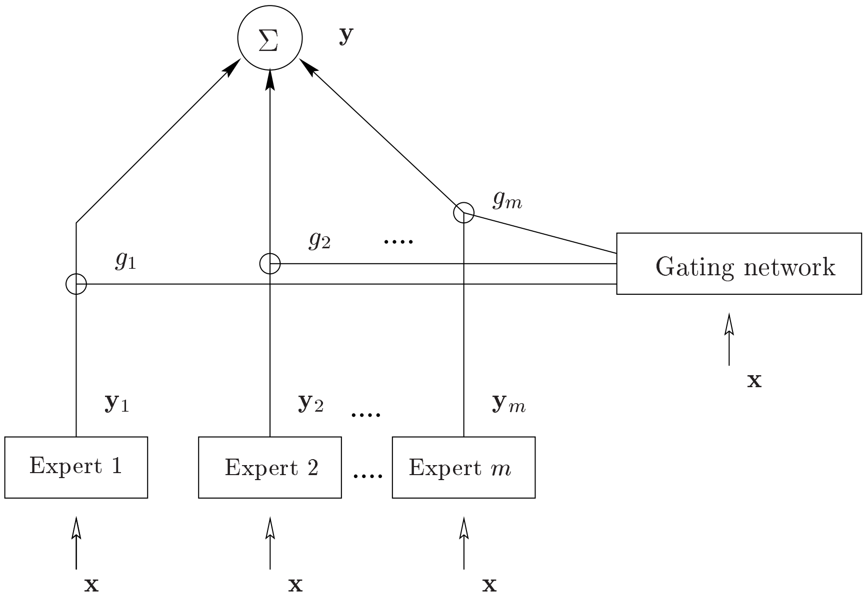


Figure 3.2: Architecture of a mixture of experts network.

The means of component densities $\phi_j(\mathbf{t}|\mathbf{x}, \boldsymbol{\theta}_j)$ are modeled by m *expert networks* (or *experts*) $\mathbf{y}_j(\mathbf{x}, \boldsymbol{\theta}_j)$ and the output of the entire ME is the linear combination of the expert outputs weighted by the gating network outputs:

$$\mathbf{y}(\mathbf{x}, \boldsymbol{\theta}) = \sum_{j=1}^{m} g_j(\mathbf{x}, \boldsymbol{\theta}_g)\mathbf{y}_j(\mathbf{x}, \boldsymbol{\theta}_j). \tag{3.12}$$

The gating network can be interpreted as a classifier that attributes patterns to the expert networks in a probabilistic way. This also implies that the gating network splits the data space in a "soft" way: data can lie in several regions simultaneously (remember Figure 1.7).

The specific choice of the expert networks depends on the problem at hand, but a standard choice are single-layer neural networks known as generalized linear models (GLMs) in the statistics literature (Jordan and Jacobs 1994; McCullagh and Nelder 1989). Denoting the parameters of expert j with a weight matrix $\mathbf{W}_j$ and the activation function with f, a GLM expert is defined as:[3]

$$\mathbf{y}_j(\mathbf{x}, \mathbf{W}_j) = f(\mathbf{W}_j\mathbf{x}). \tag{3.13}$$

GLMs motivate a direct coupling between the activation function f, the expert's component density $\phi_j(\mathbf{t}|\mathbf{x}, \boldsymbol{\theta}_j)$ and the type of problem we are dealing with. These couplings are (Jordan and Jacobs 1994):

- the identity function $f(\mathbf{x}) = \mathbf{x}$ with a Gaussian noise model (3.5) for regression problems with $\mathbf{t} \in \mathbb{R}^C$;

- the sigmoid or logistic function $f(\mathbf{x}) = 1/\{1 + \exp(-\mathbf{x})\}$ with a Bernoulli distribution (3.22) for two-class classification problems with a one-dimensional output to code the classes $t \in \{0, 1\}$;

[3] This weight matrix can include an extra column for the biases and the pattern $\mathbf{x}$ an additional coordinate equal to one for this purpose.

- the softmax function (3.11) with a multinomial distribution (3.24) for multi-class classification problems with 1-of-C coding $\mathbf{t}\in\{0,1\}^C$.

Of course, one could also use more complex experts such as multi-layer perceptrons (Weigend, Mangeas, and Srivastava 1995) but this discards the interpretation of a mixture of experts as a mixture of simple models.

A standard choice for the gating network is also a GLM with a softmax non-linearity (3.11), weight matrix $\mathbf{V}$, and weight vector $\mathbf{v}_j$ for output j $(1\leq j\leq m)$:

$$g_j(\mathbf{x},\mathbf{V}) = \frac{\exp[a_j(\mathbf{x},\mathbf{v}_j)]}{\sum_{i=1}^m \exp[a_i(\mathbf{x},\mathbf{v}_i)]} \qquad \text{with} \qquad a_j(\mathbf{x},\mathbf{v}_j) = \mathbf{v}_j^T\mathbf{x}. \tag{3.14}$$

This corresponds to the intuitively appealing interpretation of the gating network as solving a multi-class classification problem based on a multinomial distribution. A mixture of experts thus tries to learn simultaneously a "soft" partitioning of data space by the gating network and the predictions for the experts on their attributed regions. The crux of this blend of unsupervised and supervised learning is the mixture of experts error function (the negative logarithm of (3.10)):

$$E(\boldsymbol{\theta}) = -\sum_n \ln \sum_{j=1}^m g_j(\mathbf{x}^n,\boldsymbol{\theta}_g)\phi_j(\mathbf{t}^n|\mathbf{x}^n,\boldsymbol{\theta}_j). \tag{3.15}$$

In the next section, various existing methods for minimizing this ME error function are described. Learning algorithms treated are gradient-based algorithms and, not surprising given that we are dealing with a mixture model, an EM algorithm.

3.4 Training Mixtures of Experts

In this section, an extensive review is given of different learning algorithms for minimizing the ME error function (3.15). The first approach consists of standard gradient-based learning and has been applied with some success in the training of MEs (Jacobs et al. 1991; Jordan and Jacobs 1994). The second approach is an instantiation of the EM algorithm, as it has been formulated for and applied to mixtures of experts (Jordan and Jacobs 1994). Like with the unsupervised mixture models of Chapter 2, the advantage of the EM approach lies in the fact that it nicely decouples the parameter estimation for the different components of a ME model.[4]

3.4.1 Gradient-Based Optimization

Many standard iterative optimization methods (gradient descent, conjugate gradients, quasi-Newton) are based on the calculation of gradients. This has already been noted at the beginning of section 2.1 where we derived a coupled system of equations (2.5) for minimizing the negative log-likelihood with respect to its parameters θ_i:

$$\frac{\partial E(\boldsymbol{\theta})}{\partial \theta_i} = -\sum_{n,\mathbf{z}} p(\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta})\frac{\partial}{\partial\theta_i}\ln p(\mathbf{x}^n,\mathbf{z}|\boldsymbol{\theta}) = 0.$$

Transferred to the context of supervised learning, this gives:

$$\frac{\partial E(\boldsymbol{\theta})}{\partial \theta_i} = -\sum_{n,\mathbf{z}} p(\mathbf{z}|\mathbf{x}^n,\mathbf{t}^n,\boldsymbol{\theta})\frac{\partial}{\partial\theta_i}\ln p(\mathbf{t}^n,\mathbf{z}|\mathbf{x}^n,\boldsymbol{\theta}) = 0.$$

[4]This section is based on an earlier technical report (Moerland 1997c).

Although these equations are coupled, we will see that a gradient-based approach for mixtures of experts is still possible. Using the definition of a conditional mixture model (3.10) we have:

$$\begin{aligned}\frac{\partial E(\boldsymbol{\theta})}{\partial \theta_i} &= -\sum_n \sum_{j=1}^m P(z=j|\mathbf{x}^n,\mathbf{t}^n,\boldsymbol{\theta}) \frac{\partial}{\partial \theta_i} \ln\{g_j(\mathbf{x}^n,\boldsymbol{\theta}_g)\phi_j(\mathbf{t}^n|\mathbf{x}^n,\boldsymbol{\theta}_j)\} \\ &= -\sum_n \sum_{j=1}^m P(z=j|\mathbf{x}^n,\mathbf{t}^n,\boldsymbol{\theta}) \Big[\frac{\partial}{\partial \theta_i} \ln g_j(\mathbf{x}^n,\boldsymbol{\theta}_g) + \frac{\partial}{\partial \theta_i} \ln \phi_j(\mathbf{t}^n|\mathbf{x}^n,\boldsymbol{\theta}_j)\Big]. \end{aligned} \tag{3.16}$$

Like for unsupervised mixture models, the first term is the posterior of the hidden variables and can be rewritten with Bayes' rule (A.18):

$$\pi_j(\mathbf{x}^n,\mathbf{t}^n) := P(z=j|\mathbf{x}^n,\mathbf{t}^n,\boldsymbol{\theta}) = \frac{P(z=j|\mathbf{x}^n,\boldsymbol{\theta})p(\mathbf{t}^n|z=j,\mathbf{x}^n,\boldsymbol{\theta})}{p(\mathbf{t}^n|\mathbf{x}^n,\boldsymbol{\theta})} = \frac{g_j(\mathbf{x}^n,\boldsymbol{\theta}_g)\phi_j(\mathbf{t}^n|\mathbf{x}^n,\boldsymbol{\theta}_j)}{\sum_i g_i(\mathbf{x}^n,\boldsymbol{\theta}_g)\phi_i(\mathbf{t}^n|\mathbf{x}^n,\boldsymbol{\theta}_i)}. \tag{3.17}$$

These can be interpreted as indicators for the responsibility which expert j takes for a prediction on pattern $\mathbf{x}^n$ given the target output $\mathbf{t}^n$.

I suppose that the experts and the gating network are feed-forward neural networks (GLMs or MLPs). The gradients with respect to the weights θ_i in (3.16) are completely determined by the partial derivatives of the error function with respect to the network outputs a_k^n for pattern $\mathbf{x}^n$ (before thresholding $f[a_k(\mathbf{x}^n)]$). These derivatives form the basis of the back-propagation algorithm (Rumelhart, Hinton, and Williams 1986).

Gating Network

A closer look at (3.16) shows that the partial derivative with respect to the k-th output of the gating network can be written as, using (3.17):

$$\frac{\partial E(\boldsymbol{\theta})}{\partial a_k^n} = -\sum_{j=1}^m \pi_j(\mathbf{x}^n,\mathbf{t}^n) \frac{\partial \ln g_j(\mathbf{x}^n,\boldsymbol{\theta}_g)}{\partial a_k^n}.$$

This might look familiar to some readers and with reason, this is exactly the partial derivative one obtains when minimizing the cross-entropy error function for a m-class problem (Bishop 1995, section 6.9) with a softmax activation function. It reduces to:

$$\frac{\partial E(\boldsymbol{\theta})}{\partial a_k^n} = g_k(\mathbf{x}^n,\boldsymbol{\theta}_g) - \pi_k(\mathbf{x}^n,\mathbf{t}^n), \tag{3.18}$$

which has the natural interpretation of adjusting the gating network such that the outputs g_k are drawn toward the posteriors π_k.

Expert Networks

The partial derivative with respect to the k-th output of expert j can be written as (using (3.16) and (3.17)):

$$\frac{\partial E(\boldsymbol{\theta})}{\partial a_{jk}^n} = -\pi_j(\mathbf{x}^n,\mathbf{t}^n) \frac{\partial \ln \phi_j(\mathbf{t}^n|\mathbf{x}^n,\boldsymbol{\theta}_j)}{\partial a_{jk}^n}. \tag{3.19}$$

The second factor is exactly the partial derivative one obtains when minimizing a maximum likelihood-based error function (compare with (3.4)). For $\phi_j(\mathbf{t}^n|\mathbf{x}^n)$ chosen from the exponential family with its corresponding activation function, it has a particularly simple form (McCullagh and Nelder 1989; Bishop 1995, chapter 6):

$$\frac{\partial E(\boldsymbol{\theta})}{\partial a_{jk}^n} = \pi_j(\mathbf{x}^n, \mathbf{t}^n)\{y_{jk}(\mathbf{x}^n, \boldsymbol{\theta}_j) - t_k^n\}. \tag{3.20}$$

It has the natural interpretation of adapting the expert parameters such that expert output y_{jk} gets pulled toward the target output t_k^n. The posterior π_j weights the change in parameter values proportional to the responsibility of expert j for example $(\mathbf{x}^n, \mathbf{t}^n)$.

The exponential family includes the following standard distributions:

$$\text{Gaussian:} \qquad \phi_j(\mathbf{t}|\mathbf{x}) = \frac{1}{(2\pi)^{C/2}} \exp\left[-\frac{||\mathbf{t} - \mathbf{y}_j(\mathbf{x}, \boldsymbol{\theta}_j)||^2}{2}\right], \tag{3.21}$$

where C is the dimensionality of the target $\mathbf{t}$. The corresponding activation function is the identity function and the related error function is sum-of squares.

$$\text{Bernoulli:} \qquad \phi_j(\mathbf{t}|\mathbf{x}) = \prod_{k=1}^{C} \{y_{jk}(\mathbf{x}^n, \boldsymbol{\theta}_j)\}^{t_k}\{1 - y_{jk}(\mathbf{x}, \boldsymbol{\theta}_j)\}^{1-t_k}, \tag{3.22}$$

$$\tag{3.23}$$

with a sigmoid (or logistic) activation function. For two-class problems, the corresponding error function is the cross-entropy error with 0/1-coding.

$$\text{Multinomial:} \qquad \phi_j(\mathbf{t}|\mathbf{x}) = \prod_{k=1}^{C} \{y_{jk}(\mathbf{x}, \boldsymbol{\theta}_j)\}^{t_k}, \tag{3.24}$$

for multi-class classification problems with 1-of-C coding. The corresponding activation function is the softmax (3.11) and the error function is the multi-class cross-entropy.

A simple example are Glm expert networks with a single output, Gaussian conditional density, and linear activation function and a Glm gating network. The gradient descent weight updates for weights $\mathbf{w}_j$ of expert j are, using (3.20):

$$\Delta \mathbf{w}_j = -\eta \sum_n \pi_j(\mathbf{x}^n, t^n)(\mathbf{w}_j^T \mathbf{x}^n - t^n)(\mathbf{x}^n)^T,$$

and for the gating network weights $\mathbf{v}_j$ incoming to the j-th output of the gate, using (3.18):

$$\Delta \mathbf{v}_j = -\eta \sum_n [g_j(\mathbf{x}^n, \mathbf{V}) - \pi_j(\mathbf{x}^n, t^n)](\mathbf{x}^n)^T,$$

where η denotes a learning rate parameter. Equation (3.20) is similar to what one would find with ordinary sum-of-squares and cross-entropy error functions but with the posterior probabilities π_j as an extra weighting factor. Of course, the gradients obtained could also be used in more powerful non-linear optimization techniques such as conjugate gradient algorithms and quasi-Newton methods. A global least-squares approach could also be used instead of the Me error function (3.15). This might be more appropriate when we have no a priori belief that the problem can decomposed into simpler subproblems (Bradshaw, Duchâteau, and Bersini 1997).

Adaptive Variances in Mixtures of Experts

For regression problems, it is useful to introduce a local variance σ_j for each expert (Weigend et al. 1995) in the Gaussian conditional density:

$$\phi_j(\mathbf{t}|\mathbf{x}) = \frac{1}{(2\pi\sigma_j^2)^{C/2}} \exp\left[-\frac{||\mathbf{t} - \mathbf{y}_j(\mathbf{x}, \boldsymbol{\theta}_j)||^2}{2\sigma_j^2}\right]. \tag{3.25}$$

These expert variances make that the model can handle different noise levels, which is useful when dealing with piece-wise stationary time series switching between different regimes. It has been noted that this may reduce overfitting and ease the subdivision of the problem among the experts (Weigend et al. 1995). The introduction of the expert variances necessitates some small changes in the gradients derived earlier. It is easy to see that we have to add an additional factor to (3.20):

$$\frac{\partial E(\boldsymbol{\theta})}{\partial a_{jk}^n} = \pi_j(\mathbf{x}^n, \mathbf{t}^n)\frac{1}{\sigma_j^2}[y_{jk}(\mathbf{x}^n, \boldsymbol{\theta}_j) - t_k^n].$$

The factor $1/\sigma_j^2$ can be seen as a form of weighted regression that focuses on low-noise regions and that discounts high noise regions (outliers, for example).

The updates for the variance parameters are obtained using (3.16) and the definition of the spherical Gaussian (3.25):

$$\frac{\partial E(\boldsymbol{\theta})}{\partial \sigma_j} = \sum_n \pi_j(\mathbf{x}^n, \mathbf{t}^n)\frac{\partial \phi_j^n(\mathbf{t}^n|\mathbf{x}^n)}{\partial \sigma_j} = \sum_n \pi_j(\mathbf{x}^n, \mathbf{t}^n)\Big[\phi_j(\mathbf{t}^n|\mathbf{x}^n)\frac{||\mathbf{t}^n - \mathbf{y}_j(\mathbf{x}^n, \boldsymbol{\theta}_j)||^2}{\sigma_j^3} - \frac{C\phi_j(\mathbf{t}^n|\mathbf{x}^n)}{\sigma_j}\Big].$$

Setting the partial derivatives to zero, a direct solution (for the batch update) is:

$$\sigma_j^2 = \frac{1}{C}\frac{\sum_n \pi_j(\mathbf{x}^n, \mathbf{t}^n)||\mathbf{t}^n - \mathbf{y}_j(\mathbf{x}^n, \boldsymbol{\theta}_j)||^2}{\sum_n \pi_j(\mathbf{x}^n, \mathbf{t}^n)},$$

which again has a simple interpretation as the squared errors of expert j weighted by the posteriors π_j. Weigend et al. (1995) also describe the incorporation of prior belief about the expert variances in a maximum likelihood framework to avoid singularities and overfitting in regions of low noise. A Bayesian approach has also been applied to MEs (Waterhouse, MacKay, and Robinson 1996) using variational methods in this context.

We can estimate local error bars given the expert variances (see (Bishop 1995, section 6.4) for a proof):

$$\sigma(\mathbf{x}) = \sum_j g_j(\mathbf{x}, \boldsymbol{\theta}_j)\left[\sigma_j^2 + ||\mathbf{y}_j(\mathbf{x}, \boldsymbol{\theta}_j) - \mathbf{y}(\mathbf{x}, \boldsymbol{\theta})||^2\right].$$

In fact, Bishop follows a more general approach where the expert variances are also input-dependent.

3.4.2 Expectation Maximization Algorithm

The application of the EM algorithm to a mixture of experts is relatively straightforward and closely related to EM for unsupervised mixture models in section 2.2.

The E-step requires the estimation of the posteriors of the hidden variables which we already derived along the way in the previous section (see (3.17)):

$$\text{E-step:} \qquad \pi_j(\mathbf{x}^n, \mathbf{t}^n) := P(z = j|\mathbf{x}^n, \mathbf{t}^n, \boldsymbol{\theta}) = \frac{g_j(\mathbf{x}^n, \boldsymbol{\theta}_g)\phi_j(\mathbf{t}^n|\mathbf{x}^n, \boldsymbol{\theta}_j)}{\sum_i g_i(\mathbf{x}^n, \boldsymbol{\theta}_g)\phi_i(\mathbf{t}^n|\mathbf{x}^n, \boldsymbol{\theta}_i)}. \tag{3.26}$$

The M-step consists of the minimization of the expected complete error function with respect to the parameters of the mixtures of experts and gives (the supervised counterpart of (2.12)):

$$\begin{aligned}\mathcal{E}(E_c) &= -\sum_{n,\mathbf{z}} p(\mathbf{z}|\mathbf{x}^n, \mathbf{t}^n, \boldsymbol{\theta}) \ln p(\mathbf{t}^n, \mathbf{z}|\mathbf{x}^n, \boldsymbol{\theta}^{\text{new}}) \\ &= -\sum_n \sum_{j=1}^m \pi_j(\mathbf{x}^n, \mathbf{t}^n) \ln p(\mathbf{t}^n, z=j|\mathbf{x}^n, \boldsymbol{\theta}^{\text{new}}),\end{aligned}$$

which gives using (3.10):

$$\begin{aligned} &= -\sum_n \sum_{j=1}^m \pi_j(\mathbf{x}^n, \mathbf{t}^n) \ln g_j(\mathbf{x}^n, \boldsymbol{\theta}_g) \;-\; \sum_n \sum_{j=1}^m \pi_j(\mathbf{x}^n, \mathbf{t}^n) \ln \phi_j(\mathbf{t}^n|\mathbf{x}^n, \boldsymbol{\theta}_j) \qquad (3.27) \\ &= \quad E_{\text{gate}}(\boldsymbol{\theta}_g) \quad + \quad \sum_{j=1}^m E_j(\boldsymbol{\theta}_j).\end{aligned}$$

Comparing the complete error function with the original ME error function (3.15) shows that the EM algorithm has moved the logarithm inside the sum, resulting in a separate error minimization problem for each of the mixture components.

An interpretation of the cross-entropy term for the gating network $E_{\text{gate}}(\boldsymbol{\theta}_g)$, is as the entropy of distributing a pattern $\mathbf{x}$ amongst the experts. This cost is minimal if experts are mutually exclusive and increases when experts share a pattern. The second term $E_j(\boldsymbol{\theta}_j)$ for expert j has the general form of a weighted maximum likelihood problem (see also (3.19)). The weighting with π_j implies that the "influential" experts are the ones with a large value for π_j, that is by (3.26), the ones with a low error. Thus, the complete error function nicely incorporates the soft splitting of the data space which is an essential characteristic of the ME model.

The exact form of the M-step of the EM algorithm depends on the choice of the model for the gate and experts. When we choose feed-forward neural networks and $\phi_j(\mathbf{t}^n|\mathbf{x}^n)$ from the exponential family as in the previous section, gradient-based optimization leads to the same gradients as before: (3.18) for the gating network and (3.20) for the experts. When the expert and gating networks are chosen to be MLPs the M-step cannot minimize the complete error function but can at best decrease it using iterative non-linear optimization methods: the algorithm becomes generalized EM. This also implies that if one uses only one iteration of gradient descent for the experts and the gate, EM is equivalent to the gradient methods of the previous section.

When experts and gate are GLMs, the separate optimization problems reduce to maximum likelihood problems for GLMs (Jordan and Jacobs 1994). If $\phi_j(\mathbf{t}^n|\mathbf{x}^n)$ is a Gaussian conditional density, the optimization of the parameters of the experts reduces to solving a weighted least-squares problem which can be solved in one pass using pseudo-inverses. This is discussed in more detail below. For logistic or softmax GLM experts and for the softmax GLM gating network, no one-pass solution exists and iterative methods are needed. These non-linear GLMs have the nice property that their error functions have a unique minimum (Auer, Hebster, and Warmuth 1996; Nabney 1999). Jordan and Jacobs (1994) propose to use the iteratively weighted least-squares (IRLS) algorithm for these optimization problems. However, IRLS is computationally expensive since it requires the calculation of the Hessian matrix of second-order derivatives in each iteration. In the experiments in the upcoming chapters, I chose therefore for more efficient non-linear optimization methods such as the scaled conjugate gradient algorithm (Møller 1993). Algorithm 8 on the following page summarizes the EM algorithm for mixtures of experts.

A more detailed treatment of the convergence of the EM algorithm for mixture of experts can be found in (Jordan and Xu 1995).

Algorithm 8 (Generalized) EM algorithm for Mixtures of Experts

Require:

- A training set of N examples: $\{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$.
- Conditional densities $\phi_j(\mathbf{t}^n|\mathbf{x}^n)$ from the exponential family for each of the experts $1 \leq j \leq m$.
- Feed-forward networks for the experts and the gate.

Initialize the expert and gating networks
loop
  {E-step: calculate posteriors (3.26)}
  for $n := 1$ to N **do**
    for $j := 1$ to m **do**
      $\pi_j(\mathbf{x}^n, \mathbf{t}^n) := \frac{g_j(\mathbf{x}^n, \boldsymbol{\theta}_g)\phi_j(\mathbf{t}^n|\mathbf{x}^n, \boldsymbol{\theta}_j)}{\sum_i g_i(\mathbf{x}^n, \boldsymbol{\theta}_g)\phi_i(\mathbf{t}^n|\mathbf{x}^n, \boldsymbol{\theta}_i)}$.
    end for
  end for
  {M-step}
  {Gating network: decrease $E_{\text{gate}}(\boldsymbol{\theta}_g)$ (3.27)}
  Update $\boldsymbol{\theta}_g$ by calculating the gradient of $E_{\text{gate}}(\boldsymbol{\theta}_g)$
  for $j := 1$ to m **do**
    {Expert j: decrease $E_j(\boldsymbol{\theta}_j)$ (3.27)}
    Update $\boldsymbol{\theta}_j$ by calculating the gradient of $E_j(\boldsymbol{\theta}_j)$
  end for
end loop

Weighted Least-Squares Algorithms for M-Step

In this section, a simple heuristic to reduce the M-step for MEs with GLM experts and gate to a one-pass calculation (Jordan and Jacobs 1994) is described. The M-step for expert j requires the solution of the following system of normal equations (using (3.27), (3.20), and (3.13)):

$$\sum_n \pi_j(\mathbf{x}^n, \mathbf{t}^n)[f(\mathbf{W}_j\mathbf{x}^n) - \mathbf{t}^n](\mathbf{x}^n)^T = 0 \qquad \text{for all } j. \tag{3.28}$$

The M-step for the gating network involves the solution of (using (3.27), (3.18), and (3.14)):

$$\sum_n [g_j(\mathbf{x}^n, \mathbf{V}) - \pi_j(\mathbf{x}^n, \mathbf{t}^n)](\mathbf{x}^n)^T = 0 \qquad \text{for all } j. \tag{3.29}$$

With a linear activation function f for the expert networks (corresponding to Gaussian conditional densities), (3.28) is a weighted least-squares problem which has an exact solution using pseudo-inverses (Press et al. 1992). Using matrix notation (3.28) can then be written as:

$$\mathbf{X}^T\boldsymbol{\Pi}_j(\mathbf{X}\mathbf{W}_j^T - \mathbf{T}) = 0,$$

where with N training patterns, d network inputs, and C network outputs, $\mathbf{X}$ is the data matrix of size $N \times d$, $\mathbf{W}_j$ is the weight matrix for expert j with dimensions $C \times d$, $\mathbf{T}$ is the target output matrix of size $N \times C$, and $\boldsymbol{\Pi}_j$ is the diagonal matrix of the posteriors π_j of size $N \times N$. The one-step solution of this equation is:

$$\mathbf{W}_j^T = (\mathbf{X}^T\boldsymbol{\Pi}_j\mathbf{X})^{-1}\mathbf{X}^T\boldsymbol{\Pi}_j\mathbf{T}. \tag{3.30}$$

In order to avoid problems with singularities (of the matrix $\mathbf{X}^T\mathbf{\Pi}_j\mathbf{X}$), it can be solved using *singular value decomposition* (SVD) (Press et al. 1992, section 15.4), which directly finds a solution in the least-squares sense of the associated system of linear equations:

$$\sqrt{\mathbf{\Pi}_j}\mathbf{X}\mathbf{W}_j^T = \sqrt{\mathbf{\Pi}_j}\mathbf{T}.$$

For the gating network and for experts with a softmax activation function, the non-linearity makes that the least-squares approach cannot be applied directly. However, an approximate solution can be obtained by inverting the softmax function and optimizing the weights such that they solve a weighted least-squares problem on the outputs *before* thresholding. Inverting the softmax:

$$y_i = \frac{\exp(a_i)}{\sum_j \exp(a_j)} \qquad \text{gives} \qquad a_i = \ln(y_i) + \ln \sum_j \exp(a_j),$$

where the second term is constant for all a_i and disappears when the softmax is applied. This means that (3.29) can be approximated as a least-squares problem by taking the logarithm of the posteriors (using (3.14)):

$$\sum_n \left[\mathbf{v}_j^T\mathbf{x}^n - \ln \pi_j(\mathbf{x}^n, \mathbf{t}^n)\right](\mathbf{x}^n)^T = 0,$$

Similarly, for the expert networks by taking the logarithm of the target outputs in (3.28):

$$\sum_n \pi_j(\mathbf{x}^n, \mathbf{t}^n)(\mathbf{W}_j\mathbf{x}^n - \ln \mathbf{t}^n)(\mathbf{x}^n)^T = 0.$$

The exact solution is then, for the gating network:

$$\mathbf{v}_j^T = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T \ln(\boldsymbol{\pi}_j),$$

and for the expert networks:

$$\mathbf{W}_j^T = (\mathbf{X}^T\mathbf{\Pi}_j\mathbf{X})^{-1}\mathbf{X}^T\mathbf{\Pi}_j \ln(\mathbf{T}).$$

Finally, a last obstacle for the application of this technique is that we have to avoid taking the logarithm of zero values in $\mathbf{T}$ and $\mathbf{\Pi}_j$ by thresholding them away from zero. Of course, also in this case the weighted least-squares problems can be solved with SVD to avoid numerical instability.

3.5 Estimating Posterior Probabilities

It is well-known that in the limit of infinite data and for sufficiently powerful models, when minimizing sum-of-squares or cross-entropy error functions for classification problems, the optimal outputs approximate the posterior probabilities of class membership. As mentioned in section 3.2, for classification problems it is a direct consequence of the optimal estimate being the conditional mean $\langle t|\mathbf{x}\rangle$ (3.7). This has been rediscovered again and again over the last 25 years and in different contexts (Bourlard and Morgan 1994; Duda and Hart 1973; Hampshire and Pearlmutter 1990; Richard and Lippmann 1991; Ruck et al. 1990; Wan 1990). This property is a useful one, especially when the network outputs are to be used in a further decision-making stage (for example, rejection thresholds) or integrated in other statistical pattern recognition methods.

The ME error function (3.15) can be seen as a generalization of the sum-of-squares and cross-entropy error functions which arise in the special case of a ME with only one expert network. The purpose of this section is to show that also at the global minimum of the ME error function, the optimal ME outputs estimate posterior probabilities of class membership. The proof is along the lines of (Bishop 1995, section 6.1.3) for the sum-of-squares error function and based on (Moerland 1997b; Moerland 1997a).

The ME error function (3.15) was defined as:

$$E(\boldsymbol{\theta}) = -\sum_n \ln \sum_{j=1}^m g_j(\mathbf{x}^n, \boldsymbol{\theta}_g)\phi_j(\mathbf{t}^n|\mathbf{x}^n, \boldsymbol{\theta}_j).$$

The proof I gave in (Moerland 1997a) was entirely based on this error function and got quite lengthy because of the logarithm in front of the sum. It can be considerably simplified when using the complete error function (3.27) derived in the context of the EM algorithm for mixtures of experts:

$$E_c = -\sum_n \sum_{j=1}^m \pi_j(\mathbf{x}^n, \mathbf{t}^n) \ln g_j(\mathbf{x}^n, \boldsymbol{\theta}_g) \;-\; \sum_n \sum_{j=1}^m \pi_j(\mathbf{x}^n, \mathbf{t}^n) \ln \phi_j(\mathbf{t}^n|\mathbf{x}^n, \boldsymbol{\theta}_j) = E_{\text{gate}} + \sum_{j=1}^m E_j.$$

Since we know that minimizing the complete error function also minimizes the original ME error function (section 2.1), the rest of the proof will be done in terms of E_c.

In the limit of an infinite data set the finite sum over the patterns (when divided by the number of patterns) can be replaced by an integral:

$$E_c = -\iint \Big[\sum_{j=1}^m \pi_j(\mathbf{x}, \mathbf{t}) \ln\{g_j(\mathbf{x})\phi_j(\mathbf{t}|\mathbf{x})\}\Big]\, p(\mathbf{t}, \mathbf{x})\, d\mathbf{t}\, d\mathbf{x},$$

factoring the joint distribution:

$$E_c = -\iint \Big[\sum_{j=1}^m \pi_j(\mathbf{x}, \mathbf{t}) \ln\{g_j(\mathbf{x})\phi_j(\mathbf{t}|\mathbf{x})\}\Big]\, p(\mathbf{t}|\mathbf{x})p(\mathbf{x})\, d\mathbf{t}\, d\mathbf{x}. \tag{3.31}$$

The interpretation of the ME outputs when this error function is minimized, can be obtained by setting to zero, the functional derivatives (Bishop 1995, Appendix D; Jost and Li-Jost 1998) of E_c with respect to the gating network outputs $g_j(\mathbf{x})$ and the expert network outputs $y_{jc}(\mathbf{x})$. The solutions of these equations are expressions for $g_j(\mathbf{x})$ and $\mathbf{y}_j(\mathbf{x})$ at the minimum of E_c. The use of functional derivatives is based on the assumption that the expert and gating networks have sufficient functional capacity to model these optimal estimates and we come back to it at the end of this section.

Gating Network

The functional derivative of (3.31) with respect to the gating outputs g_j has to be constrained since the gating outputs should sum to one: $\sum_j g_j(\mathbf{x}) = 1$. This can be done with a Lagrange multiplier λ and leaving out the terms which do not depend on g_j, the Lagrangian becomes:

$$L(\boldsymbol{g}, \lambda) = \lambda\big[\sum_j g_j(\mathbf{x}) - 1\big] - \iint \Big[\sum_{j=1}^m \pi_j(\mathbf{x}, \mathbf{t}) \ln g_j(\mathbf{x})\Big]\, p(\mathbf{t}|\mathbf{x})p(\mathbf{x})\, d\mathbf{t}\, d\mathbf{x}.$$

The functional derivative set to zero with respect to g_j is:

$$\frac{\delta L(\boldsymbol{g}, \lambda)}{\delta g_j(\mathbf{x})} = \lambda - \int \frac{\pi_j(\mathbf{x}, \mathbf{t})}{g_j(\mathbf{x})} p(\mathbf{t}|\mathbf{x})p(\mathbf{x})\, d\mathbf{t} \;= 0 \qquad \text{for all } j.$$

Summing over j and using the constraint on g_j and the fact that the posteriors sum to one, we find $\lambda = p(\mathbf{x})$. Substituting this value in the above equation gives the optimal gating network outputs:

$$g_j(\mathbf{x}) = \int \pi_j(\mathbf{x}, \mathbf{t})\, p(\mathbf{t}|\mathbf{x})\, d\mathbf{t}. \tag{3.32}$$

This has the natural interpretation of the conditional mean of the posteriors.

Experts: Gaussian Conditional Density

Assume that the experts have a linear activation function and a Gaussian conditional density:

$$\phi_j(\mathbf{t}^n|\mathbf{x}^n) = \frac{1}{(2\pi)^{C/2}} \exp\left[-\frac{||\mathbf{t} - \mathbf{y}_j(\mathbf{x})||^2}{2}\right], \tag{3.33}$$

where C is the dimensionality of $\mathbf{t}$. The functional derivative of (3.31) with respect to the expert network outputs $y_{jc}(\mathbf{x})$ is, by cancelling out the logarithm against the exponential:

$$\frac{\delta E_c}{\delta y_{jc}(\mathbf{x})} = \frac{1}{2} \int \pi_j(\mathbf{x}, \mathbf{t})\, [y_{jc}(\mathbf{x}) - t_c]\, p(\mathbf{t}|\mathbf{x}) p(\mathbf{x})\, d\mathbf{t} \;= 0.$$

Therefore, at the minimum of E_c the expert outputs satisfy:

$$y_{jc}(\mathbf{x}) = \frac{\int \pi_j(\mathbf{x}, \mathbf{t}) t_c\, p(\mathbf{t}|\mathbf{x})\, d\mathbf{t}}{\int \pi_j(\mathbf{x}, \mathbf{t})\, p(\mathbf{t}|\mathbf{x})\, d\mathbf{t}}. \tag{3.34}$$

This is the conditional mean of the target outputs but, as one would expect, weighted by the posteriors π_j.

Experts: Multinomial Conditional Density

Assume that the experts have a softmax activation function and a multinomial conditional density:

$$\phi_j(\mathbf{t}^n|\mathbf{x}^n) = \prod_{c=1}^{C} (y_{jc}^n)^{t_c^n}. \tag{3.35}$$

The functional derivative of (3.31) with respect to the expert outputs y_{jc} has to be constrained since the outputs should sum to one: $\sum_c y_{jc}(\mathbf{x}) = 1$. Introducing a Lagrange multiplier for the constraint and leaving out the terms independent of y_{jc}, gives:

$$L(\mathbf{y}_j, \lambda) = \lambda\big[\sum_c y_{jc}(\mathbf{x}) - 1\big] - \iint \Big[\sum_{j=1}^{m} \pi_j(\mathbf{x}, \mathbf{t}) \ln \phi_j(\mathbf{t}|\mathbf{x})\Big]\, p(\mathbf{t}|\mathbf{x}) p(\mathbf{x})\, d\mathbf{t}\, d\mathbf{x}.$$

Substituting the multinomial density (3.35):

$$L(\mathbf{y}_j, \lambda) = \lambda\big[\sum_c y_{jc}(\mathbf{x}) - 1\big] - \iint \Big[\sum_{j=1}^{m} \pi_j(\mathbf{x}, \mathbf{t}) \Big\{\sum_c t_c \ln y_{jc}(\mathbf{x})\Big\}\Big]\, p(\mathbf{t}|\mathbf{x}) p(\mathbf{x})\, d\mathbf{t}\, d\mathbf{x}.$$

The functional derivative with respect to $y_{jc}(\mathbf{x})$ is:

$$\frac{\delta L(\mathbf{y}_j, \lambda)}{\delta y_{jc}(\mathbf{x})} = \lambda - \int \pi_j(\mathbf{x}, \mathbf{t}) \frac{t_c}{y_{jc}(\mathbf{x})} p(\mathbf{t}|\mathbf{x}) p(\mathbf{x})\, d\mathbf{t} \;= 0 \qquad \text{for all } c.$$

If we sum over c and use the constraint on y_{jc} and the fact that the target outputs sum to one (1-of-C coding), we find $\lambda = \int \pi_j(\mathbf{x},\mathbf{t})p(\mathbf{t}|\mathbf{x})p(\mathbf{x})\,d\mathbf{t}$. Substituting this value in the above equation gives the optimal expert outputs:

$$y_{jc}(\mathbf{x}) = \frac{\int \pi_j(\mathbf{x},\mathbf{t})t_c\, p(\mathbf{t}|\mathbf{x})\,d\mathbf{t}}{\int \pi_j(\mathbf{x},\mathbf{t})\, p(\mathbf{t}|\mathbf{x})\,d\mathbf{t}}, \tag{3.36}$$

as with a Gaussian conditional density (3.34).

Interpretation of Network Outputs

Finally, using (3.32) and (3.34), the output vector of a mixture of experts that minimizes the ME error function is (using (3.12)):

$$y_c(\mathbf{x}) = \sum_j g_j(\mathbf{x}) y_{jc}(\mathbf{x}) = \sum_j \int \pi_j(\mathbf{x},\mathbf{t})t_c\, p(\mathbf{t}|\mathbf{x})\,d\mathbf{t},$$

exchanging integration and summation:

$$\int \sum_j \pi_j(\mathbf{x},\mathbf{t})t_c\, p(\mathbf{t}|\mathbf{x})\,d\mathbf{t} = \int t_c\, p(\mathbf{t}|\mathbf{x})\,d\mathbf{t} := \langle t_c|\mathbf{x}\rangle, \tag{3.37}$$

where we have used that the posterior probabilities $\pi_j(\mathbf{x},\mathbf{t})$ sum to one. The interpretation of (3.37) is that the output $y_c(\mathbf{x})$ of a ME at the minimum of the ME error function is equal to the conditional average of the target data as for the outputs of a network trained by minimizing the sum-of-squares or cross-entropy error functions. As we saw in section 3.2 for a classification problem with 1-of-C coding, the conditional average of the target data is:

$$y_c(\mathbf{x}) = \langle t_c|\mathbf{x}\rangle = P(\mathcal{C}_c|\mathbf{x}),$$

so that the outputs of a ME do indeed estimate the posterior probability that $\mathbf{x}$ belongs to class $\mathcal{C}_c$.

Discussion

This is only a rather weak consistency result based on the assumption that there is an unlimited amount of the data and that both experts and gates have sufficient functional capacity. The latter excludes GLMs and would imply the need for MLPs, for example.

More relevant is the case in which there is only a limited amount of data available. This has been studied only recently for mixtures of experts (Jiang and Tanner 1999b; Zeevi, Meir, and Maiorov 1998). Zeevi et al. (1998) have investigated a mixture of linear GLMs. They show that such a simple ME can approximate a wide class of smooth functions and they give bounds which depend on the number of experts. With respect to consistency, they show that the conditional mean can be estimated consistently with increasing sample size when minimizing the sum-of-squares.

It has been shown that useful results can also be obtained within the maximum likelihood framework for conditional mixture models described in this chapter (Jiang and Tanner 1999a; Jiang and Tanner 1999b). They consider both hierarchical and standard MEs with general GLMs as experts. The bounds are similar to the ones obtained by Zeevi et al. (1998). Moreover, they also provide bounds for the estimation of unimodal conditional densities from the exponential family.

CHAPTER 4

Localized Mixtures of Experts

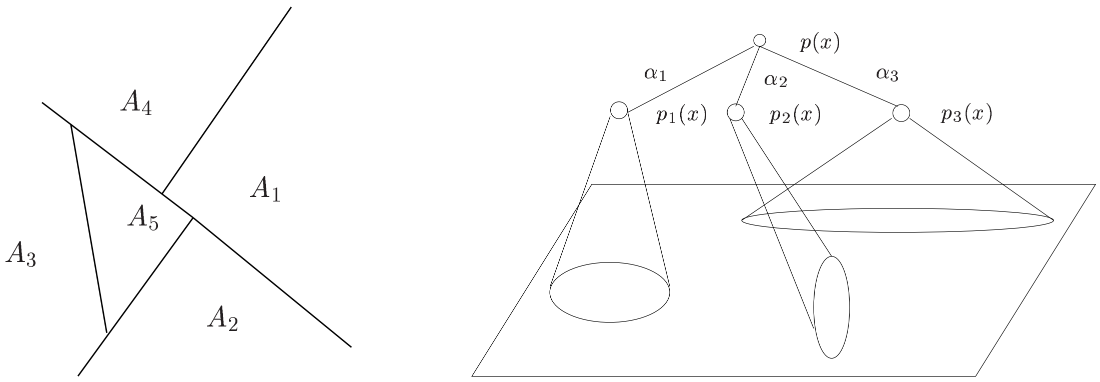
The focus of Chapter 2 was on the use of unsupervised mixture models for density estimation and feature extraction. The promising results obtained with mixtures of latent variable models motivate this chapter in which such mixture models are used in Bayes classifiers and as a gating network in a mixture of experts model.

Roadmap

As we saw in the previous chapter, a mixture of experts consists of a gating network which learns to partition the data space and of experts networks attributed to these different regions. We start with a discussion on the choice of the gating network in a mixture of experts. In Chapter 3, it has been assumed that the gate is a feed-forward neural network (NN). It is motivated by an illustrative example that it might be worthwhile to try another type of gate based on an unsupervised mixture model (section 4.1). Such a *localized* gating network was first proposed by Xu et al. (1995) based on GMMs. Section 4.2 gives a derivation of the EM algorithm for a localized mixture of experts. This general derivation clearly shows that one can choose any type of mixture model as gating network and enables us to use not only GMMs but also mixtures of latent variable models. It is shown that the M-step for the mixture-based gating network is one-pass and almost identical to the unsupervised case.

The rest of this chapter consists of a series of experiments on the same collection of about 20 data sets as in Chapter 2 but this time for supervised learning of classification problems. We start with an experimental evaluation of mixture models in Bayes classifiers following the general scheme outlined in section 3.1. The class-conditional densities are modeled by GMMs, MPCAs, and MFAs and the results obtained on the benchmark problems often are surprisingly good compared with the ones obtained with MEs in section 4.4. Bayesian MPCAs and MFAs (section 2.7) are again shown to offer a viable way of selecting the dimensionality of latent space. This is especially interesting for Bayes classifiers with mixture models because the Bayesian framework enables the model to select a different dimensionality for each mixture component in each class-conditional density.

Section 4.4 compares the localized model with GMM, MPCA, and MFA gates with standard mixtures of experts having single or multi-layer perceptrons as gating network (Moerland 1999a; Moerland 1998). Standard MEs clearly outperform the localized ones and some possible explanations are given. The chapter ends with a short discussion on the possibility of performing localized dimensionality reduction with a localized mixture of experts with a MPCA gate.



(a) Convex division: GLM gate.

(b) Ellipsoidal division: gate based on a mixture model.

Figure 4.1: Two different divisions of the data space.

4.1 Choice of the Gating Network

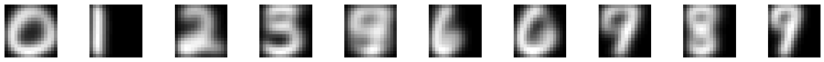
Since the gating network in a mixture of experts deals with the decomposition into smaller tasks, the choice of the type of gating network is an important one. The standard mixture of experts model as defined in the previous chapter, has a GLM with a softmax activation function as gate (Jordan and Jacobs 1994). This leads to a division of the data space by soft hyper-planes with decision boundaries which are simply connected and convex (see the left panel of Figure 4.1) which might seem too limited an approach. An alternative is the use of what Weigend et al. (1995) coined *gated experts*. In this model, the gate is a MLP with a softmax output activation function. This enables far more complex decompositions with non-linear decision boundaries. A third approach is a hierarchical mixture of experts (HME) (Jordan and Jacobs 1994) which has a tree structure. The leaves of the tree contain the expert networks and the non-terminal nodes contain the gating networks. This model also enables complex decompositions while using simple gating networks. Finally, it is also possible to divide the data space with soft hyper-ellipsoids using normalized Gaussian kernels (Xu et al. 1995), each localized to a specific expert (right panel of Figure 4.1). That is, the j-th output gating network takes the following form (the counterpart of (3.11)):

$$g_j(\mathbf{x}, \boldsymbol{\theta}_g) = \frac{\alpha_j p_j(\mathbf{x}, \boldsymbol{\theta}_j)}{\sum_i \alpha_i p_i(\mathbf{x}, \boldsymbol{\theta}_i)}, \tag{4.1}$$

which is normalized to ensure a probabilistic interpretation and where $p_j(\mathbf{x})$ is a Gaussian distribution. The gating network can thus be interpreted as a GMM where each component of the mixture model defines the region of "influence" of the corresponding expert. Xu et al. (1995) derived an EM algorithm for this type of mixtures of experts and obtained promising results on some toy problems. This has been confirmed on some isolated problems by Fritsch (1996) and Ramamurti and Ghosh (1999). Since this type of gate decomposes the data space with soft hyper-ellipsoids, I will refer to the whole model as a *localized* mixture of experts. The resulting model bears some resemblance to a normalized radial basis function network (Moody and Darken 1989), the main difference being that in that case the expert output is a constant value.

As we will see, this localized approach is in no way limited to GMMs as a gating network and may be applied to any mixture model which can be trained with the EM algorithm. In Chapter 2, mixtures

Centers of a 10-component 16-factor MPCA



Digit										
0	812	0	1	5	17	95	612	0	15	0
1	1	1570	2	0	0	119	0	0	5	1
2	53	0	1276	6	35	18	2	0	71	0
3	27	0	30	814	255	41	2	0	411	0
4	3	0	11	0	458	88	0	503	121	247
5	9	0	1	619	371	20	1	0	272	3
6	131	0	1	2	23	697	587	0	19	0
7	2	0	27	0	238	3	0	690	6	603
8	11	0	22	52	228	138	2	6	969	56
9	0	0	3	0	72	12	0	480	27	895

Figure 4.2: Density estimation with a 10-component 16-factor MPCA trained on the NIST data set. Each pattern is assigned to the component (illustrated by its center) on which it has the highest likelihood score $p_j(\mathbf{x}^n)$ and the total number of such examples for a digit class is listed below each center.

of latent variable models were shown to be a flexible alternative to GMMs in density estimation and, therefore, a potential candidate for a localized gating network. Do mixtures of latent variable models indeed find interesting decompositions of the problem which can be easily solved by simple expert networks? Are they also a flexible alternative to GMMs in the context of supervised learning? Is the localized approach better than having a feed-forward NN as gate? These questions will be explored in more detail in the rest of this chapter but first I illustrate the potential of a localized gate with an example.

The example shows that the unsupervised learning of a mixture model may find a useful decomposition of the problem (Figure 4.2). I trained a 10-component 16-factor MPCA on the NIST data (Appendix C) resulting in a model with the 10 centers shown in the figure.[1] Most centers look like blurred or merged instances of the digits.[2] This is exemplified when evaluating the likelihood of a digit under each of the 10 component densities $p_j(\mathbf{x})$ and assigning a digit to the mixture component which maximizes its likelihood. This may be interpreted as a hard decision and a separate classifier could be trained on the examples attributed to each center. As can be seen by inspecting the columns of Figure 4.2, this indeed leads to a simpler classification problem in the majority of the cases. The column of the seventh center, for example, virtually boils down to the two-class problem of distinguishing zeros from sixes. This is a problem which most likely can be handled with a simple model and which might even be linearly separable. Columns 5 and 6, however, do not lead to a nice decomposition with at least five classes being present with many examples. A localized gate as in (4.1) would do this but in a more sophisticated way. Firstly, the definition of the localized gate does not lead to a hard partitioning of the data space but to a soft split with cooperation between experts. Secondly, a localized mixture of experts is trained as a whole and thus it is expected that the localized gate positions itself so as to ease the task of the experts.

[1] This is not a Bayes classifier! The entire data set was used to train the mixture.

[2] For those who wonder, I used pre-processed data as it was available at IDIAP and apparently the data has been aligned to the left.

4.2 EM Algorithm for Localized Mixtures of Experts

In this section, an EM algorithm for localized mixtures of experts is derived which is a slightly generalized version of what has been proposed in (Xu, Jordan, and Hinton 1995). As already said before, a localized gating network consists of normalized kernels each assigned to a specific expert:

$$g_j(\mathbf{x}) = P(j|\mathbf{x}) = \frac{\alpha_j p_j(\mathbf{x})}{\sum_i \alpha_i p_i(\mathbf{x})}, \tag{4.2}$$

where $\sum_i \alpha_i = 1$, $\alpha_i \geq 0$, and p_i a probability density function; thus the gating network outputs g_j sum to one and are non-negative. The numerator of (4.2) can be interpreted as the component of a simple mixture model. This choice of the gating outputs leads to the following conditional probability density for a mixture of experts (substituting (4.2) in definition (3.10)):

$$p(\mathbf{t}|\mathbf{x}) = \sum_{j=1}^{m} \frac{\alpha_j p_j(\mathbf{x})}{\sum_i \alpha_i p_i(\mathbf{x})} \phi_j(\mathbf{t}|\mathbf{x}). \tag{4.3}$$

If we now go back to the expected complete error function derived for a mixture of experts (3.27), we see that the M-step for the localized gating network has to minimize:

$$E_{\text{gate}} = -\sum_n \sum_{j=1}^{m} \pi_j(\mathbf{x}^n, \mathbf{t}^n) \ln \frac{\alpha_j p_j(\mathbf{x})}{\sum_i \alpha_i p_i(\mathbf{x})}, \tag{4.4}$$

which still contains the logarithm of a sum and thus calls for a non-linear optimization method. It would, however, be nice if we could in some way use the EM algorithm for unsupervised mixture models and reduce the M-step for the gating network to a one-pass calculation. Jordan and Xu (1995) realized that this can be done by performing maximum likelihood estimation not on the conditional density (4.3) but on the joint density:

$$p(\mathbf{x}, \mathbf{t}) = p(\mathbf{t}|\mathbf{x})p(\mathbf{x}) = \sum_{j=1}^{m} \alpha_j p_j(\mathbf{x}) \phi_j(\mathbf{t}|\mathbf{x}),$$

which by maximum likelihood leads to the following error function on the training data $\{(\mathbf{x}^n, \mathbf{t}^n)\}$:

$$E = -\sum_n \ln \sum_{j=1}^{m} \alpha_j p_j(\mathbf{x}^n) \phi_j(\mathbf{t}^n|\mathbf{x}^n). \tag{4.5}$$

The E-step then consists of calculating the posteriors of the hidden variables which follows directly from (3.26) by substituting (4.2) for $g_j(\mathbf{x}^n)$:

$$\text{E-step:} \qquad \mathcal{E}(z_j^n) = \pi_j(\mathbf{x}^n, \mathbf{t}^n) = \frac{\alpha_j p_j(\mathbf{x}^n) \phi_j(\mathbf{t}^n|\mathbf{x}^n)}{\sum\limits_{i=1}^{m} \alpha_i p_i(\mathbf{x}^n) \phi_i(\mathbf{t}^n|\mathbf{x}^n)}. \tag{4.6}$$

The expected complete error function corresponding to (4.5) is (using (3.27) and (4.2)):

$$\begin{aligned}\mathcal{E}(E_c) &= -\sum_n \sum_{j=1}^{m} \pi_j(\mathbf{x}^n, \mathbf{t}^n) \ln \{\alpha_j p_j(\mathbf{x}^n)\} \; - \; \sum_n \sum_{j=1}^{m} \pi_j(\mathbf{x}^n, \mathbf{t}^n) \ln \{\phi_j(\mathbf{t}^n|\mathbf{x}^n)\}. \\ &= \quad E_{\text{gate}}(\boldsymbol{\theta}_g) \quad + \quad \sum_{j=1}^{m} E_j(\boldsymbol{\theta}_j).\end{aligned}$$

The M-step consists of minimizing or decreasing this complete error function with respect to the parameters of the expert networks and the gate. Comparing the above equation with (3.27) shows that the expert error functions have the same form and thus the optimization of the expert networks in the M-step is as in section 3.4.2. We focus therefore on the gating error function:

$$E_{\text{gate}} = -\sum_n \sum_{j=1}^{m} \pi_j(\mathbf{x}^n, \mathbf{t}^n) \ln \alpha_j - \sum_n \sum_{j=1}^{m} \pi_j(\mathbf{x}^n, \mathbf{t}^n) \ln p_j(\mathbf{x}^n). \tag{4.7}$$

But this is exactly the complete error function we derived for an unsupervised mixture model (2.13)! The only difference is in the definition of their posteriors $h_j(\mathbf{x}^n)$ (2.11) and $\pi_j(\mathbf{x}^n, \mathbf{t}^n)$ (4.6) respectively. In the case of a localized mixture of experts, the posteriors include both patterns and target outputs reflecting the supervised nature of the model. This gives us the one-pass M-step we were looking for. Moreover, any mixture model which can be trained with EM could be used as a gating network in this framework. This makes it possible to use not only GMMs as in (Xu, Jordan, and Hinton 1995) but also MPCAs and MFAs. These alternative choices for the gating network have been evaluated in the experiments described in section 4.4.

An alternative approach for a virtually similar model with a GMM gate and conditional Gaussian experts has been proposed by Jebara and Pentland (1999). They manage to avoid having to perform ML estimation on the joint density by upper bounding E_{gate} (4.4). This can be done with a simple variational bound of the logarithm $\ln(x) \leq x - 1$:

$$E_{\text{gate}} \leq -\sum_n \sum_{j=1}^{m} \pi_j(\mathbf{x}^n, \mathbf{t}^n) \ln \alpha_j p_j(\mathbf{x}) + \sum_n \sum_{j=1}^{m} \pi_j(\mathbf{x}^n, \mathbf{t}^n) \sum_i \{\alpha_i p_i(\mathbf{x}) - 1\}.$$

The minimization of this upper bound can be done analytically although along the way other bounds have to be introduced. This method has the advantage of still performing ML estimation on the conditional density which is our ultimate goal. Results on toy regression data indicate better performance than when estimating the joint density.

4.3 Experiments: Bayes Classifiers

Before describing the experiments with the localized mixtures of experts, I first report on the results of a series of experiments with mixture models for estimating the class-conditional densities in a Bayes classifier. That is, each class conditional density is modeled by a separate mixture model is:

$$p(\mathbf{x}|\mathcal{C}_k) = \sum_i \alpha_{ki} p_{ki}(\mathbf{x}),$$

and the corresponding Bayes decision rule (3.2):

$$\mathbf{x} \text{ is assigned to } \mathcal{C}_k \quad \Leftrightarrow \quad P(\mathcal{C}_k) \sum_i \alpha_{ki} p_{ki}(\mathbf{x}) \geq P(\mathcal{C}_j) \sum_i \alpha_{ji} p_{ji}(\mathbf{x}) \quad \text{for all } j \neq k. \tag{4.8}$$

Oddly enough, the use of mixture models in Bayes classifiers is not that widespread in the machine learning community, but see for example (Hastie and Tibshirani 1996; Hastie, Tibshirani, and Buja 1999; Kambhatla and Leen 1995).

4.3.1 Experimental Set-Up

The experiments were set up in the same way as those for density estimation with mixture models described in section 2.6.1. The main difference is that we now take the class labels into account and form separate training sets for each class on which to train the mixture models. These separate mixture models were initialized and trained with the EM algorithm as in section 2.6.1.

The NIST and satimage data come with a fixed division in a training and a test set (see Table 2.1) and McNemar's test was used to determine whether the difference in performance between two methods was statistically significant (Dietterich 1998a). This test involves only a single training run and is described in detail in Appendix B. For all other data sets the 5x2cv F test was used with five replications of twofold cross-validation. In each round one third of the training data was set aside for validation purposes (respecting the class distributions) on the entire Bayes classifier. The results on the validation data were used to select the best model for each type of mixture model on a data set. For most data sets, the number of mixture components tried for each class-conditional density had at least five different values ranging from 1 to a value depending on the complexity of the problem but with a maximum of 20. The dimension of latent space for MPCAs and MFAs was varied in a similar way but with a maximum value of 10 (and of course upper-bounded by the dimension of data space).

In each Bayes classifier, the mixture models for each class-conditional density were chosen to be identical, that is, of the same type, with the same number of components and (for the latent variable models) with the same dimension of latent space. Classification of the examples was based on Bayes decision rule (4.8). The priors $P(\mathcal{C}_k)$ were estimated on the basis of the training set as the proportion of samples of class $\mathcal{C}_k$.

4.3.2 Evaluation

The results of the experiments with Bayes classifiers are listed in Table 4.1, where the best method and the ones that are not significantly worse (90% on the 5x2cv F test or McNemar's test) are set underlined. Note that on the pima data set, the results do not allow any conclusion about the relative performance of the different models: all of them perform equally well and no underlining is used in this case. The last row in Table 4.1 gives an idea of the global performance of the different models by specifying the total number of underlined scores for each model type. The number of wins shows that MFAs and MPCAs provide the best Bayes classifiers though their advantage is not as impressive as with unsupervised density modeling (section 2.6). This is related to the fact that in Bayes classifiers the relation between the quantity we are optimizing, viz. the likelihood of the data from a given class, and the ultimate goal, viz. minimizing classification error, is obscure. It can very well happen that the Bayes classifier with highest average likelihood on the test data is not the one that minimizes the classification error.

The mixtures of latent variable models are outperformed twice by mixtures of spherical GMMs on the cancer and dermatology data. This seems to be due to overfitting since the mean classification error on the training set was consistently higher with the more complex mixture models. Spherical and diagonal GMMs give satisfactory results on small data sets. This does not come as a surprise since data can be very scarce in this case with only few examples for each class. The mixtures of latent variable models are also outperformed three times by mixtures of tied or full GMMs. These include data sets for which data is plentiful such as letter and pen on which full GMMs already showed good performance in density estimation (Table 2.4).

It is also interesting to note that in most cases, the best mixture model has several components and performs better than a single component model of the same type. A general remark on these Bayes classifiers is that their classification error often is comparable with other state-of-the-art classifiers. A nice example is the score (95% correct) obtained on letter with a mixture of 20 tied GMMs for each of the 26 classes. The best result obtained in the StatLog project (Michie, Spiegelhalter, and Taylor 1994) in which a host of statistical and machine learning were compared, was 93.6%. This score has been improved to around 98% (Schwenk and Bengio 1998) only very recently with boosting methods such as AdaBoost.

The results on the NIST handwritten digit data are satisfactory as well: the best score of 97.3% was

Data	spherical	diagonal	tied	full	MPCA	MFA
banana	$89.2(1.79)^{5}$	$90.9(1.19)^{5}$	$88.8(1.74)^{5}$	$\underline{93.4}(1.36)^{5}$	$\underline{93.3}(1.80)^{5,1}$	$\underline{93.0}(0.87)^{5,1}$
cancer	$\underline{96.1}(1.22)^{3}$	$95.8(0.82)^{2}$	$94.1(1.26)^{5}$	$94.9(1.04)^{2}$	$95.2(1.10)^{2,3}$	$95.0(1.38)^{1,1}$
dermatology	$\underline{95.7}(1.03)^{1}$	$93.7(2.26)^{2}$	$92.1(1.29)^{3}$	$92.1(1.63)^{1}$	$94.9(1.31)^{1,1}$	$94.1(2.02)^{1,3}$
glass	$61.5(3.87)^{3}$	$63.0(4.43)^{3}$	$\underline{62.6}(5.65)^{5}$	$\underline{65.4}(3.17)^{3}$	$56.9(4.75)^{5,1}$	$\underline{63.5}(4.38)^{2,1}$
heart	$\underline{81.7}(1.95)^{1}$	$\underline{81.7}(2.44)^{1}$	$\underline{77.6}(2.38)^{1}$	$77.5(3.73)^{1}$	$\underline{81.1}(2.33)^{1,1}$	$\underline{81.9}(3.31)^{1,1}$
ionosphere	$90.7(1.33)^{5}$	$89.5(1.49)^{2}$	$85.6(3.58)^{2}$	$85.8(3.65)^{1}$	$\underline{93.9}(1.33)^{1,3}$	$\underline{93.3}(1.28)^{1,3}$
iris	$92.9(3.14)^{3}$	$93.5(2.42)^{1}$	$96.2(2.50)^{1}$	$\underline{95.8}(2.47)^{2}$	$\underline{96.5}(1.79)^{1,3}$	$\underline{96.9}(1.99)^{1,1}$
letter	$84.9(0.35)^{20}$	$86.9(0.72)^{20}$	$\underline{95.0}(0.29)^{20}$	$93.7(0.36)^{10}$	$92.0(0.59)^{10,5}$	$94.1(0.27)^{10,5}$
NIST	94.2^{20}	95.7^{40}	–	96.7^{10}	$\underline{97.1}^{10,5}$	$\underline{97.3}^{20,10}$
optical	$93.8(0.61)^{10}$	$93.1(0.67)^{5}$	$94.6(0.61)^{2}$	$94.6(0.68)^{1}$	$\underline{95.9}(0.60)^{3,5}$	$95.0(0.51)^{2,5}$
pen	$97.1(0.50)^{10}$	$96.6(0.35)^{10}$	$98.6(0.13)^{10}$	$\underline{99.2}(0.15)^{10}$	$98.3(0.21)^{5,3}$	$98.5(0.20)^{10,5}$
pima	$73.6(1.13)^{1}$	$74.6(1.23)^{1}$	$73.3(1.97)^{1}$	$73.9(1.50)^{1}$	$74.5(1.33)^{1,5}$	$72.5(2.19)^{3,1}$
satimage	86.1^{10}	87.6^{10}	87.7^{10}	84.9^{3}	$\underline{88.9}^{10,5}$	$87.2^{10,3}$
segmentation	$89.1(0.76)^{10}$	$89.7(3.95)^{10}$	$\underline{91.4}(1.77)^{10}$	$\underline{92.6}(0.85)^{10}$	$89.2(2.12)^{5,5}$	$\underline{91.6}(0.95)^{10,3}$
sonar	$\underline{75.8}(3.93)^{5}$	$\underline{74.4}(6.03)^{2}$	$63.3(1.94)^{3}$	$70.4(3.54)^{5}$	$\underline{78.1}(4.10)^{1,3}$	$\underline{81.6}(6.32)^{2,3}$
soybean	$85.3(3.15)^{2}$	$\underline{90.7}(2.02)^{2}$	$\underline{91.5}(1.11)^{1}$	$\underline{91.6}(0.93)^{1}$	$89.0(1.73)^{1,5}$	$\underline{92.9}(1.82)^{1,1}$
vowel	$72.7(2.58)^{5}$	$75.6(2.11)^{5}$	$81.8(1.96)^{5}$	$\underline{88.0}(2.13)^{5}$	$78.6(2.19)^{2,3}$	$85.2(2.51)^{2,5}$
waveform	$\underline{84.1}(1.21)^{2}$	$\underline{81.8}(2.37)^{3}$	$74.7(2.21)^{3}$	$74.3(1.57)^{1}$	$\underline{80.1}(2.55)^{3,1}$	$\underline{83.2}(1.75)^{2,1}$
waveform-noise	$\underline{79.9}(2.71)^{1}$	$\underline{79.2}(2.46)^{1}$	$66.7(2.03)^{3}$	$67.0(1.71)^{1}$	$77.8(2.14)^{1,1}$	$\underline{80.6}(1.48)^{1,1}$
wins	6	5	5	7	9	11

Table 4.1: Results of the experiments with Bayes classifiers with different mixture models for modeling the class-conditional densities. Scores are in percentage of correct classification on the test set and are the average over 10 experiments in a 5x2cv F test framework or a single run for NIST and satimage. The standard deviation is given between parentheses. Each of the scores corresponds to the best model out of the particular class of models as selected on validation data. The first superscript indicates the number of mixture components of each class conditional density in the best model. For mixtures of latent variable models, the second superscript specifies the dimension of latent space. The underlined scores are the ones that do not pass the 5x2cv F test or McNemar's test with 90% confidence when compared with the model having the best score.

obtained by modeling each of the ten digits with a 10-component 10-factor MFA. This is comparable to the best scores I obtained with large MLPs. Figure 4.3 shows the centers of each of these mixture models. It can be seen that the mixture model succeeded in learning variations in writing style. Some centers of the model for a "two", for example, correspond to a "two" with curl and others to a "two" without curl. This example illustrates that Bayes classifiers based on mixture models not only provide fast training and good performance but also solutions which are relatively easy to interpret.

4.3.3 Bayesian MPCAs and MFAs in Bayes Classifiers

In section 2.7, we saw that it is possible to train mixtures of latent variable models in a Bayesian framework such that the appropriate dimensions of latent space are determined automatically. This can be especially interesting in the context of Bayes classifiers since in the above experiments we assumed that the number of factors is equal for all mixture components and for each class. This was necessary to keep model selection on validation data tractable. One would expect that within the Bayesian framework results can be obtained which are at least as good since the Bayesian framework enables us to cover the whole model space. Note that ideally, we would also like to automatically

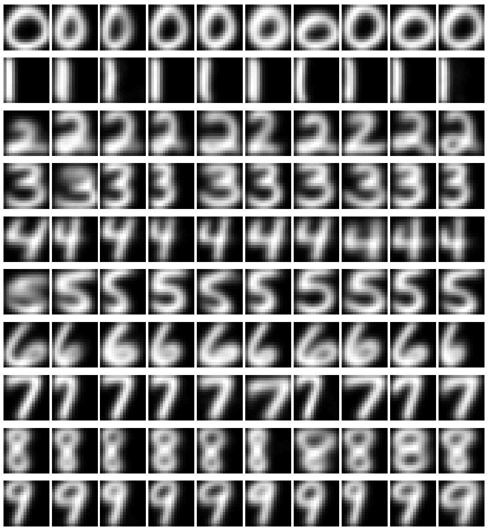


Figure 4.3: Means of a 10-component 10-factor MFA for each digit of the NIST data in a Bayes classifier.

select the number of mixture components for each class. This is another free parameter which was assumed to be equal for all class-conditional densities in the above experiments. A heuristic Bayesian way to select the number of components has been proposed recently for MFAs (Ghahramani and Beal 2000) (see also the discussion at the end of section 2.7). In the experiments I did not pursue this idea and fixed the number of mixture components.

The set-up of the experiments is almost the same as the framework described in section 4.3.1. The main difference is that for each mixture model 40 iterations of EM were performed with the new updates for the weight matrices: (2.62) for MPCAs and (2.61) for MFAs. After each fifth iteration of the EM algorithm, the hyperparameters $\boldsymbol{\gamma}$ were re-estimated (2.57).

The results for Bayes classifiers with MPCAs are listed in Table 4.2. For convenience, the results for the best ML mixtures found in Table 4.1 have been copied here together with the number of mixture components and the dimension of latent space. The Bayesian MPCAs had the same number of mixture components as their ML counterparts and a latent dimension of $\ell = d-1$. For the high-dimensional NIST, optical, sonar, and soybean data sets, a lower value of ℓ was chosen. This was done partly to save computation time and partly to avoid problems with the cheap and cheerful approximation already outlined in section 2.7.[3] The results for Bayes classifiers with MFAs are in Table 4.3. The set-up was as with the Bayesian MPCAs and again the ML scores have been copied from Table 4.1.

When comparing the classification scores of the ML and the Bayesian models, one observes that in most cases the difference is quite small (and very likely to be not significant). This does not show the improvement we were hoping for by having a more fine-grained model search. However, Bayesian estimation is certainly a useful and efficient approach since cross-validation is only needed for selecting the number of mixture components. The average selected dimensionality over all mixture components and all classes with Bayesian mixtures is given in the last column of Tables 4.2 and 4.3. This value is often quite close to the value selected on validation data with ML, especially for MFAs. With MPCAs the selected number of factors is in general higher than with ML.

[3]The value of ℓ was again selected on validation data, which is not very satisfactory since this is exactly what we wanted to avoid in the Bayesian framework! Luckily it is only an upper bound on the dimensions of latent space that can be selected in the Bayesian framework.

Data	# components	ML MPCA classification (%)	ML MPCA # factors	Bayesian MPCA classification (%)	Bayesian MPCA # factors
banana	5	93.3(1.80)	1	92.5(1.71)	0.7/1
cancer	2	95.2(1.10)	3	95.7(0.71)	4.1/9
dermatology	1	94.9(1.31)	1	95.8(1.73)	5.2/33
glass	5	56.9(4.75)	1	59.1(3.71)	0.1/8
heart	1	81.1(2.33)	1	81.2(3.47)	2.3/12
ionosphere	1	93.9(1.33)	3	89.3(3.22)	20.0/33
iris	1	96.5(1.79)	3	96.0(3.07)	2.0/3
letter	10	92.0(0.59)	5	93.1(0.49)	5.8/15
NIST	10	97.1	5	97.3	18.5/20
optical	3	95.9(0.60)	5	96.8(0.34)	11.6/15
pen	5	98.3(0.21)	3	98.7(0.25)	9/15
pima	1	74.5(1.33)	5	74.0(1.81)	4.0/7
satimage	10	88.9	5	88.8	9.4/35
segmentation	5	89.2(2.12)	5	87.8(3.04)	4.8/18
sonar	1	78.1(4.10)	3	78.5(3.60)	10.0/30
soybean	1	89.0(2.55)	5	86.4(2.54)	0/50
vowel	2	78.6(2.19)	3	78.9(2.78)	3.6/9
waveform	3	80.1(2.55)	1	83.2(2.48)	2.5/20
waveform-noise	1	77.8(2.14)	1	79.4(2.27)	2.5/39

Table 4.2: A comparison of Bayes classifiers with MPCAs trained with maximum likelihood and in the Bayesian framework of section 2.7. Scores are in percentage of correct classification on the test set and are the average over 10 experiments in a 5x2cv F test set-up or a single run for NIST and satimage. The standard deviation is given between parentheses. The ML score corresponds to the best MPCA model as evaluated on validation data. The second column (# components) indicates the number of mixture components in the "best" model. The fourth column specifies the dimension of latent space of the best model. The Bayesian approach has been applied to a MPCA with the same number of components (as indicated by the second column) and with a number of factors $\ell = d-1$. The final column gives the mean number of factors selected by the ARD prior in the Bayesian approach versus ℓ.

Let us now have a look at a particular example of the dimensionalities of latent space selected in the Bayesian framework. I took the satimage data on which the Bayesian MFA outperformed its ML cousin (89.4% against 87.2%, significance according to McNemar's test 99.7%). The Bayes classifier consisted of a 10-component 35-factor MFA for each of the 6 classes in the satimage data. A boxplot of the estimated dimensionalities of latent space for each class-conditional density is shown in Figure 4.4. This clearly indicates that Bayesian mixtures attribute a different number of factors to each mixture component and also allow for differences between the classes.

4.4 Experiments: Mixtures of Experts

In this section, the results of a series of experiments with various types of mixture of experts models are described. The main goal of the experiments was to evaluate the influence of the choice of the gating network on the overall performance of the whole model. Three of the gating networks are of the localized type described earlier and are based on GMMs, MPCAs, and MFAs respectively. These

		ML MFA		Bayesian MFA	
Data	# components	classification (%)	# factors	classification (%)	# factors
banana	5	93.0(0.87)	1	93.1(0.77)	0.7/1
cancer	1	95.0(1.38)	1	94.9(1.07)	4.2/9
dermatology	1	94.1(2.02)	3	95.3(1.51)	0/33
glass	2	63.5(4.38)	1	63.8(4.04)	0.7/8
heart	1	81.9(3.31)	1	81.0(3.57)	2/12
ionosphere	1	93.3(1.28)	3	92.1(2.71)	19.5/33
iris	1	96.9(1.99)	1	96.9(2.74)	1.3/3
letter	10	94.1(0.27)	5	93.9(0.31)	4.4/15
NIST	20	97.3	10	97.6	5.4/30
optical	2	95.0(0.51)	5	96.0(0.38)	7.6/15
pen	10	98.5(0.20)	5	98.3(0.39)	5.4/15
pima	3	72.5(2.19)	1	69.0(4.78)	3.1/7
satimage	10	87.2	3	89.4	8.1/35
segmentation	10	91.6(0.95)	3	90.9(1.24)	3.8/18
sonar	2	81.6(6.32)	3	83.6(1.72)	6.4/30
soybean	1	92.9(1.82)	1	92.7(0.87)	0/50
vowel	2	85.2(2.51)	5	83.1(2.49)	2.7/9
waveform	2	83.2(1.75)	1	84.3(1.48)	0.8/20
waveform-noise	1	80.6(1.48)	1	81.1(1.68)	1.0/39

Table 4.3: A comparison of Bayes classifiers with MFAs trained with maximum likelihood and in the Bayesian framework of section 2.7. See Table 4.2 for more details.

models are compared with the standard ME model in which the gate is a GLM with a softmax activation function and a ME model with a softmax MLP gate. The expert networks were chosen to be GLMs in all cases.

In the existing literature, experiments with localized mixtures of experts based on GMMs have mainly been performed on isolated problems. The paper by Xu et al. (1995) in which the localized model has been proposed, only evaluates the model on a toy problem for regression. Xu's model has also been extended to a constructive approach for training MEs which has been evaluated on several small regression problems (Ramamurti and Ghosh 1999) but almost without comparing it with other models. Various mixture of expert models have been evaluated on a large speech recognition database in a hybrid neural network + hidden Markov model (Fritsch 1996). Hierarchical mixtures of experts, mixtures of experts, gated experts and a mixture of Gaussian experts were used in this hybrid model. The mixture of Gaussian experts is an extension of the localized mixture of experts that also uses Gaussian kernels as experts. An empirical comparison on a handwritten digit problem of localized mixtures of experts trained by gradient-based methods with MLPs and radial basis function networks is given in (Alpaydin and Jordan 1996). Gated experts have not yet been used often and if so, only on isolated problems. Weigend et al. (1995) applied gated experts to several time series problems, especially those with different regimes. As stated before, in (Fritsch 1996) a gated expert model is used on a problem in automatic speech recognition, like in (Waterhouse and Cook 1997) where it also compared with a MLP.

A thorough experimental evaluation of MEs has been done by Waterhouse (1997) in the DELVE framework (Rasmussen et al. 1996). He compared standard MEs with his extensions: a constructive algorithm for hierarchical mixtures of experts and a Bayesian training method for MEs. The choice

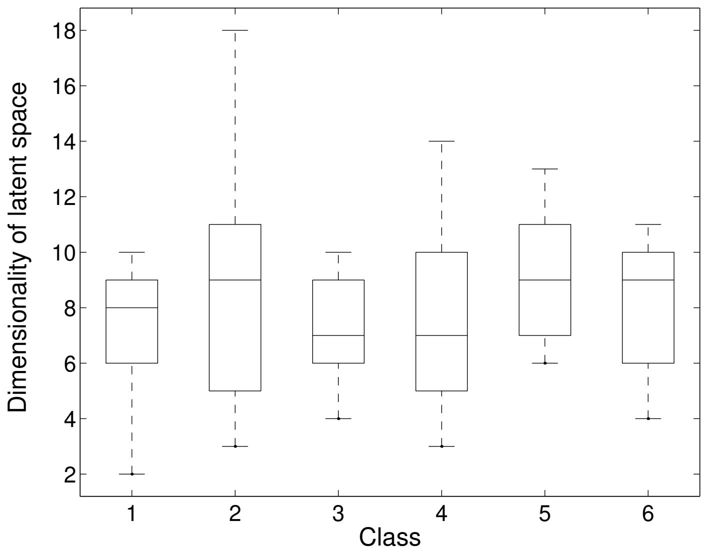


Figure 4.4: Boxplot of the estimated dimension of latent space for a 10-component 35-factor MFAs for each of the 6 classes of the satimage data in a Bayes classifier. Each box has lines at the lower quartile, median and upper quartile values. The whiskers are lines extending from each end of the box to show the extent of the rest of the values.

of the DELVE framework has the advantage that the results can be easily compared with the ones obtained on DELVE with other methods. For classification problems, however, it suffers from a lack of data sets and of results on other state-of-the-art models (see also Appendix B). I therefore chose the experimental set-up which is described in the next section with a larger number of data sets.

4.4.1 Experimental Set-Up

The experiments with the MEs were performed on the classification problems listed in Table 2.1. Pre-processing of the attribute values was done as described in section 2.6.1. The target outputs are based on the 1-of-C coding scheme with one output for each class.

Training of the localized mixtures of experts consisted of two phases. In the first phase, the mixture model for the gating network was initialized and trained in an unsupervised fashion exactly as described in section 2.6.1 to find a good initial configuration. In the second phase the whole model was trained in the EM framework described in section 4.2. The M-step for each of the experts consisted of three iterations of the scaled conjugate gradient algorithm (Møller 1993). The M-step for the gate is as described in section 4.2, based on the M-step for the corresponding mixture model. For the experiments with standard MEs, the initial weights of the experts and the gate were chosen randomly from a Gaussian distribution around zero. The M-step for each of the experts and the gating network again consisted of three iterations of the scaled conjugate gradient algorithm. Early stopping on a validation set (which contains one third of the training data and has balanced classes) was used to avoid overfitting. The expert networks were GLMs with a softmax activation function in all the different types of MEs.

McNemar's test was used for the NIST and satimage data. For all other data sets the 5x2cv F test was used with five replications of twofold cross-validation. The validation data set aside for early stopping was also used to select the best model for each type of ME on a data set. The free

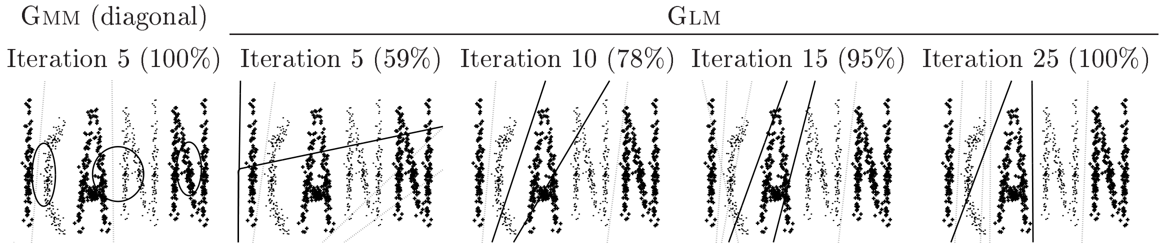


Figure 4.5: Two-class (+ and ·)[5] toy problem. The first solution is found by a localized 3-expert ME with a GMM gate. The Gaussian components of the gating network are represented as ellipses at a distance of one standard deviation. The hyperplanes of the GLM experts (dotted lines) nicely separate the letters to which they are attributed. The next panels show 4 snapshots of training a 4-expert ME with a GLM gate and their predicted values. The hyperplanes of the GLM gate are displayed (solid lines) and are slowly positioned so as to decompose the problem into subproblems which are linearly separable. The hyperplanes of the GLM experts in the last panel solve their respective subproblems.

parameters varied were the number of experts (at least 5 different values), the dimension of latent space for localized MEs with a MPCA or MFA gate, and the number of hidden neurons for a MLP gate.

4.4.2 Evaluation on Artificial Data

The first example is a two-class toy problem in which I generated the acronym "ICANN" from several Gaussians and attributed the "C" and the first "N" to one class and the others to the second class (Figure 4.5). A 3-expert localized ME with a GMM gate trained using the EM algorithm described in section 4.2, quickly found a solution. The Gaussian components of the gating network are positioned in such a way that the subproblems are linearly separable in the most evident way. A 3-expert ME with a GLM gate was not able to find a solution which classified all examples correctly but a 4-expert ME was. However, the number of EM iterations was higher and the decomposition found is less elegant. The solution is shown in the four right-hand panels of Figure 4.5 together with the hyperplanes of the gating network. Interestingly enough the final solution only requires three experts and the fourth expert is never active. In fact, the GLM gate does divide the data space into four regions but their intersections lie far out of the range of the training data.

As a second example, we come back to the waveform and waveform-noise data (the last two rows of Table 2.1) already used in section 2.6.2. The optimal Bayes classification rate for these two toy data sets is about 86%. The results on waveform and waveform-noise are in Tables 4.4 and 4.5. The results are the average over 10 experiments in a 5x2cv F test (90%) framework and standard deviation is given between parentheses.

On both data sets the standard MEs perform best, often significantly better than the localized MEs. To gain some insight in the solutions found by a mixture of 3 experts, Figure 4.6 shows the projection of the waveform data onto its two leading principal components. Each of the 3 classes turns out to lie on the edge of a triangle. The left half of Figure 4.6 illustrates the solution found by a mixture of 3 experts with a MPCA gate. The three clusters found by the localized gate lie close to the vertices of the triangle, and the subproblem solved by each of the experts is therefore effectively reduced to a two-class problem for separating the two edges out of this vertex. The right half of Figure 4.6 shows that the GLM gate did not find a meaningful decomposition of the problem but that

[5]Or dark and light for the less keen-sighted.

Gate	test
Spherical	79.7(3.28)
Diagonal	79.7(2.73)
MFA-2	80.5(2.00)
MFA-1	80.6(2.85)
MPCA-3	80.8(2.16)
Full	<u>81.2</u>(1.40)
MFA-3	<u>81.6</u>(1.39)
MPCA-2	<u>81.8</u>(1.34)
MLP (4)	<u>82.2</u>(1.63)
MPCA-1	<u>82.3</u>(1.28)
Perceptron	<u>83.2</u>(1.16)

Table 4.4: Classification results on waveform with a mixture of 3 experts. Scores are in percentage of correct classification.

Gate	test
Full	76.8(1.38)
Spherical	77.4(1.50)
MFA-1	77.5(1.08)
Diagonal	77.6(1.74)
MFA-2	77.6(0.92)
MPCA-2	78.0(1.66)
MPCA-1	78.1(1.57)
MFA-3	78.2(1.59)
MPCA-3	78.7(0.77)
MLP (4)	<u>79.6</u>(3.44)
Perceptron	<u>81.7</u>(1.02)

Table 4.5: Classification results on waveform-noise with a mixture of 3 experts. Scores are in percentage of correct classification.

this was not necessary anyway. All 3 experts separate the classes more or less nicely and the entire ME performs a sort of averaging which is known to often improve accuracy.

4.4.3 Evaluation on Real-World Data

Experiments were performed on the real-world data sets listed in Table 2.1. For most data sets, the number of experts tried had at least five different values ranging from two to a value depending on the complexity of the problem but with a maximum of 20. The dimension of latent space for MPCAs and MFAs gates was varied in a similar way but with a maximum value of 10 (and of course upper-bounded by the dimension of data space). The number of hidden neurons in the MLP gate was either 5, 10 or 20.

The results of the experiments are shown in Table 4.6, where the best method and the ones which are not significantly worse (90% on the 5x2cv F test or McNemar's test) are set underlined. Note that on eight data sets, the results do not allow any conclusion about the relative performance of the different models. On all remaining data sets, however, a standard ME (with a GLM or MLP gate) is significantly better than most of the localized MEs. In this respect the number of wins specified in the bottom row of Table 4.6 is instructive. When comparing the various localized mixtures of experts, it is clear that the results are not uniform. None of the GMMs or mixtures of latent variable models can be preferred over the others. More specifically, there seems to be no correlation between the performance of a gate as a density estimator in data space (section 2.6) and the classification results in a localized ME. The results illustrate that the more complex decompositions offered by a MLP gate do not make a difference. The scores are almost identical to the ones obtained with a GLM gate.

A criterion for quantifying the difference between feed-forward gates and localized gates is the entropy of the gating outputs:

$$H(\boldsymbol{g}) = -\sum_n \sum_{j=1}^{m} g_j(\mathbf{x}^n) \ln g_j(\mathbf{x}^n),$$

which we can normalize by dividing it by the entropy of the uniform distribution. On almost all data sets, the entropy of the NN gates was bigger than the entropy of the localized gates by at least

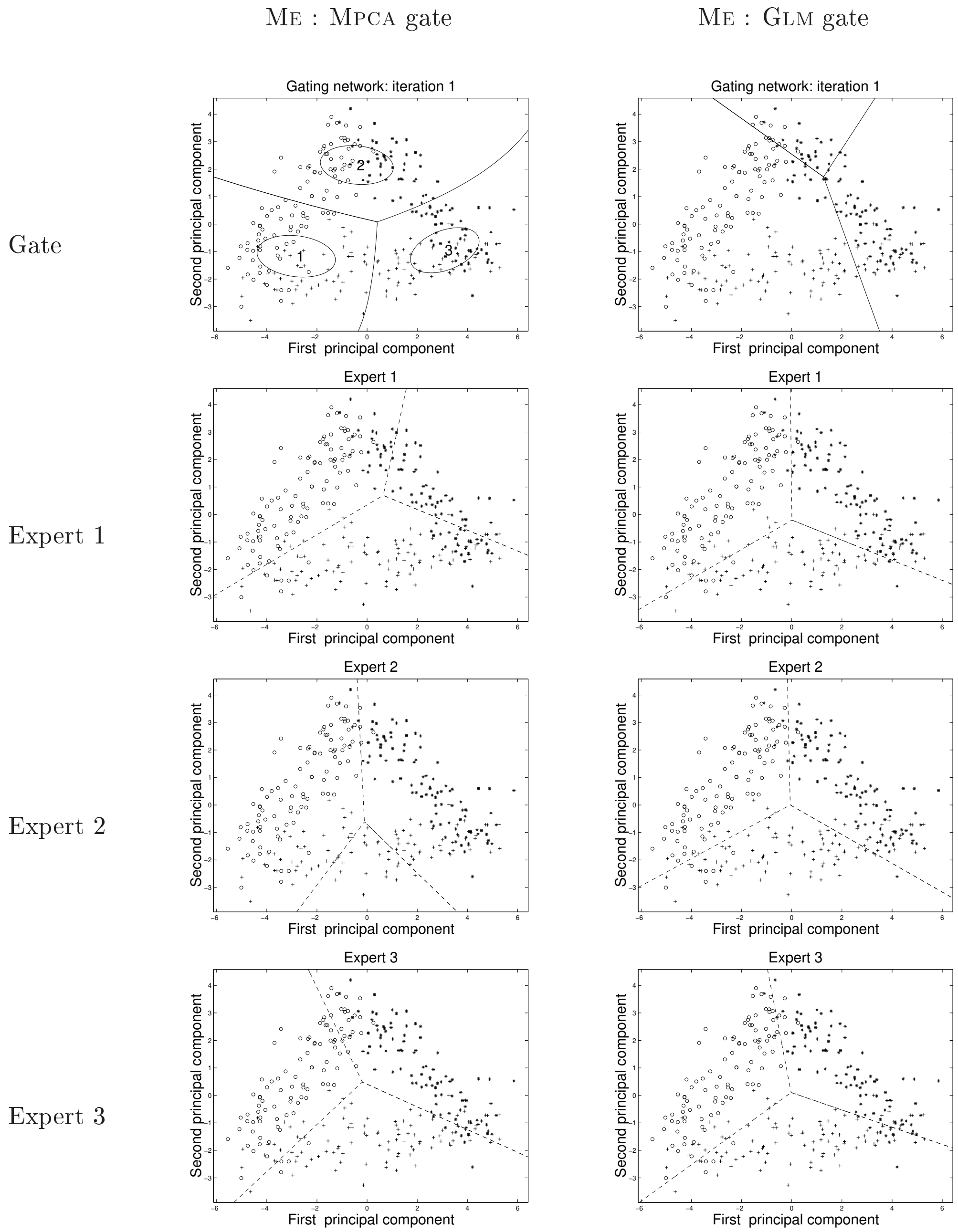


Figure 4.6: The solutions found on the 3-class waveform data by a localized (Mpca with one factor) and a standard mixture of 3 experts. The data has been projected onto its two leading principal components. The Gaussian components of the Mpca gate are shown as ellipses at a distance of one standard deviation. The decision boundaries of the gating networks (solid lines) and of the expert networks (dotted lines) are also displayed.

	GMM						
Data	spherical	diagonal	full	MPCA	MFA	GLM	MLP
banana	$88.8(2.95)^{5}$	$92.4(2.17)^{5}$	$91.6(1.97)^{5}$	$92.4(0.99)^{5,1}$	$91.1(2.70)^{5,1}$	$90.1(5.16)^{5}$	$92.0(1.19)^{5,20}$
cancer	$96.1(0.63)^{6}$	$96.6(0.64)^{2}$	$96.5(0.58)^{2}$	$96.4(0.69)^{4,1}$	$95.8(0.96)^{6,1}$	$96.5(0.48)^{4}$	$96.4(0.84)^{2,10}$
dermatology	$\underline{97.1}(0.57)^{2}$	$94.8(1.96)^{2}$	$94.2(1.89)^{4}$	$\underline{96.2}(1.14)^{2,1}$	$95.5(1.72)^{4,3}$	$\underline{96.6}(1.10)^{6}$	$\underline{96.3}(1.20)^{2,10}$
glass	$65.3(3.12)^{4}$	$63.5(5.15)^{6}$	$62.4(4.22)^{2}$	$63.4(2.90)^{4,1}$	$64.3(3.67)^{6,1}$	$65.2(4.53)^{2}$	$64.1(4.06)^{6,10}$
heart	$80.2(3.80)^{2}$	$80.8(3.37)^{2}$	$79.6(3.15)^{2}$	$79.8(2.47)^{4,1}$	$79.0(3.37)^{2,1}$	$80.0(2.71)^{2}$	$81.1(2.07)^{4,10}$
ionosphere	$\underline{87.7}(1.60)^{2}$	$\underline{87.8}(2.08)^{4}$	$\underline{83.2}(5.11)^{6}$	$87.9(2.07)^{6,1}$	$86.3(1.57)^{6,1}$	$\underline{89.5}(1.90)^{4}$	$\underline{87.9}(2.77)^{4,10}$
iris	$94.9(2.24)^{2}$	$94.8(2.03)^{2}$	$93.3(4.07)^{2}$	$92.1(3.99)^{3,1}$	$93.5(3.13)^{4,1}$	$93.3(4.70)^{2}$	$94.5(1.32)^{2,20}$
letter	$85.9(0.85)^{10}$	$85.3(2.29)^{10}$	$89.0(1.27)^{10}$	$88.2(1.33)^{10,5}$	$87.5(0.99)^{10,5}$	$90.0(0.64)^{10}$	$\underline{90.8}(0.55)^{10,20}$
NIST	95.1^{10}	95.4^{10}	94.2^{10}	$\underline{96.2}^{20,10}$	$95.7^{10,10}$	$\underline{96.2}^{20}$	$\underline{96.0}^{10,10}$
optical	$\underline{95.5}(0.86)^{2}$	$95.0(0.84)^{2}$	$95.8(0.48)^{2}$	$96.0(0.66)^{4,5}$	$\underline{95.9}(0.41)^{2,1}$	$\underline{96.8}(0.37)^{4}$	$\underline{96.4}(0.31)^{4,10}$
pen	$97.8(0.45)^{10}$	$97.6(0.39)^{2}$	$97.8(0.24)^{2}$	$97.7(0.82)^{4,3}$	$97.8(0.21)^{2,1}$	$\underline{98.9}(0.20)^{10}$	$\underline{98.9}(0.18)^{10,20}$
pima	$74.9(2.18)^{2}$	$74.7(1.67)^{2}$	$74.0(1.86)^{6}$	$74.7(0.96)^{2,3}$	$74.5(1.78)^{2,5}$	$75.6(1.46)^{2}$	$75.2(1.42)^{6,10}$
satimage	82.4^{2}	85.2^{2}	85.2^{2}	$86.6^{6,3}$	$85.8^{6,1}$	$\underline{88.4}^{6}$	$\underline{87.6}^{4,20}$
segmentation	$93.8(1.11)^{4}$	$93.4(0.91)^{4}$	$93.6(1.49)^{4}$	$94.0(0.94)^{4,1}$	$94.0(0.77)^{6,1}$	$\underline{95.2}(0.83)^{6}$	$\underline{95.6}(0.60)^{6,20}$
sonar	$\underline{78.3}(2.65)^{4}$	$\underline{79.4}(2.90)^{2}$	$67.7(4.69)^{6}$	$\underline{78.2}(4.46)^{2,1}$	$\underline{79.8}(3.00)^{2,5}$	$\underline{79.0}(4.45)^{2}$	$\underline{79.9}(3.75)^{4,20}$
soybean	$\underline{89.9}(2.70)^{2}$	$\underline{89.9}(2.90)^{2}$	$89.3(1.75)^{2}$	$\underline{89.2}(3.22)^{2,5}$	$\underline{90.3}(1.17)^{2,4}$	$\underline{90.1}(1.54)^{2}$	$\underline{91.1}(1.13)^{2,10}$
vowel	$82.0(1.95)^{11}$	$82.8(2.22)^{11}$	$81.0(3.35)^{11}$	$82.4(3.56)^{11,1}$	$82.8(2.77)^{11,5}$	$81.5(2.61)^{11}$	$81.3(2.71)^{11,10}$
waveform	$83.1(1.72)^{2}$	$82.6(1.45)^{2}$	$82(2.00)^{2}$	$82.6(1.45)^{2,3}$	$82.5(1.52)^{2,3}$	$82.0(1.19)^{2}$	$83.5(1.77)^{5,20}$
waveform-noise	$77.8(2.09)^{2}$	$78.5(1.59)^{2}$	$78.7(2.04)^{2}$	$79.9(1.91)^{2,1}$	$78.6(1.98)^{3,5}$	$\underline{81.7}(2.04)^{5}$	$\underline{82.6}(2.00)^{3,20}$
wins	5	3	1	4	3	10	11

Table 4.6: Results of the experiments with a mixture of experts and different gating networks. Scores are in percentage of correct classification on the test set and are the average over 10 experiments in a 5x2cv F test framework or a single run for NIST and satimage. The standard deviation is given between parentheses. Each of the scores corresponds to the best model out of the particular class of models as selected on validation data. The first superscript indicates the number of experts in the best model. For mixtures of latent variable models, the second superscript specifies the dimension of latent space. The second superscript for the MLP gate gives the number of hidden neurons. The underlined scores are the ones that do not pass the 5x2cv F test or McNemar's test with 90% confidence when compared with the model having the best score.

one order of magnitude. This illustrates that standard mixtures of experts are far less localized and attribute patterns across the experts, if this happens to reduce the total error. It seems to confirm the claim of Jordan and Jacobs (1994) that the soft splits of the standard mixture of experts reduce the variance of the model which might explain the better results obtained using NN gates.

The fact that localized gates may lead to hard splits of the data space can also be seen informally from the definition of the posteriors $\pi_j(\mathbf{x}, \mathbf{t})$ (4.6) and the localized gate (4.2):

$$\pi_j(\mathbf{x}^n, \mathbf{t}^n) = \frac{\alpha_j p_j(\mathbf{x}^n)\phi_j(\mathbf{t}^n|\mathbf{x}^n)}{\sum_{i=1}^{m} \alpha_i p_i(\mathbf{x}^n)\phi_i(\mathbf{t}^n|\mathbf{x}^n)} = \frac{g_j(\mathbf{x}^n)\phi_j(\mathbf{t}^n|\mathbf{x}^n)}{\sum_{i=1}^{m} g_i(\mathbf{x}^n)\phi_i(\mathbf{t}^n|\mathbf{x}^n)}.$$

If the initial values of $g_j(\mathbf{x})$ after unsupervised EM training of the mixture model are very close to one for one output and hence close to zero for the others, it is likely that the posteriors will be close to values of the gating outputs. Moreover, as we know from (3.18), during training the gating outputs get pulled towards the posteriors, which in this case means that the initial hard split will almost not change. This is what I observed in some experiments: the centers of the Gaussian components almost do not move during the EM training of the entire localized ME.

It also suggests possible ways to improve the results with localized MEs. One way might be to initialize the mixture models for the gate in such a way that there is some overlap between the component densities. Some initial tests show that this does indeed sometimes improve accuracy but not enough to cover the gap with standard MEs. Another possibility is to generalize the E-step of the EM algorithm by introducing a parameter β:

$$\pi_j(\mathbf{x}^n, \mathbf{t}^n) = \frac{[\alpha_j p_j(\mathbf{x}^n)\phi_j(\mathbf{t}^n|\mathbf{x}^n)]^\beta}{\sum_{i=1}^{m}[\alpha_i p_i(\mathbf{x}^n)\phi_i(\mathbf{t}^n|\mathbf{x}^n)]^\beta}.$$

Thus, for $\beta = 1$ standard EM is obtained and decreasing β smoothens the distribution of the posteriors which becomes uniform for $\beta = 0$. We could now start by training the localized ME in the standard manner and gradually decrease β to force the gating network to decompose the data space in a smoother way; it can be interpreted as inverse annealing. This has been proposed by Hoffman (1999) in a different context but with a similar goal. In (Hoffman 1999) this method gave results comparable to carefully annealed models and reduced overfitting of models trained with standard EM.

Let us conclude with a rough comparison of the results obtained with mixture models in Bayes classifiers (Table 4.1) and those with MEs (Table 4.6). If we take the highest scores on each data set, Bayes classifiers are best in 13 out of 19 cases! The difference in performance can be considerable: with a full GMM 95% of the test data is correctly classified on letter while the best ME cannot do better than 90.8%. We will perform a more a direct comparison on two machine vision problems in Chapter 6.

4.5 Discussion

I also tried to take advantage of having a localized ME with a MPCA gate to perform localized dimensionality reduction for each of the experts. This should be opposed to performing a global PCA on the data and using the same reduced data set for all experts. In the localized scheme, each expert can be assigned to a local PCA model (a mixture component out of the MPCA gate) and this local PCA can be used to map the input data for the corresponding expert into a lower dimension (Tipping and Bishop 1999). This idea is clearly in agreement with the assumption that different features might be important in different regions of the data space. Local dimensionality reduction was implemented by first training the localized gate in an unsupervised fashion and then constructing the reduced dimension data set for each expert by projecting the data points onto the local linear model corresponding to the expert. This led to much better results than with the global approach. Note that in both cases, the gating network uses the original data and the experts the reduced data. However, both models were clearly outperformed by a simple global PCA for standard MEs.

An alternative approach for the combination of local experts using a decomposition through density estimation has been proposed by Rida et al. (1999). It differs in several aspects from the localized ME approach pursued in this chapter. First of all, they completely separate the training of the gate and the experts. The gate is a GMM which is initialized in a constructive way using a variant of the Kohonen map that tries to find an appropriate number of clusters automatically. Patterns are then assigned to at least one expert using a threshold on the likelihood of a pattern for each mixture component. Each of the experts is then trained on the (localized) subset of the data assigned to it. This decoupling has the advantage that any standard classifier can be used as expert and not only the ones which have a probabilistic interpretation as in MEs. This allows, for example, to use support vector machines as experts in a straightforward way. Combination of the trained experts is done as in this chapter with a normalized gating output possibly with additional coefficients adapted on validation data. The initial experiments in (Rida et al. 1999) look promising and their combination of experts outperforms global models on two data sets.

Recently, an on-line EM algorithm for a localized ME with a full GMM gate has been proposed (Sato and Ishii 2000) for regression problems. The idea is based on similar observations as the ones made in section 2.8.4 regarding the on-line EM algorithm. When dealing with GLM experts with a linear activation function, the M-step reduces to a weighted least-squares problem which depends entirely on sufficient statistics (3.30). As we saw in section 4.2, the M-step for a GMM gate is almost identical to the unsupervised case and is also fully specified in terms of sufficient statistics (section 2.8.4). This means that the general on-line EM algorithm (Neal and Hinton 1999) can be applied in this case as well. Sato and Ishii (2000) go one step further and develop a stochastic scheme which can handle real on-line learning with data varying over time. This is done by including an exponentially decaying average of the statistics of recently visited data points. They obtain good results on some low-dimensional function approximation problems in a dynamic environment. Real on-line learning could in principle also be used in classification problems with non-linear experts when making the approximations described in section 3.4.2. This would again lead to a weighted least-squares problem for each expert summarized by its sufficient statistics.

CHAPTER 5

Combining Boosted Experts Dynamically

This chapter stands a little apart from the rest of the thesis. While adhering to the principle of making a complex model out of simpler ones, this time we do not cast the problem in probabilistic terms. An extension of Freund and Schapire's (1997) AdaBoost algorithm is presented which allows for an input-dependent combination of the base experts. A separate model is used for determining the input-dependent coefficients of each expert. The error function minimized by these additional models is a margin cost function that has also been shown to be minimized by AdaBoost. The models used for dynamically combining the base experts are simple one-layer neural networks (Moerland and Mayoraz 1999).

Roadmap

Ensemble methods such as bagging (Breiman 1996) and boosting (Freund and Schapire 1997) operate by taking a base algorithm and invoking it repeatedly with different training sets. A main characteristic of boosting is that it maintains a distribution on the original training set and sequentially adapts this distribution to emphasize patterns which have been frequently misclassified. Boosting algorithms, such as AdaBoost (Freund and Schapire 1997) and its variants, have shown very good performance on a wide range of classification problems and with many different base algorithms (also called weak learners) (Freund and Schapire 1996; Quinlan 1996; Schwenk and Bengio 1998; Dietterich 1998b; Bauer and Kohavi 1999).

The combination of the different experts generated by a boosting algorithm is often based on some form of weighted voting. In AdaBoost, for example, the higher weights are given to the experts with the lower error. An alternative to using fixed weights when combining the base experts, is to allow the weights to be input-dependent. This idea can be traced back to Quinlan (1996) who proposed to define a confidence measure for an input pattern when using C4.5 as base expert. This heuristic led to an improvement on a large majority of the data sets evaluated by Quinlan (1996). A related approach is described in (Avnimelech and Intrator 1999) where MLPs are dynamically combined using a confidence measure based on the difference between the highest and the second highest output of a MLP on a multi-class problem. Waterhouse and Cook (1997) used boosting to initialize a mixture of MLP experts and then retrained these initial experts and a randomized gate in the mixture of experts framework. The gating network, thus, provides a dynamic combination of the base experts. On a large speech recognition problem, Waterhouse observed that only when both experts and the gate were retrained, performance improved with respect to the boosted model used for initializing the experts.

In this chapter, a more principled approach for dynamically combining boosted experts is proposed (called *Dynaboost*). The approach is based on the recent idea that various boosting algorithms can be

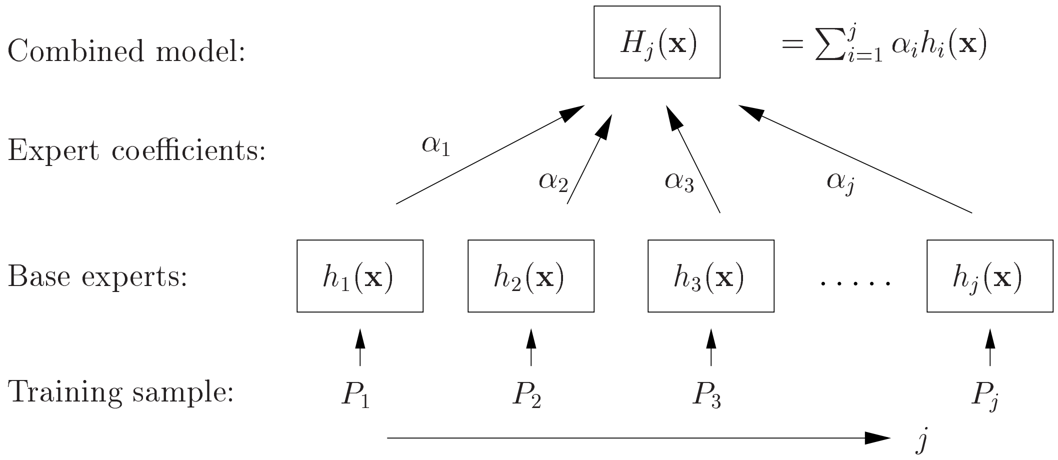


Figure 5.1: AdaBoost: iterative generation of an ensemble $\{h_i\}$ each member of which is trained on data sampled from the original training data according to a distribution P_i.

interpreted as a stage-wise gradient descent optimization of a cost function of the *margins* (Breiman 1997; Friedman et al. 1998; Rätsch et al. 1998; Mason et al. 1999). The margin concept plays an important role in explaining the resistance to overfitting of boosting algorithms: larger margins imply lower generalization error (Schapire et al. 1998). They also show that AdaBoost tends to increase the margins of the training examples.

The DynaBoost algorithm extends other boosting algorithms by using a separate model for determining the input-dependent coefficients of each expert. This means that the combined DynaBoost model resembles a mixture of experts with a gating network dynamically combining the base experts. The error function minimized by the models which make up the gating network is exactly the cost function of the margins that has been shown to be minimized by AdaBoost. In section 5.1, we recall the definition of AdaBoost and describe how to extend it to DynaBoost. The error function of the margins is described in more detail.

The experimental results for the DynaBoost algorithm and a comparison with AdaBoost on a range of binary and multi-class classification problems are reported in section 5.2. In all experiments discussed here, the models used for dynamically combining the base experts are simple GLMs. It is shown that the dynamic approach of DynaBoost significantly improves the results on most data sets when rather weak base experts are used (GLMs in this case). With MLP base experts the difference in performance between DynaBoost and AdaBoost is small.

5.1 AdaBoost and DynaBoost

The basic idea behind boosting is illustrated in Figure 5.1. An ensemble of base experts is constructed in a greedy way adding classifiers one by one to the ensemble. It takes as input a training set of N examples $D = \{(\mathbf{x}^1, t^1), \ldots, (\mathbf{x}^N, t^N)\}$ where $\mathbf{x}^i$ is drawn from the input space X and $t^i \in Y$ is the class label associated with $\mathbf{x}^i$. AdaBoost starts with a uniform weighting P_1 of the training data and adapts the weighting such that the misclassified examples get more weight. On round i, a base expert $h_i : X \to Y$ is trained to minimize the error on the training data weighted according to the distribution P_i. The expert coefficients α_i are chosen such that greater weight is given to the expert with lower weighted error. The combined model is a weighted vote over the base experts. The pseudo-code for AdaBoost is given in the top half of Algorithm 9.

Algorithm 9 AdaBoost.M1 for C-class problems (Freund and Schapire 1997) and DynaBoost

Require:

- A training set D of N examples: $\{(\mathbf{x}^1, t^1), \ldots, (\mathbf{x}^N, t^N)\}$ with labels $t^i \in Y = \{1, \ldots, C\}$.
- A weak learning algorithm $L(D, P)$ which returns a base expert which minimizes the weighted (on a distribution P) error on the training set D.

Ensure: Combined model $H_j(\mathbf{x}) = \text{argmax}_{t \in Y} \sum_{i \le j : h_i(\mathbf{x}) = t} \alpha_i$ (AdaBoost.M1)

$P_1(n) := 1/N$ for all $n = 1, \ldots, N$
for $j := 1$ to T **do**
  $h_j := L(D, P)$
  $\varepsilon_j := \sum_{n : h_j(\mathbf{x}^n) \neq t^n} P_j(n)$ $\{\varepsilon_j$ is the weighted error of $h_j\}$
  if $\varepsilon_j \ge 1/2$ **then**
    return H_{j-1}
  end if
  $\alpha_j := \frac{1}{2} \ln(\frac{1-\varepsilon_j}{\varepsilon_j})$ $\{\alpha_j > 0\}$
  $Z_j := 2\sqrt{\varepsilon_j(1-\varepsilon_j)}$ $\{Z_j$ is a normalization constant$\}$
  for all $n = 1, \ldots, N$ **do**

$$P_{j+1}(n) := P_j(n). \begin{cases} e^{-\alpha_j}/Z_j & \text{if } h_j(\mathbf{x}^n) = t^n \\ e^{\alpha_j}/Z_j & \text{if } h_j(\mathbf{x}^n) \neq t^n \end{cases} \tag{5.1}$$

  end for
end for

Ensure: Combined model $H_j^{\text{Dyna}}(\mathbf{x}) = \text{argmax}_{t \in Y} \sum_{i \le j : h_i(\mathbf{x}) = t} \alpha_i(\mathbf{x})$ (DynaBoost)

$P_1(n) := 1/N$ for all $n = 1, \ldots, N$
for $j := 1$ to T **do**
  $h_j := L(D, P)$
  $\varepsilon_j := \sum_{n : h_j(\mathbf{x}^n) \neq t^n} P_j(n)$ $\{\varepsilon_j$ is the weighted error of $h_j\}$
  if $\varepsilon_j \ge 1/2$ **then**
    return H_{j-1}^{Dyna}
  end if
  Determine the parameters for the jth gater $\boldsymbol{\theta}_j$ which minimize (5.4): $C_j^{\text{Dyna}} = \sum_{n=1}^{N} C_{j,n}^{\text{Dyna}}$
  for all $n = 1, \ldots, N$ **do**
    $P_{j+1}(n) := C_{j,n}^{\text{Dyna}} / C_j^{\text{Dyna}}$
  end for
end for

One of the nice properties of AdaBoost is that if the base experts have weighted error ε_j smaller than 1/2, the training error of the combined model goes to zero exponentially fast (Freund and Schapire 1997). While this theorem only addresses the training error, there is also strong experimental evidence that AdaBoost is quite robust to overfitting and generalizes well (Freund and Schapire 1996; Quinlan 1996). Recent explanations of this phenomenon are based on the notion of *margins*. The margin of an example is defined as the difference between the total weight assigned to the correct label and the one assigned to a wrong label:[1]

$$m(H_j, \mathbf{x}, t) = \left(\sum_{i \le j : h_i(\mathbf{x}) = t} \alpha_i \right) - \left(\sum_{i \le j : h_i(\mathbf{x}) \neq t} \alpha_i \right). \tag{5.2}$$

In the two-class case, when encoding the two class labels as -1 and 1 respectively, this reduces to the compact: $m(H_j, \mathbf{x}, t) = t \sum_{i=1}^{j} \alpha_i h_i(\mathbf{x}) = t H_j(\mathbf{x})$. A pattern has a positive margin if and only if it is correctly classified by the combined model. Moreover, the magnitude of the margin can be interpreted as a measure of confidence in the prediction.

Schapire et al. (1998) have demonstrated that larger margins imply lower generalization error and that AdaBoost tends to increase the margins of the training examples. This is intuitively clear when comparing the update of P (5.1) with the definition of the margin (5.2): most weight is placed on the examples for which the margin is smallest.

Another recent break-trough in the understanding of boosting algorithms, has been the proof that various boosting algorithms can be interpreted as a stage-wise gradient descent optimization of cost functions of the margins (Breiman 1997; Friedman et al. 1998; Rätsch et al. 1998; Mason et al. 1999). The margin cost function for AdaBoost.M1 at round j is:

$$C_j = \sum_{n=1}^{N} e^{-m(H_j, \mathbf{x}^n, t^n)} \tag{5.3}$$

and it can be shown that the updates of the distribution P_j and the expert coefficients α_j, and the stopping criteria on ε_j, all follow from a gradient descent optimization of the above margin cost function (see, for example, (Mason et al. 1999)). The cost function is a (rather loose) upper bound of the 0-1 loss function.

Now, we are ready to replace the fixed expert coefficients α_j of AdaBoost by input-dependent coefficients $\alpha_j(\mathbf{x})$ and present the DynaBoost algorithm. The combined model becomes:

$$H_j^{\text{Dyna}}(\mathbf{x}) = \underset{t \in Y}{\operatorname{argmax}} \sum_{i \le j : h_i(\mathbf{x}) = t} \alpha_i(\mathbf{x}).$$

The main problem is how to determine $\alpha_j(\mathbf{x})$. Adhering to the greedy approach of AdaBoost, it seems logical to use separate weak learners (which I coin *gaters*) for determining the input-dependent coefficients: $\alpha_j(\mathbf{x}) = \alpha_j(\mathbf{x}, \boldsymbol{\theta}_j)$, with parameters $\boldsymbol{\theta}_j$. The error function which has to be minimized for these new weak learners is readily obtained by generalizing (5.2) and (5.3):

$$C_j^{\text{Dyna}} = \sum_{n=1}^{N} e^{\left(\sum_{i \le j : h_i(\mathbf{x}^n) \neq t^n} \alpha_i(\mathbf{x}^n, \boldsymbol{\theta}_i)\right) - \left(\sum_{i \le j : h_i(\mathbf{x}^n) = t^n} \alpha_i(\mathbf{x}^n, \boldsymbol{\theta}_i)\right)} = \sum_{n=1}^{N} C_{j,n}^{\text{Dyna}}, \tag{5.4}$$

with $C_{j,n}^{\text{Dyna}}$ as short-hand for the error of the combined model at iteration j on pattern $\mathbf{x}^n$. The error function which has to be minimized on round $j+1$ can then be written as the sum of the old $C_{j,n}^{\text{Dyna}}$

[1] This is actually a lower bound on the definition of the margin in (Schapire et al. 1998).

multiplied by an exponential factor:

$$C_{j+1}^{\text{Dyna}} = \sum_{n=1}^{N} C_{j,n}^{\text{Dyna}} e^{-\alpha_{j+1}(\mathbf{x}^n, \boldsymbol{\theta}_i) I(h_{j+1}(\mathbf{x}^n) = t^n)}, \tag{5.5}$$

where $I(\text{true}) = 1$ and $I(\text{false}) = -1$. If we choose the $\alpha_{j+1}(\mathbf{x}) = \alpha_{j+1}(\mathbf{x}, \boldsymbol{\theta}_{j+1})$ to be differentiable in its parameters $\boldsymbol{\theta}_{j+1}$, gradient-based optimization algorithms can be used for minimizing (5.5).

In the experiments described in the next section, a GLM with a sigmoid activation function was chosen for each gater:

$$\alpha_j(\mathbf{x}, \boldsymbol{\theta}_j) = \sigma(\boldsymbol{\theta}_j \cdot x) = 1/(1 + \exp(-\boldsymbol{\theta}_j \cdot x)).$$

The choice of the activation function is motivated by the fact that the fixed expert coefficients α_j in AdaBoost are strictly positive. The lower asymptote of the sigmoid function guarantees that it is also the case for the $\alpha_j(\mathbf{x})$ in DynaBoost. The higher asymptote of the sigmoid function has no counter-part in AdaBoost: α_j as defined in Algorithm 9 can grow arbitrarily large when $\varepsilon_j \to 0$. The higher asymptote may nevertheless be useful since allowing large values for $\alpha_j(\mathbf{x})$ could correspond to overly confident predictions and a certain risk of overfitting. The main motivation for using a GLM is that is an easy to learn rather weak learner which can be trained with gradient-based methods. Another reason is that we are also interested in links between DynaBoost and mixtures of experts: the combined DynaBoost model resembles a mixture of experts with a gating network dynamically combining the base experts. The combined model obtained by DynaBoost could, for example, be retrained in the framework of mixtures of experts as in (Waterhouse and Cook 1997).

The pseudo-code for DynaBoost is given in the bottom half of Algorithm 9. DynaBoost algorithm is similar to AdaBoost but with all α_j replaced by $\alpha_j(\mathbf{x})$, and the update of $\alpha_j(\mathbf{x})$ consists of a gradient-based optimization of a margin cost function (5.5). This cost function is minimized by putting more weight on correct predictions and $C_{j,n}^{\text{Dyna}}$ plays a role similar to $P_j(n)$: most weight is placed on the difficult examples.

5.2 Experiments

The experiments were performed on 10 of the data sets listed in Table 2.1. On all benchmarks, 5x2cv was used and the results presented are the average over the 10 outcomes.

We compare AdaBoost and DynaBoost using two different base experts: GLMs and MLPs with 10 hidden neurons. In all cases, we performed boosting by sampling where examples are drawn with replacement from the training set with a probability proportional to the distribution P. The output activation function for the base experts was either a sigmoid function for two-class problems or a softmax for multi-class problems. The error function minimized by the base experts was cross-entropy. The base experts and the GLM gaters (for DynaBoost) were trained for 30 iterations of the scaled conjugate gradient algorithm (Møller 1993). The number of rounds of boosting varied with the data sets and was chosen to be 100, 200, or 500. The reported experiments are the first executions of DynaBoost and more have to be done in order to allow more robust conclusions on the performance of the method. Nevertheless, some general comments can be safely made on the basis of these experiments.

Figure 5.2 clearly illustrates that the behavior of DynaBoost is quite similar to the one of AdaBoost when strong learners are used as experts (e.g. MLPs, see the 10 plots in the right-hand half of the figure). On the contrary, important differences are observed between the two methods when weak learners are used.

The training error for DynaBoost always converged to zero, usually much faster than the one for AdaBoost (not shown). This is not surprising since the use of dynamic coefficients considerably increases the power of the learning model. The drawback of a powerful model is the increased risk of overfitting. Interestingly enough, none of the ten experiments with MLP base experts show any overfitting behavior.

The analysis is more complex in the case of weak learners. An impressive improvement is obtained with the use of dynamic coefficients in four cases, without any overfitting: banana, vowel, glass, and segmentation. In two other cases, there are no significant differences between DynaBoost and AdaBoost (sonar and cancer). For soybean, ionosphere and optical, the initial behavior of DynaBoost is good but after some rounds the model starts to overfit. In the first two cases, the error of DynaBoost after a few rounds is smaller than the smallest one achieved by AdaBoost and after a large number of rounds the two errors are comparable. This means, in particular, that the use of a validation set to stop the boosting process would have given a small and well performing learning model with dynamic coefficients. Finally, in the case of pima, which is known to have a high level of noise, DynaBoost overfits dramatically, while AdaBoost overfits only slightly.

5.3 Discussion

The DynaBoost algorithm seems an interesting extension of Freund and Schapire's (1997) AdaBoost and can improve upon it when the base experts are relatively weak. The use of dynamic coefficients considerably increases the power of the learning model and the training error for DynaBoost usually converges to zero much faster than for AdaBoost. The drawback of a powerful model is the increased risk of overfitting, but the experimental results indicate that in many cases DynaBoost does not overfit.

Directions for future research include the investigation of the convergence properties of DynaBoost. Can we prove that the training error of the combined model goes to zero as is the case for AdaBoost? Is it possible say something about the generalization error of DynaBoost? On the experimental side, more simulations have to be done in order to allow more robust conclusions on the performance of the method. This can include the use of other base experts, such as stumps and general decision trees. It would also be interesting to pursue other choices for the gaters. Finally, the combined model obtained by DynaBoost could be retrained in the framework of mixtures of experts to improve performance further.

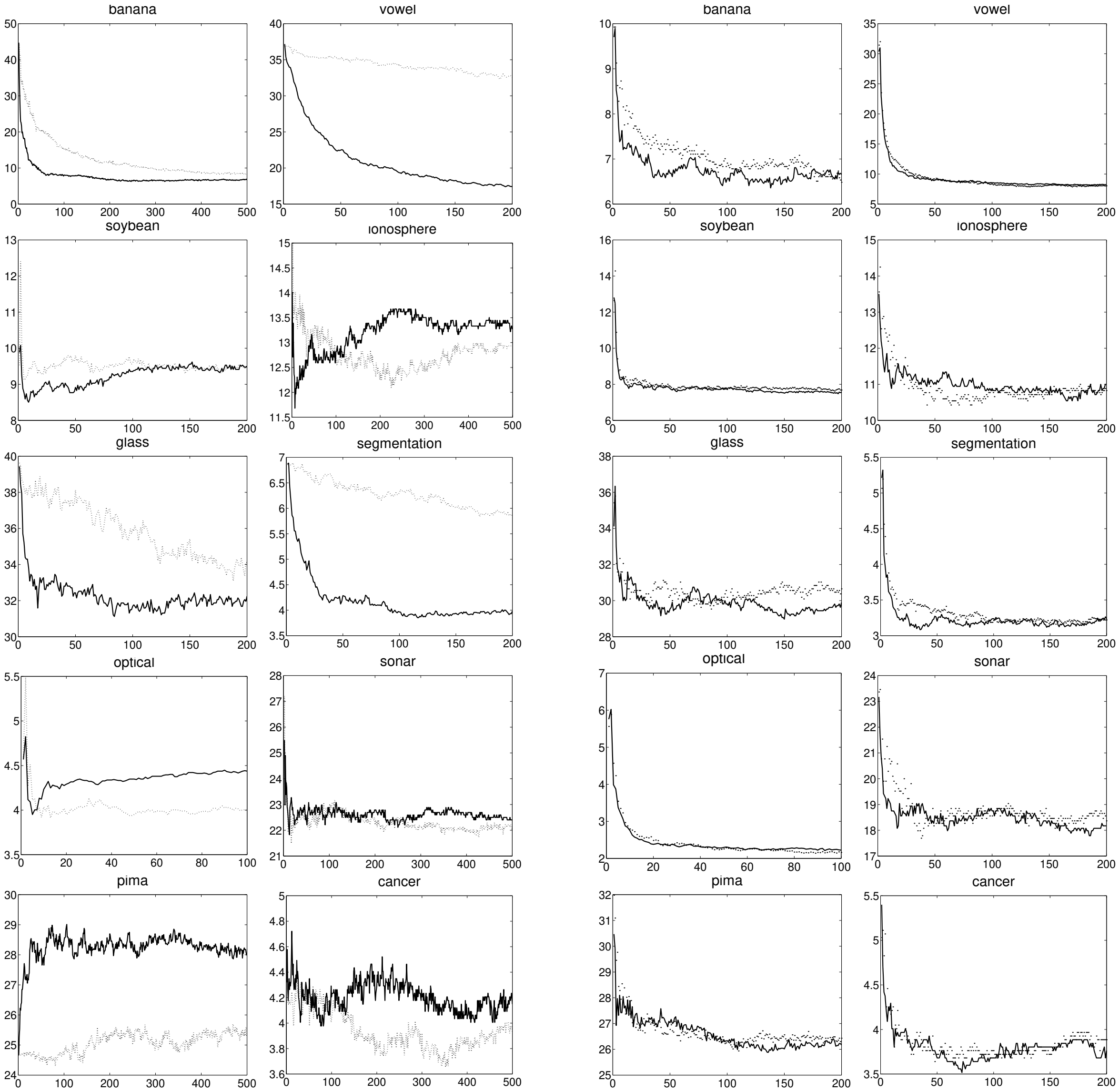


Figure 5.2: Comparison of AdaBoost (dotted) and DynaBoost (solid) on 10 classification problems: test errors (in percentage of misclassified examples) are displayed as a function of the number of rounds of boosting. The left-hand half gives the results with Glm base experts and the right-hand half with Mlp base experts.

CHAPTER 6

Applications to Computer Vision

Roadmap

Some of the models presented in the previous chapters are applied to two classification problems in the area of computer vision. The first data set is quite small and high-dimensional and consists of grey level images of faces and non-faces. The second benchmark is the popular MNIST data set of handwritten digits. I limited myself to Bayes classifiers using mixture models for the class-conditional densities and standard MEs. The results with a ME are slightly better than the ones obtained with MLPs but they both are clearly outperformed by a Bayes classifier using mixtures of latent variable models. Without any further pre-processing the latter come quite close to the best results on the MNIST data set.

6.1 Face Versus Non-Face Classification

In this section, we construct classifiers which have to distinguish faces from non-faces. The resulting classifiers could, for example, be used in a face detection system in which the appearance of a face in a larger image has to be detected (Ben-Yacoub 1997). Possible applications are in face recognition or identification for access-critical services such as information about one's bank account. Note that the goal was not to build a state-of-the-art system but mainly to evaluate some of the models discussed in the previous chapters on a small but high-dimensional data set.

For this purpose, we used a small database of face and non-face images which is described in detail in Appendix D. The database consists of 25×25 grey level images of 1476 faces and 1628 non-faces and was split in a training set of 200 faces and 200 non-faces, a validation set of 538 faces and 614 non-faces, and a test set of 738 faces and 814 non-faces. Each of the images is scaled in the interval $[0, 1]$ and no other pre-processing steps were made (such as normalization to compensate for illumination changes).

The classifiers constructed are Bayes classifiers using mixture models and mixtures of experts. The set-up of the experiments was similar to the scheme described in section 4.3.1 for Bayes classifiers and in section 4.4.1 for mixtures of experts. Both GMMs and mixtures of latent variable models were used for modeling each of the two classes in a Bayes classifier. Mixtures models were trained varying the number of mixture components $m \in \{1, 2, 3, 5, 7, 10, 15, 20, 30, 50, 100\}$ and the dimension of latent space for MPCAs and MFAs $\ell \in \{1, 2, 3, 5, 10, 15, 20, 30, 50, 75\}$.[1] Mixtures of experts with GLM

[1] When increasing the number of mixture components, the maximum dimension of latent space was gradually decreased to avoid singularities.

Gmm					Me	
spherical	diagonal	full	Mpca	Mfa	Glm	Mlp
91.2^{3}	90.1^{100}	92.8^{1}	$94.4^{10,20}$	$\underline{95.4}^{1,75}$	93.1^{3}	$92.5^{10,25}$

Table 6.1: Results of the experiments with Bayes classifiers and mixtures of experts on the face data set. Scores are in percentage of correct classification on the test set. Each of the scores corresponds to the best model out of the particular class of models as evaluated on a validation set. The underlined score is best with 90% confidence using McNemar's test. Superscripts are as in Table 4.1 (for Bayes classifiers) and Table 4.6 (for mixtures of experts).

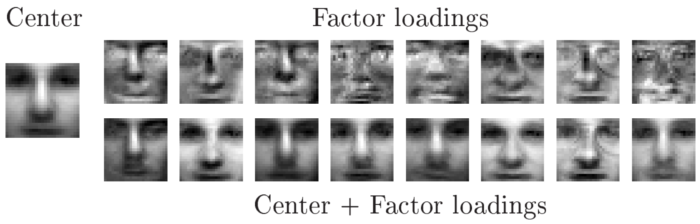


Figure 6.1: Mfa with 1 mixture component and 75 factors trained on 200 faces of the face data set. Out of the 75 factors, 10 typical examples have been selected. The sum of the center and these factors illustrates some of the transformations which have been captured.

experts were used varying the number of experts $m \in \{1, 2, 3, 4, 5, 7, 10, 15, 20\}$. With Mlp experts $m \in \{1, 2, 5, 10\}$ while the number of hidden units was chosen from $\{4, 8, 15, 25, 50\}$.

The results of the experiments of the classifiers which perform best on the validation set are given in Table 6.1. A simple factor analyzer with $\ell = 75$ outperforms all other models (McNemar's test with 90% confidence). Some characteristic aspects of the resulting model are visualized in Figure 6.1. The center $\boldsymbol{\mu}$ of the Fa represents the "average" face occurring in the training data. The factor loadings, that is, the columns of $\mathbf{W}$ in (2.18), can be interpreted as transformations of this average face to account for illumination changes and other facial characteristics. Figure 6.1 shows 10 of the 75 factor loadings and the sums of the center and these factor loadings. Some of the features extracted are a mustache in the first factor loading and glasses in the seventh factor loading. Eyes become in general more visible and one can also observe changes in nose and mouth shape.

The Fa and Ppca models are closely related to the well-known method of *eigenfaces* (Turk and Pentland 1991) which is just a catchy name for a standard Pca on the face images: each image is projected onto the principal eigenvectors and classified as a face if its reconstruction error falls below a certain threshold. As already observed in section 2.3, the advantage of probabilistic Pca with respect to standard Pca is that it is based on a proper probability model and incorporates the *in-subspace* error. Factor analysis has moreover the advantage of separately modeling the pixel noise and this might explain its good performance on this task. Good performance in a face recognition task has also been obtained with Mfas in (Frey, Colmenarez, and Huang 1998).

The best performing Mpca model has 10 mixture components, each with a dimension of latent space $\ell = 20$. The 10 centers of the Mpcas for the face images and the non-face images are shown in Figure 6.2. This clearly illustrates that the mixture components for the face images model different types of faces and variations in lighting. For the non-face images one observes both texture images and parts of a face (the ninth center, for example). The factor loadings of a mixture component can now be interpreted as transformations of each of these centers. A mixture model is better suited for

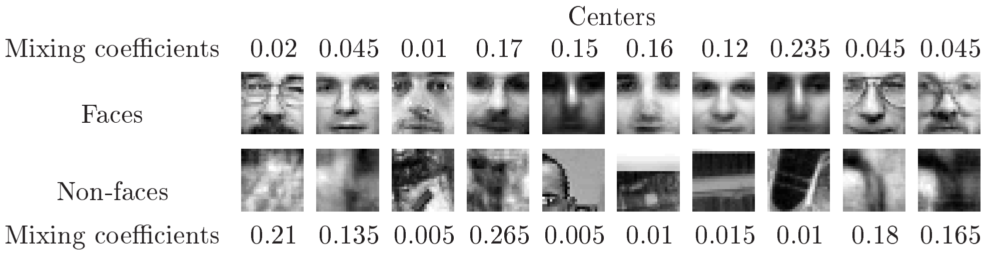


Figure 6.2: Two MPCAs with 10 mixture components and 20 factors trained on 200 faces and 200 non-faces of the face data set. The first row shows the 10 centers of the MPCA for the face examples and the second row of the MPCA for the non-face examples. The mixture coefficients α_i of each mixture component are also indicated.

capturing the non-linear manifold of "face space" than simple linear models PCA and FA. However, in this experiment it does not seem to pay off. A likely explanation is the limited size of the training set with only 200 images for both classes.

The Gaussian mixture models are clearly outperformed by the mixtures of latent variable models. As is to be expected when modeling the entire covariance matrix with so few data in 625-dimensional space, the best full GMM has only one mixture component. Spherical and diagonal GMMs are again not flexible enough to model the data.

The mixture of expert models are also outperformed by the best Bayes classifiers. However, they are still better than a simple GLM (90.5%) and big MLPs (91,3%) on the same data.

6.2 Handwritten Digit Recognition

From the small data set of the previous section, we now go to a large data set of handwritten digits which can be obtained from (LeCun 2000). This MNIST data set is described in detail in Appendix D. It comes as 28×28 grey level images in a training set of 60,000 examples and a test set of 10,000 examples. Each of the 8-bit pixel values is scaled in the interval $[0,1]$. The MNIST data set as it is available from (LeCun 2000) is already normalized and centered. However, it is still highly diverse in writing style, line thickness, slant, and arc size (see Figure D.3). This makes it a challenging test bed for learning algorithms. It also has the advantage of being widely used; this enables the comparison with many other classifiers.

The classifiers constructed are again Bayes classifiers using mixture models and mixtures of experts. The set-up of the experiments is similar to the scheme described in section 4.3.1 for Bayes classifiers and in section 4.4.1 for mixtures of experts. Both GMMs and mixtures of latent variable models were used for modeling each of the ten classes in a Bayes classifier. Mixtures models were trained varying the number of mixture components $m\in\{10,20,30,40,50,100,200\}$ and the dimension of latent space for MPCAs and MFAs $\ell\in\{10,20,30,40\}$.[2] Mixtures of experts were used varying the number of experts $m\in\{10,20,30,40,50\}$ and with both GLM and MLP experts. The number of hidden units in each MLP expert was chosen to be 50 or 150. Training of the ME models deviated slightly from the standard EM set-up. The total training set was divided into a set of 40,000 examples for estimating the parameters of a ME and a validation set of the remaining 20,000 examples for early stopping. Because of the size of the training set, it was divided into 20 bunches of 2,000 examples and the EM algorithm was applied on each of those bunches cyclicly. This means that we lose the convergence guarantee of the

[2] Not all possible combinations were tried though.

		GMM				ME	
m	ℓ	spherical	diagonal	MPCA	MFA	GLM	MLP
10	10	92.0	92.6	97.9	97.9	96.4	97.2^{150}
	20			98.2	98.0		
	30			98.3	98.3		
20	10	93.6	93.8	98.0	97.7	96.9	97.5^{50}
	20			98.3	98.3		
	30			98.3	98.3		
30	10	94.2	94.7	98.1	97.9	97.1	97.6^{50}
	30			$\underline{98.5}$	98.4		
40	20	94.4	94.9	$\underline{98.5}$	$\underline{98.4}$	97.2	
	40			$\underline{98.5}$	$\underline{98.5}$		
50	20	95.0	95.2	$\underline{98.4}$	$\underline{98.4}$	97.5	
100		95.8	95.9			97.4	
200		95.8	96.5				

Table 6.2: Results of the experiments with Bayes classifiers and mixtures of experts on the MNIST data set. Scores are in percentage of correct classification on the test set. The underlined scores are best with 90% confidence using McNemar's test. m indicates the number of mixture components or the number of experts. ℓ indicates the dimension of latent space for the mixtures of latent variable models. The number of hidden neurons in a mixture of experts with MLP experts is superscript.

EM algorithm proper, but in practice this simple strategy turns out to work quite well.

All classifiers have been trained only once and the corresponding results on the 10,000 example test set are given in Table 6.2. McNemar's test was used to determine whether the difference in performance between two classifiers was statistically significant. These results were obtained with subsampled 16×16 images in order to reduce the computational complexity.

We first have a look at the results obtained with the Bayes classifiers using a mixture model for each digit class. As one can see, the GMMs are again not flexible enough to model the data. Even when increasing the number of mixture components to 200, performance is still significantly worse than for the other models.

Mixtures of latent variable models show good performance on the MNIST data set, also when compared with previous results using other classifiers, as we will see later on in this section. This does not come as a surprise as already in (Hinton et al. 1997; Tipping and Bishop 1999) good results are obtained with MPCAs and MFAs on a smaller data set of handwritten digits.[3] Hinton et al. (1997) proposed the use of mixtures of latent variable models since digit images are supposed to lie on a non-linear, lower-dimensional, and smooth manifold. Using a mixture model takes into account the non-linearity of the manifold, with the mixture components capturing different writing styles and significant transformations. Each mixture component corresponds to a *linear* latent variable model and captures local invariances induced by small transformations.

The different writing styles and transformations captured by a MPCA are illustrated in Figure 6.3. Each digit was modeled by a mixture of 20 PPCAs with a dimension of latent space $\ell = 20$. Figure 6.3 shows the centers of each mixture model. Examples of different writing styles can be seen in the presence of twos with and without curl and of several sevens with an additional bar. Variations in

[3] Though (Hinton et al. 1997) used an approximation of the EM algorithm for these models in which examples are attributed in a soft way to the mixture components which reconstruct them well.

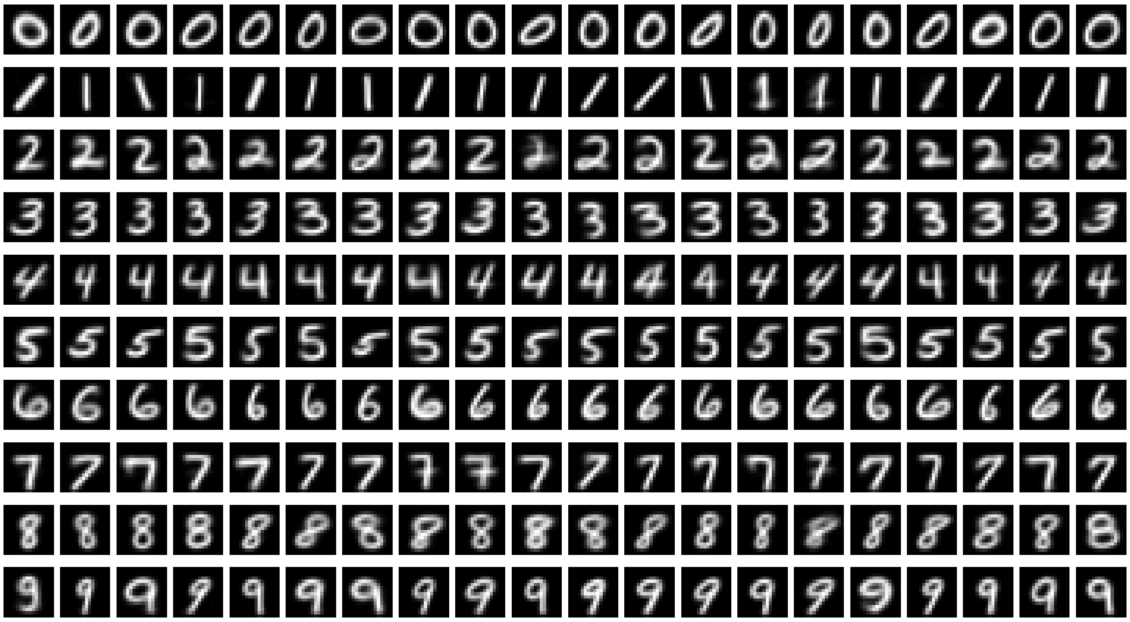


Figure 6.3: Centers of a Mpca with 20 mixture components (and 20 factors) for each digit of the MNIST data set.

line thickness have also been taken into account; see for example, the first and second center for 1. The centers incorporate the variations in slant of the digits as well.

A priori one might have expected Mfas to outperform Mpcas, given the more flexible diagonal noise model of factor analyzers. I can see two possible explanations for the fact that their performance is quite similar. Firstly, the normalization of the digit images during pre-processing might lessen the need of a diagonal noise model. Secondly, as in the previous chapters, the noise model in a Mfa was assumed to be tied across the mixture components. Maybe this is too strong a constraint and better results might be obtained by not tying the noise model. Similar performance with Mpcas and Mfas on a smaller handwritten digit data set was also observed in (Hinton et al. 1997).

Mixtures of experts perform considerably worse on the MNIST data when compared with mixtures of latent variable models (Table 6.2). Moreover, they take far more time to train than mixture models in Bayes classifiers. While the training time for the best Mpcas and Mfas is a couple of hours, one should at least count in terms of days for training a mixture of experts on the MNIST data; this is especially so when having Mlp experts.

The results with mixtures of experts are actually quite good when comparing with previous results for MNIST. A short summary of these is given in Figure 6.4 in which the best scores with some popular models on the same data set *without* further pre-processing are displayed. The best Mlp misclassifies 2.95% of the test examples and has two hidden layers with 500 and 150 hidden units respectively. This amounts to approximately 277,500 parameters (20×20 images were used). A mixture of 30 Glm experts gives a similar misclassification rate of 2.9% (Table 6.2) but using three times less parameters. Mixtures of Glm experts also compare favorably with pairwise linear classifiers which give 7.6% error. Pairwise coupling consists of training a separate linear model on each pair of classes and combining the predictions to make a classification. For handwritten digit recognition this leads to 45 linear classifiers, each of which is trained using about 1/5 of the data. Such a pairwise linear classifier is clearly outperformed by a mixture of 40-50 Glm experts. This is also the case when using a pairwise

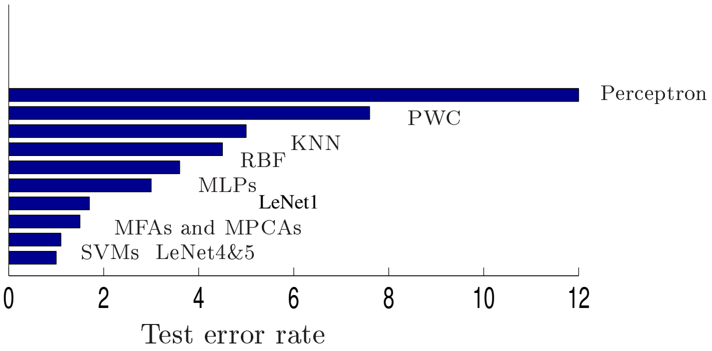


Figure 6.4: A summary of the best scores obtained on the MNIST data set in this chapter and as available in (LeCun 2000) and (LeCun et al. 1995). PWC stands for a pairwise coupling of linear classifiers and KNN is a K-nearest neighbor classifier.

coupling of linear support vector machines (5.4% error in Kreßel 1999). Of course, each expert of a mixture of experts is trained using all data[4] and computationally more demanding.

Bayes classifiers with mixtures of latent variable models are among the best classifiers on the MNIST data, although their performance is not yet state-of-the-art. The best off-the-shelf classifier is a support vector machine with a polynomial kernel (1% error). Other classifiers which perform better incorporate, in one way or another, prior knowledge of the problem of handwritten digit recognition. I expect that better results can also be obtained with mixtures of latent variable by some additional pre-processing. A simple approach would be to straighten up the slanted digits. This has proved fruitful for other classifiers. In fact, the best result with a MLP has been obtained in this way and reduced the misclassification rate to 1.6% (LeCun et al. 1995). A more ambitious step would be to incorporate transformation invariance directly into the latent variable model (Jojic and Frey 2000).

[4] The posteriors in the EM algorithm might actually have a similar effect when going to zero.

CHAPTER 7

Conclusions and Outlook

The principle of making complex models out of simpler ones has proved a fruitful idea throughout this thesis. One of the important advantages of such an approach is that it leads to learning algorithms which are analytically and computationally tractable. In fact, the algorithms for the simple models are often at the heart of the algorithm for the complex one. The EM algorithm for maximum likelihood estimation is a versatile tool which embodies exactly this idea in a probabilistic context. It has been shown at work both in unsupervised learning and supervised learning with mixture models. We can also use EM in models with continuous latent variables that form the hidden causes for the patterns. The kernel trick can also be interpreted as a way of making a complex (non-linear) model out of a simple one. It is possible to apply this trick to any algorithm which can be formulated solely in terms of dot products between the patterns and it does not require any other changes of the algorithm. Boosting is yet another incarnation of the "complex out of simple" idea. It can be used with any type of classifier as a simple model where the boosting algorithm mainly determines the specific training set for each of the base models.

These basic models and techniques are well-known in the machine learning community but one of the contributions of this thesis is that they are presented in a unified framework. We now have a closer look at the more technical contributions and possible directions for future research.

Mixtures of latent variable models A detailed derivation of the general EM algorithm was given, which was specialized to maximum likelihood estimation for GMMs and mixtures of latent variable models. The first original contribution of the present work is mainly of empirical nature. Mixtures of latent variable models were demonstrated to consistently outperform Gaussian mixture models on 18 real-world data sets when applied to the problem of density estimation. This holds both in terms of generalization performance and computational complexity. A disadvantage of mixtures of latent variable models is that one has to choose extra free parameters, viz. the dimensions of latent space. While this can be done using validation data (and in fact that is what I did in the experiments mentioned above), it would be far more attractive if the appropriate dimensions could be determined automatically. It was shown that a Bayesian procedure can be used for this purpose without much computational burden. Bayesian PCA (Bishop 1999a) was derived as a special case of Bayesian FA. A series of experiments on toy and real-world data illustrated that the method is capable of finding the appropriate dimensions of latent space during training. Results obtained are at least as good as with maximum likelihood while avoiding a discrete model search.

Mixture models were used as class-conditional densities in a Bayes classifier. Also in this case, mixtures of latent variable models perform better than GMMs and Bayesian inference again proved to be effective. On the MNIST data set results were obtained which are close to the best state-of-the-art classifiers. I do expect mixtures of latent variable models to become an alternative to GMMs in other contexts, for example as an output density in a Hidden Markov model (Saul and Rahim 1998).

The above mixture models are mixtures of Gaussians whether constrained or not. This implies that they are most suitable for modeling continuous variables. Real-world data, however, often contains binary and categorical attributes mixed with continuous ones. Discrete attributes can be handled by *non-linear* latent variable models known as latent trait models in the field of statistics (Bartholomew and Knott 1999). The non-linear models loose the analytical tractability of the Gaussian ones but a variational approximation can be used to give an efficient algorithm, as was shown recently for the binary case (Tipping 1999). It would be interesting to apply these techniques to mixed data and develop mixtures of non-linear latent variable models.

I also proposed exact on-line versions of the EM algorithm for a mixture of latent variable models. On-line EM for PCA was demonstrated to speed up the extraction of principal components on a few small data sets. This merits a more careful experimental evaluation and also the extension to mixture models has not been tried out yet. Moreover, stochastic versions of the EM algorithm are expected to lead to a further speed-up of the learning process.

Kernel PCA As an application of the incremental and on-line EM algorithms for PCA, I showed that they can make kernel PCA feasible for large data sets with more than 10,000 patterns. This allowed us to extract a high number of non-linear features which were subsequently used to train simple 1-layer neural networks. The results obtained on some of the data sets outperform all other models described in this thesis. EM for PCA showed rapid convergence in practice but it is an open question whether numerically superior techniques exist which also avoid storing the kernel matrix.

A disadvantage of EM for kernel PCA is the need to calculate the entire kernel matrix in an incremental way on every iteration. Experimental results indicate that non-linear features which are almost as good can often be found by applying kernel PCA to only a small random subset of the data. This is not a very systematic approach but a recent proposal (Smola, Mangasarian, and Schölkopf 1999) holds the promise of doing it in a well-founded greedy way. It might well be the way to go for kernel PCA and other kernel methods.

Mixtures of experts A review of training methods for MEs was presented which collects results scattered around the literature. As a corollary a weak consistency property of the ME error function was proved demonstrating that at its global minimum the ME outputs estimate posterior probabilities. The link to the chapter on mixture models in unsupervised learning was provided by using mixture models as a gating network. These localized mixtures of experts turn out to perform not as well as standard MEs with a GLM gate. This is due to the softer splits in data space induced by the latter. I did not pursue a systematic comparison of MEs with other non-linear models. However, on most of the data sets used throughout this thesis I obtained similar scores with MLPs. A ME model sometimes has an edge on MLPs in terms of training time and the number of parameters though, as was illustrated on the MNIST data set.

The situation can be different if we have an a priori belief that the data space can be decomposed in a meaningful way. MEs might be the model of choice in that case. Examples are time series with different regimes (Weigend, Mangeas, and Srivastava 1995) and modeling multi-modal data for regression (Bishop 1995, section 6.4).

Boosting The last part of this thesis described a non-probabilistic way of making complex models out of simple ones, viz. boosting. The AdaBoost algorithm was extended by allowing an input-dependent combination of the experts. It was shown that the dynamic approach often leads to better scores when weak GLM experts are used. This work is rather preliminary and can be extended in various ways. A first issue is that the choice of a logistic function in the gate is quite arbitrary and other choices (linear, for example) should be experimented with. The choice of a linear activation function would have the advantage of giving a stronger bound on the training error than AdaBoost, since it contains having fixed weights as a special case. Other choices for the experts, such as stumps and decision trees, and for the gate, such as a small RBF network, are also worth exploring.

APPENDIX A

Matrix and Probability Identities

Notation and Some Pointers

As in the rest of this thesis, lower-case bold letters $\mathbf{a}$, $\mathbf{b}$ etc., denote vectors and upper-case bold letters $\mathbf{X}$, $\mathbf{Y}$ etc., denote matrices. For a more detailed description of the notational conventions, see page xiii. In what follows, I assume that matrices are non-singular when taking the inverse. Matrices and vectors are also supposed to have the right size for the operations applied to them.

Most of these identities are a subset of the ones collected on a cheat sheet made available by Sam Roweis: `http://www.gatsby.ucl.ac.uk/~roweis/notes/{matrixid.ps.gz,gaussid.ps.gz}`. Other useful references are: (Marcus and Minc 1992; Golub and Van Loan 1996; Kreyszig 1999) for linear algebra and (Mardia, Kent, and Bibby 1979) for probability theory.

Trace Rotation

$$\mathrm{tr}(\mathbf{X}\mathbf{Y}\cdots\mathbf{Z}) = \mathrm{tr}(\mathbf{Z}\mathbf{X}\mathbf{Y}\cdots) = \cdots = \mathrm{tr}(\mathbf{Z}\cdots\mathbf{X}\mathbf{Y}) \tag{A.1}$$

Derivatives

$$\frac{\partial(\mathbf{a}^T\mathbf{b})}{\partial\mathbf{a}} = \frac{\partial(\mathbf{b}^T\mathbf{a})}{\partial\mathbf{a}} = \mathbf{b} \tag{A.2}$$

$$\frac{\partial(\mathbf{a}^T\mathbf{X}\mathbf{a})}{\partial\mathbf{a}} = (\mathbf{X} + \mathbf{X}^T)\mathbf{a} \tag{A.3}$$

$$\frac{\partial\mathrm{tr}(\mathbf{X}^T\mathbf{Y}\mathbf{X})}{\partial\mathbf{X}} = (\mathbf{Y} + \mathbf{Y}^T)\mathbf{X} \tag{A.4}$$

$$\frac{\partial\ln|\mathbf{X}|}{\partial\mathbf{X}} = (\mathbf{X}^T)^{-1} \tag{A.5}$$

$$\frac{\partial(\mathbf{a}^T\mathbf{X}\mathbf{b})}{\partial\mathbf{X}} = \mathbf{a}\mathbf{b}^T \tag{A.6}$$

$$\frac{\partial(\mathbf{a}^T\mathbf{X}^T\mathbf{b})}{\partial\mathbf{X}} = \mathbf{b}^T\mathbf{a} \tag{A.7}$$

Matrix inversion lemma This lemma is useful for calculating the inverse of certain matrices in a more efficient way:

$$(\mathbf{X} + \mathbf{Y}\mathbf{Z}\mathbf{Y}^T)^{-1} = \mathbf{X}^{-1} - \mathbf{X}^{-1}\mathbf{Y}(\mathbf{Z}^{-1} + \mathbf{Y}^T\mathbf{X}^{-1}\mathbf{Y})^{-1}\mathbf{Y}^T\mathbf{X}^{-1}, \tag{A.8}$$

where $\mathbf{X}$ and $\mathbf{Z}$ are square and non-singular. If $\mathbf{X}$ is easy to invert (for example, diagonal) and $\mathbf{Y}$ has many rows and few columns, this lemma can make taking the inverse computationally efficient. It is also known as the Sherman-Morrison-Woodbury formula in the literature (Golub and Van Loan 1996, page 50).

Determinant factoring lemma For diagonal and non-singular $\ell \times \ell$ matrix $\mathbf{R}$, $\ell \times k$ matrix $\mathbf{X}$ and $k \times \ell$ matrix $\mathbf{Y}$:

$$|\mathbf{R} + \mathbf{X}\mathbf{Y}| = |\mathbf{R}||\mathbf{I}_k + \mathbf{Y}\mathbf{R}^{-1}\mathbf{X}|. \tag{A.9}$$

If $\ell \gg k$ and using that the inverse and determinant of diagonal matrix $\mathbf{R}$ are easy to calculate, this lemma can considerably simplify the calculation of the determinant.

Vec operator Function that stacks the columns of a matrix into one vector:

$$\mathrm{vec}\left[\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}\right] = (a_{11}\ a_{21}\ a_{12}\ a_{22})^T. \tag{A.10}$$

Diag operator Function from vectors to matrices that puts the vector on the main diagonal:

$$\mathrm{diag}\left[\begin{pmatrix} a_1 \\ a_2 \end{pmatrix}\right] = \begin{pmatrix} a_1 & 0 \\ 0 & a_2 \end{pmatrix}. \tag{A.11}$$

I also use it as a function from matrices to matrices that sets the off-diagonal elements to zero:

$$\mathrm{diag}\left[\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}\right] = \begin{pmatrix} a_{11} & 0 \\ 0 & a_{22} \end{pmatrix}.$$

Kronecker product The Kronecker product of two matrices is defined as:

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \otimes \mathbf{B} = \begin{pmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} \end{pmatrix}. \tag{A.12}$$

Then with the "vec" operator (A.10), we have:

$$\mathrm{vec}(\mathbf{A}\mathbf{B}\mathbf{C}) = (\mathbf{C}^T \otimes \mathbf{A})\mathrm{vec}(\mathbf{B}) \tag{A.13}$$

(Golub and Van Loan 1996, page 180). This is a fact that often comes in handy when solving matrix equations.

Element-wise product The element-wise product of two matrices is:

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \circ \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} & a_{12}b_{12} \\ a_{21}b_{21} & a_{22}b_{22} \end{pmatrix}. \tag{A.14}$$

With the "vec" (A.10) and "diag" (A.11) operators , the element-wise product can be removed from an expression:

$$\mathrm{vec}(\mathbf{A} \circ \mathbf{B}) = \mathrm{diag}\{\mathrm{vec}(\mathbf{A})\}\mathrm{vec}(\mathbf{B}) \tag{A.15}$$

Basic Probability Rules

$$P(\mathbf{x}) = \sum_{\mathbf{y}} P(\mathbf{x}, \mathbf{y}) \qquad \text{(Sum rule)} \tag{A.16}$$

$$P(\mathbf{x}, \mathbf{y}) = P(\mathbf{y}|\mathbf{x})P(\mathbf{x}) \qquad \text{(Product rule)} \tag{A.17}$$

$$P(\mathbf{x}|\mathbf{y}) = \frac{P(\mathbf{y}|\mathbf{x})P(\mathbf{x})}{P(\mathbf{y})} \qquad \text{(Bayes' theorem)} \tag{A.18}$$

Linear functions of a Gaussian vector

$$\mathbf{a} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \quad \Rightarrow \quad \mathbf{Xa} + \mathbf{b} \sim \mathcal{N}(\mathbf{X}\boldsymbol{\mu} + \mathbf{b}, \mathbf{X}\boldsymbol{\Sigma}\mathbf{X}^T), \tag{A.19}$$

which is a special case of (no matter how **a** is distributed):

$$\mathcal{E}(\mathbf{Xa} + \mathbf{b}) = \mathbf{X}\mathcal{E}(\mathbf{a}) + \mathbf{b} \tag{A.20}$$

$$\text{Covar}(\mathbf{Xa} + \mathbf{b}) = \mathbf{X}\text{Covar}(\mathbf{a})\mathbf{X}^T. \tag{A.21}$$

Marginals and Conditionals of a Gaussian

$$\mathbf{z} = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix}, \begin{bmatrix} \mathbf{X} & \mathbf{Z} \\ \mathbf{Z}^T & \mathbf{Y} \end{bmatrix} \right),$$

with **Z** the cross-variance matrix between **x** and **y**, the Gaussian marginal distributions are:

$$\mathbf{x} \sim \mathcal{N}(\mathbf{a}, \mathbf{X}) \tag{A.22}$$

$$\mathbf{y} \sim \mathcal{N}(\mathbf{b}, \mathbf{Y}), \tag{A.23}$$

and Gaussian conditional distributions:

$$\mathbf{x}|\mathbf{y} \sim \mathcal{N}(\mathbf{a} + \mathbf{Z}\mathbf{Y}^{-1}(\mathbf{y} - \mathbf{b}), \mathbf{X} - \mathbf{Z}\mathbf{Y}^{-1}\mathbf{Z}^T) \tag{A.24}$$

$$\mathbf{y}|\mathbf{x} \sim \mathcal{N}(\mathbf{b} + \mathbf{Z}^T\mathbf{X}^{-1}(\mathbf{x} - \mathbf{a}), \mathbf{Y} - \mathbf{Z}^T\mathbf{X}^{-1}\mathbf{Z}). \tag{A.25}$$

APPENDIX B

Statistical Tests

The statistical tests that have been used in the evaluation of the experiments throughout this thesis are the 5x2cv F test and McNemar's test.

5x2cv Test

Dietterich (1998a) proposes to use a 5x2cv t test if learning is not too computationally demanding and an algorithm can be repeated several times. This test is a variation on the well-known k-fold cross-validation: the data sample is divided into k portions and the algorithm is applied k times where for each run a different portion is left out for testing and the $k-1$ remaining portions are used for training. A standard choice in the machine learning literature is to have 10 folds. However, in this case there is a 80% overlap between each pair of training sets which severely invalidates the independence assumptions needed for applying a t test.

Dietterich's 5x2cv t test offers a partial (of course, having a single realization of the data of limited size, we cannot expect to guarantee independence) solution for this problem. This test consists of 5 (random) replications of 2-fold cross-validation, thus making the 2 training sets in a replication non-overlapping and reducing dependence. Dietterich shows that his test has acceptable Type I error and reasonable power. The 5x2cv t test, however, involves an arbitrary choice of some factor and a more robust variant has been proposed by Alpaydin (1999): the combined 5x2cv F test which has even lower Type I error and higher power.

The 5x2cv F test and its assumptions can be derived as follows. In each replication, the two learning algorithms A and B that we want to compare, are then trained on one halve of the data and tested on the other halve. We define:

$$p_{ij} := \text{difference between the (test) error of the two algorithms on fold } j \text{ in replication } i \qquad j = 1, 2 \text{ and } i = 1, \cdots, 5.$$

The test is now based on the following assumptions under the null hypothesis that the two algorithms have the same error rate. Each of the error rates can be considered normally distributed (in the case of regression) or binomially distributed (in the case of classification) and hence approximated by a normal distribution, for example (Feller 1970, chapter 7) and (Mitchell 1997, chapter 5). Each of the p_{ij} is then the difference of two normally and identically distributed quantities and (wrongly) assuming that they are independent:

$$p_{ij} \sim \mathcal{N}(0,\sigma) \quad \Rightarrow \quad p_{ij}/\sigma \sim \mathcal{N}(0,1) \quad \Rightarrow \quad p_{ij}^2/\sigma^2 \sim \chi_1^2 \quad \overset{*}{\Rightarrow} \quad N := \sum_i \sum_j (p_{ij}^2)/\sigma^2 \sim \chi_{10}^2. \tag{B.1}$$

On the other hand, denoting the mean of the error rates of replication i as $\overline{p}_i$ and the estimated variance as $\mathrm{var}_i = (p_{i1} - \overline{p}_i)^2 - (p_{i2} - \overline{p}_i)^2$ and (wrongly) assuming that p_{i1} and p_{i2} are independent:

$$p_{ij} \sim \mathcal{N}(0,\sigma) \quad \Rightarrow \quad \mathrm{var}_i/\sigma^2 \sim \chi^2_1 \quad \overset{*}{\Rightarrow} \quad M := \sum_i \mathrm{var}_i/\sigma^2 \sim \chi^2_5, \tag{B.2}$$

where in both steps marked with a $*$ in (B.1) and (B.2), we also (wrongly) assumed independence of the summands. Finally, we can use a well-known fact about F distributions (Kreyszig 1999):

$$X_1 \sim \chi^2_n \text{ and } X_2 \sim \chi^2_m \text{ and } X_1, X_2 \text{ independent } \Rightarrow \frac{X_1/n}{X_2/m} \sim F_{n,m}.$$

And (again) wrongly assuming that N and M defined in (B.1) and (B.2) respectively, are independent, we arrive at the 5x2cv F test (Alpaydin 1999):

$$f = \frac{N/10}{M/5} = \frac{\sum_i \sum_j (p_{ij}^2)}{2\sum_i \mathrm{var}_i} \sim F_{10,5}.$$

Thus, the null hypothesis that the two algorithms have the same error rate can be rejected with 95% confidence if $f > F_{10,5} = 4.74$.

McNemar's Test

Dietterich (1998a) proposes to use McNemar's test if learning is computationally expensive and can only be performed once. In this case, the data sample is split in a fixed training set and test set. The two learning algorithms A and B that we want to compare, are then trained on the training set and their prediction on the test set leads to the following contingency table:

$$\begin{array}{c|c} n_{00} & n_{01} \\ \hline n_{10} & n_{11} \end{array} := \begin{array}{c|c} \text{\# of ex. misclassified by } A \text{ and } B & \text{\# of ex. misclassified by } A\text{, not by } B \\ \hline \text{\# of ex. misclassified by } B\text{, not by } A & \text{\# of ex. correctly classified by } A \text{ and } B \end{array}$$

Under the null hypothesis that the two algorithms have the same error rate, the following statistic is approximately χ^2 distributed with one degree of freedom (Everitt 1977):

$$q = \frac{(|n_{01} - n_{10}| - 1)^2}{n_{01} + n_{10}}.$$

Thus, the null hypothesis that the two algorithms have the same error rate can be rejected with 95% confidence if $q > \chi^2_1 = 3.84$.

Of course, a major disadvantage of McNemar's test is the fact that it is entirely based on fixed sets and thus does not take into account variability. However, since this test is typically applied when the given data set is big, this variability is expected to be quite small. According to Dietterich (1998a), McNemar's test has acceptable Type I error and is also quite powerful.

Remark The panoply of assumptions that were necessary to derive McNemar's and the 5x2cv F test clearly illustrates one of the main problems of experimental evaluations in machine learning. A different approach has been taken in the DELVE project (Rasmussen et al. 1996) in which large data sets are used to be able to construct various non-overlapping training and test sets of varying size. This allows for more independence between runs and a more systematic exploration of the problem space. But as stated by Neal (1998), the major weakness of DELVE is its lack of data sets; this is especially so for classification problems. Therefore, I opted for a more "traditional" experimental protocol in my thesis. **End of** Remark

APPENDIX C

Data Sets

Most of the data sets used in this thesis can be obtained from the UCI Machine Learning Repository (Blake, Keogh, and Merz 1998) via the Internet: `http://www.ics.uci.edu/~mlearn/MLRepository.html`. At the same address a more detailed description of the data sets and further references can also be found. The list below gives, for each entry, the name I used to designate a data set in this thesis, followed between brackets by the name under which one can find it in the repository and a short description. The reader is referred to Table 2.1 for a compact representation of some characteristics of the data sets.

banana Two-dimensional toy problem with two classes generated from several nonlinearly transformed Gaussian and uniform blobs (Figure C.1). It was first used in (Rätsch, Onoda, and Müller 1998). A script for generating the banana data has been made available by Gunnar Rätsch: `http://www.first.gmd.de/~raetsch/data/banana.txt`

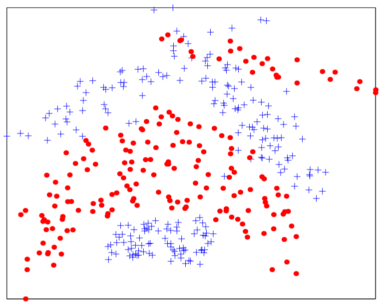


Figure C.1: banana data

cancer (Wisconsin Breast Cancer) Each example has one of 2 possible classes: benign or malignant. The breast cancer data set was made available by Dr. William H. Wolberg from the University of Wisconsin Hospitals, Madison (Mangasarian and Wolberg 1990).

dermatology Diagnosis of 6 dermatological (erythemato-squamous) diseases.

glass (Glass Identification) Classification of 6 types of glass motivated by criminological investigation. At the scene of the crime, the glass left can be used as evidence ... if it is correctly identified.

heart (Heart Disease) I used the Cleveland part of the heart disease data sets made available by V.A. Medical Center, Long Beach and Cleveland Clinic Foundation: Robert Detrano, M.D., Ph.D. Goal is to distinguish between presence and absence of heart disease in a patient.

ionosphere Classification of radar returns from the ionosphere. The targets were free electrons in the ionosphere. "Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not; their signals pass through the ionosphere.

iris R. A. Fisher's classic data set where the goal is to predict the type of iris plant given the sepal and petal lengths and widths.

letter (Letter Recognition) The objective is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the alphabet. Each display was converted into 16 primitive numerical attributes (statistical moments and edge counts).

NIST Data from the NIST Special Database 3 of handwritten digits (Garris and Wilkinson 1992). Data has been size-normalized, deskewed, centered, and smoothed to obtain 16×16 images with their values scaled to the interval $[0, 1]$. The training set consists of 15,025 digits from 140 writers and the test set of 4,975 digits from 48 writers (not overlapping with training set).

optical (Optical Recognition of Handwritten Digits) From a total of 43 people, 30 contributed handwritten digits to the training set and another 13 to the test set. This gives a training set of 3823 examples and a test set of 1797 examples. Note that I used only the training set in all experiments.

pen (Pen-Based Recognition of Handwritten Digits) A digit database obtained by collecting 250 samples from 44 writers on a pressure sensitive tablet with a cordless stylus. The digits are represented as a sequence of points regularly spaced in arc length. Only the training set of 30 writers was used in the experiments.

pima (Pima Indians Diabetes) The diagnostic investigated is whether the patient (Pima Indian woman) shows signs of diabetes.

satimage (part of the Statlog Project Databases) Generated from Landsat satellite data. Goal is to predict the type of soil given multi-spectral values of pixels in 3×3 neighborhoods in a satellite image. Data comes as a separate training (4435 examples) and test set (2000 examples).

segmentation (Image Segmentation) The instances were drawn randomly from a database of 7 outdoor images. The images were handsegmented to create a classification of the type of surface for every pixel.

sonar (part of the Undocumented Databases) Discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock.

soybean Prediction of 19 types of soybean disease given 35 categorical attributes.

twos The subset of "twos" from our NIST data set.

vowel (part of the Undocumented Databases) Speaker independent recognition of the eleven steady state vowels of British English using LPC derived log area ratios.

waveform (Waveform Data Generator) Toy data from (Breiman, Friedman, Olshen, and Stone 1984, pages 49–55). A script for generating the data is available from the UCI repository. See also page 40 of this thesis for a more detailed description. Known Bayes optimal classification rate of 86% accuracy.

waveform-noise (Waveform Data Generator) Like waveform but with 19 extra attributes that are all noise attributes with mean 0 and variance 1.

APPENDIX D

Vision Data Sets

face Data Set

The face data set as available at IDIAP (Ben-Yacoub 1997) consists of 25×25 images of 1476 faces and 1628 non-faces. Out of these two ensembles a training and test set were made of equal size each containing 738 faces and 814 non-faces. Some examples of the training and the test set are shown in Figure D.1 and D.2. The face images contain both men and women[1], people with glasses, a beard or a mustache, and a variety of different facial expressions. The non-face images were mainly taken from general textured images, although some contain a small part of a face (for example, the first two non-faces in Figure D.1).

Figure D.1: Some examples of faces and non-faces in the face training set.

Figure D.2: Some examples of faces and non-faces in the face test set.

In the experiments described in section 6.1, the training data has been further subdivided into a set for determining the parameters of a model consisting of 200 faces and 200 non-faces, and a validation set for model selection with the remaining images (538 faces and 614 non-faces).

MNIST Data Set

The MNIST handwritten digit data set has been made available by Y. LeCun of AT&T Labs Research at `http://www.research.att.com/~yann/ocr/mnist/index.html`. This data set has a training set

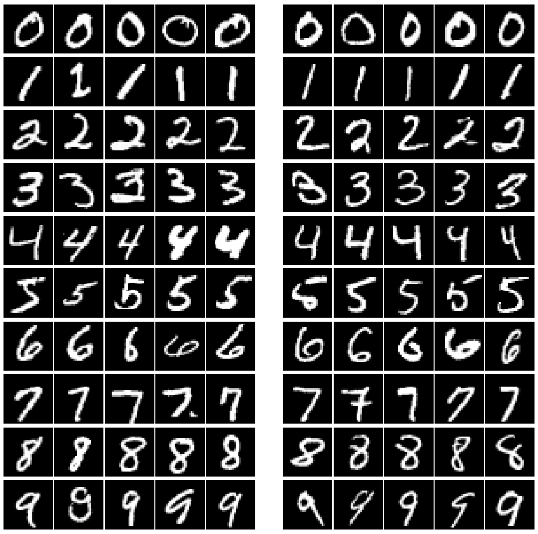
[1] though not many.

of 60,000 examples and a test set of 10,000 examples. It is actually a subset of the union of two different data sets provided by the US National Institute of Standards and Technology (NIST): NIST Special Database 3 (SD-3)[2] and NIST Test Data 1 (SD-1). The former has been collected amongst US census workers while SD-1 consists of digits written by high school students. SD-1 is, therefore, much more difficult to recognize than SD-3.

At AT&T out of SD-1 and SD-3, the MNIST data set has been constructed to obtain a bigger and more balanced data set. The 60,000 example training set is composed of 30,000 patterns from SD-1 and 30,000 patterns from SD-3. The 10,000 example test set is composed of 5,000 patterns from SD-1 and 5,000 patterns from SD-3. The training set contains examples from approximately 500 writers. Sets of writers of the training set and test set are disjoint.

The original black and white (bilevel) images from NIST were size normalized to fit in a 20×20 pixel box while preserving their aspect ratio. The resulting images contain (8-bit) grey levels as a result of the anti-aliasing technique used by the normalization algorithm. These images were centered in a 28×28 image by computing the center of mass of the pixels and translating the image.

Figure D.3 shows 10 examples of each digit from both the training and the test set. As one can see, there is a large variation in line thickness, writing style and angle, and even some noise (for example, in the box of the fifth 6 and fourth 7 in Figure D.3a). In most of the experiments described in section 6.2 the images were subsampled to 16×16 pixels to save computation time.



(a) Training data. (b) Test data.

Figure D.3: Examples of hand-written digits in the MNIST data set.

[2] of which the NIST data used throughout my thesis is again another subset.

APPENDIX E

From Factor Analyzers to Principal Component Analyzers

It is described in detail how to transform the EM algorithm for a MFA (Algorithm 3, page 34) in an EM algorithm for MPCA (Algorithm 4, page 36) in the form as it occurs in (Tipping and Bishop 1999).

The first stage of the two algorithms is exactly the same; therefore, we focus on the second stage of the algorithms and for convenience it is repeated here for the MFA algorithm:

{Second stage: E-step (EM for MFA)}
for $j := 1$ to m **do**
  $\mathbf{N}_j := \mathbf{I}_\ell + \mathbf{W}_j^T \mathbf{R}_j^{-1} \mathbf{W}_j$
  for $n := 1$ to N **do**
    $\langle \mathbf{z}_j^n \rangle := \mathbf{N}_j^{-1} \mathbf{W}_j^T \mathbf{R}_j^{-1} (\mathbf{x}^n - \boldsymbol{\mu}_j)$
    $\langle \mathbf{z}_j^n (\mathbf{z}_j^n)^T \rangle := \mathbf{N}_j^{-1} + \langle \mathbf{z}_j^n \rangle \langle \mathbf{z}_j^n \rangle^T$
  end for
end for
{Second stage: M-step}
for $j := 1$ to m **do**
  $\mathbf{W}_j := \left[\sum_n h_j(\mathbf{x}^n)(\mathbf{x}^n - \boldsymbol{\mu}_j)\langle \mathbf{z}_j^n \rangle^T\right]\left[\sum_n h_j(\mathbf{x}^n)\langle \mathbf{z}_j^n (\mathbf{z}_j^n)^T \rangle\right]^{-1}$
  $\mathbf{R}_j := \frac{1}{\sum_n h_j(\mathbf{x}^n)} \mathrm{diag}\left[\sum_n h_j(\mathbf{x}^n)\{(\mathbf{x}^n - \boldsymbol{\mu}_j) - \mathbf{W}_j \langle \mathbf{z}_j^n \rangle\}(\mathbf{x}^n - \boldsymbol{\mu}_j)^T\right]$
end for

A first simple rewriting consists of replacing the diagonal noise covariance matrices of a MFA by the single variance parameter σ_j^2 of a MPCA in the equations for the E-step:

$$\mathbf{N}_j = \mathbf{I}_\ell + \mathbf{W}_j^T \mathbf{W}_j / \sigma_j^2 \tag{E.1}$$

$$\langle \mathbf{z}_j^n \rangle = \mathbf{N}_j^{-1} \mathbf{W}_j^T (\mathbf{x}^n - \boldsymbol{\mu}_j) / \sigma_j^2 . \tag{E.2}$$

Now, it is merely a question of manipulating the updates of the parameters $\mathbf{W}_j$ and $\mathbf{R}_j$ (which of course has to be reduced to an update of σ_j^2). We start with the factor loading matrix $\mathbf{W}_j$:

$$\mathbf{W}'_j = \Big[\sum_n h_j(\mathbf{x}^n)(\mathbf{x}^n - \boldsymbol{\mu}_j)\langle \mathbf{z}_j^n \rangle^T\Big]\Big[\sum_n h_j(\mathbf{x}^n)\langle \mathbf{z}_j^n (\mathbf{z}_j^n)^T \rangle\Big]^{-1}$$
$$= \Big[\sum_n h_j(\mathbf{x}^n)(\mathbf{x}^n - \boldsymbol{\mu}_j)\langle \mathbf{z}_j^n \rangle^T\Big]\Big[\sum_n h_j(\mathbf{x}^n)(\mathbf{N}_j^{-1} + \langle \mathbf{z}_j^n \rangle\langle \mathbf{z}_j^n \rangle^T)\Big]^{-1}.$$

Substituting (E.2) and using that $\mathbf{N}_j$ is symmetric:

$$= \Big[\sum_n h_j(\mathbf{x}^n)(\mathbf{x}^n - \boldsymbol{\mu}_j)(\mathbf{x}^n - \boldsymbol{\mu}_j)^T\Big](\mathbf{W}_j\mathbf{N}_j^{-1}/\sigma_j^2)\Big[\sum_n h_j(\mathbf{x}^n)\{\mathbf{N}_j^{-1} +$$
$$+ \mathbf{N}_j^{-1}\mathbf{W}_j^T(\mathbf{x}^n - \boldsymbol{\mu}_j)(\mathbf{x}^n - \boldsymbol{\mu}_j)^T\mathbf{W}_j\mathbf{N}_j^{-1}/\sigma_j^4\}\Big]^{-1}.$$

Defining the weighted covariance matrix $\mathbf{S}_j = \frac{1}{\sum_n h_j(\mathbf{x}^n)}\sum_n h_j(\mathbf{x}^n)(\mathbf{x}^n - \boldsymbol{\mu}_j)(\mathbf{x}^n - \boldsymbol{\mu}_j)^T$ and using that $\mathbf{A}^{-1}\mathbf{B}^{-1} = (\mathbf{B}\mathbf{A})^{-1}$:

$$= \Big[\sum_n h_j(\mathbf{x}^n)\Big]\mathbf{S}_j\mathbf{W}_j\Big[\sum_n h_j(\mathbf{x}^n)\{\sigma_j^2\mathbf{I}_\ell + \mathbf{N}_j^{-1}\mathbf{W}_j^T(\mathbf{x}^n - \boldsymbol{\mu}_j)(\mathbf{x}^n - \boldsymbol{\mu}_j)^T\mathbf{W}_j/\sigma_j^2\}\Big]^{-1}$$
$$= \Big[\sum_n h_j(\mathbf{x}^n)\Big]\mathbf{S}_j\mathbf{W}_j\Big[\sum_n h_j(\mathbf{x}^n)\sigma_j^2\mathbf{I}_\ell + \sum_n h_j(\mathbf{x}^n)\mathbf{N}_j^{-1}\mathbf{W}_j^T(\mathbf{x}^n - \boldsymbol{\mu}_j)(\mathbf{x}^n - \boldsymbol{\mu}_j)^T\mathbf{W}_j/\sigma_j^2\Big]^{-1}.$$

Moving $\sum_n h_j(\mathbf{x}^n)$ within the inverse:

$$= \mathbf{S}_j\mathbf{W}_j\Big[\sigma_j^2\mathbf{I}_\ell + \{\sum_n h_j(\mathbf{x}^n)(\sigma_j^2\mathbf{N}_j)^{-1}\mathbf{W}_j^T(\mathbf{x}^n - \boldsymbol{\mu}_j)(\mathbf{x}^n - \boldsymbol{\mu}_j)^T\mathbf{W}_j\}\sum_n h_j(\mathbf{x}^n)\Big]^{-1}$$
$$= \mathbf{S}_j\mathbf{W}_j\Big[\sigma_j^2\mathbf{I}_\ell + (\sigma_j^2\mathbf{N}_j)^{-1}\mathbf{W}_j^T\mathbf{S}_j\mathbf{W}_j\Big]^{-1}.$$

Defining, $\mathbf{M}_j = \sigma_j^2\mathbf{N}_j = \sigma^2\mathbf{I}_\ell + \mathbf{W}_j^T\mathbf{W}_j$, the new factor loadings become:

$$\mathbf{W}'_j = \mathbf{S}_j\mathbf{W}_j(\sigma_j^2\mathbf{I}_\ell + \mathbf{M}_j^{-1}\mathbf{W}_j^T\mathbf{S}_j\mathbf{W}_j)^{-1}. \tag{E.3}$$

The MFA update of the noise covariance matrix is:

$$\mathbf{R}'_j = \frac{1}{\sum_n h_j(\mathbf{x}^n)}\mathrm{diag}\Big[\sum_n h_j(\mathbf{x}^n)\{(\mathbf{x}^n - \boldsymbol{\mu}_j) - \mathbf{W}'_j\langle \mathbf{z}_j^n \rangle\}(\mathbf{x}^n - \boldsymbol{\mu}_j)^T\Big]$$
$$= \mathrm{diag}\Big[\mathbf{S}_j - \frac{1}{\sum_n h_j(\mathbf{x}^n)}\sum_n h_j(\mathbf{x}^n)\mathbf{W}'_j\langle \mathbf{z}_j^n \rangle(\mathbf{x}^n - \boldsymbol{\mu}_j)^T\Big].$$

Substituting (E.2):

$$= \mathrm{diag}\Big[\mathbf{S}_j - \frac{1}{\sum_n h_j(\mathbf{x}^n)}\mathbf{W}'_j(\mathbf{N}_j^{-1}/\sigma_j^2)\mathbf{W}_j^T\sum_n h_j(\mathbf{x}^n)(\mathbf{x}^n - \boldsymbol{\mu}_j)(\mathbf{x}^n - \boldsymbol{\mu}_j)^T\Big]$$
$$= \mathrm{diag}\Big[\mathbf{S}_j - \mathbf{W}'_j(\mathbf{N}_j^{-1}/\sigma_j^2)\mathbf{W}_j^T\mathbf{S}_j\Big].$$

Replacing the diagonal $\mathbf{R}_j$ with σ_j^2 by taking the trace and dividing by the length of the diagonal, the new noise variance becomes:

$$(\sigma_j^2)' = \frac{1}{d}\mathrm{tr}(\mathbf{S}_j - \mathbf{W}'_j\mathbf{M}_j^{-1}\mathbf{W}_j^T\mathbf{S}_j). \tag{E.4}$$

The second stage of the EM algorithm for MPCA can now be written as (using (E.3),(E.4), and the definitions of $\mathbf{M}_j$ and the weighted covariance matrix $\mathbf{S}_j$):

{Second stage (EM for MPCA)}
for $j := 1$ to m **do**
  $\mathbf{S}_j := \frac{1}{\alpha_j N}\sum_n h_j(\mathbf{x}^n)(\mathbf{x}^n - \boldsymbol{\mu}_j)(\mathbf{x}^n - \boldsymbol{\mu}_j)^T$
  $\mathbf{M}_j := \sigma_j^2\mathbf{I}_\ell + \mathbf{W}_j^T\mathbf{W}_j$
  $\mathbf{W}_j, \mathbf{W}_{\mathrm{old},j} := \mathbf{S}_j\mathbf{W}_j(\sigma_j^2\mathbf{I}_\ell + \mathbf{M}_j^{-1}\mathbf{W}_j^T\mathbf{S}_j\mathbf{W}_j)^{-1}, \mathbf{W}_j$
  $\sigma_j^2 := \frac{1}{d}\mathrm{tr}(\mathbf{S}_j - \mathbf{W}_j\mathbf{M}_j^{-1}\mathbf{W}_{\mathrm{old},j}^T\mathbf{S}_j)$
end for

---

and finally gives us Algorithm 4, page 36. This is exactly the same algorithm as derived by Tipping and Bishop (1999); the reader is referred to section 2.4 for a discussion of its complexity.

APPENDIX **F**

EM for Gaussian Mixture Models

For completeness' sake, I will give a short derivation of the M-step of the EM algorithm for a Gaussian mixture model (see also any of the standard text books (Titterington et al. 1985; McLachlan and Basford 1988; Bishop 1995)). Each probability density function p_j is chosen to be a spherical Gaussian with variance σ_j^2:

$$p_j(\mathbf{x}|\boldsymbol{\theta}_j) = \frac{1}{(2\pi\sigma_j^2)^{d/2}} \exp\left(-\frac{||\mathbf{x}-\boldsymbol{\mu}_j||^2}{2\sigma_j^2}\right).$$

The expected complete error function to be minimized in the M-step is as we have seen (2.13):

$$E_c = -\sum_n \sum_{j=1}^m h_j(\mathbf{x}^n) \ln\{\alpha_j^{\text{new}} p_j(\mathbf{x}^n|\boldsymbol{\theta}_j^{\text{new}})\} = -\sum_n \sum_{j=1}^m h_j(\mathbf{x}^n) \ln \alpha_j^{\text{new}} - \sum_n \sum_{j=1}^m h_j(\mathbf{x}^n) \ln p_j(\mathbf{x}^n|\boldsymbol{\theta}_j^{\text{new}}), \tag{F.1}$$

The estimation of the mixing coefficients is as before (2.14):

$$\alpha_j^{\text{new}} = \frac{1}{N} \sum_n h_j(\mathbf{x}^n),$$

where N is the number of patterns in the training set $\{\mathbf{x}^n\}$.

For the centers of the Gaussian kernels, the partial derivative of the second part of the complete error function (F.1) gives:

$$\frac{\partial\left[-\sum_n \sum_{j=1}^m h_j(\mathbf{x}^n) \ln p_j(\mathbf{x}^n|\boldsymbol{\theta}_j^{\text{new}})\right]}{\partial \boldsymbol{\mu}_j^{\text{new}}} = -\sum_n \frac{h_j(\mathbf{x}^n)}{p_j(\mathbf{x}^n|\boldsymbol{\theta}_j^{\text{new}})} \frac{\partial p_j(\mathbf{x}^n|\boldsymbol{\theta}_j^{\text{new}})}{\partial \boldsymbol{\mu}_j^{\text{new}}} = -\sum_n h_j(\mathbf{x}^n) \frac{(\mathbf{x}^n - \boldsymbol{\mu}_j^{\text{new}})}{(\sigma_j^{\text{new}})^2}.$$

Setting this partial derivative to zero, we obtain a new estimate for the means:

$$\boldsymbol{\mu}_j^{\text{new}} = \frac{\sum_n h_j(\mathbf{x}^n)\mathbf{x}^n}{\sum_n h_j(\mathbf{x}^n)}.$$

For the variance parameters of the Gaussian kernels, the partial derivative of the second part of the complete error function (F.1) is:

$$\begin{aligned}\frac{\partial\left[-\sum_n \sum_{j=1}^m h_j(\mathbf{x}^n) \ln p_j(\mathbf{x}^n|\boldsymbol{\theta}_j^{\text{new}})\right]}{\partial \sigma_j^{\text{new}}} &= -\sum_n \frac{h_j(\mathbf{x}^n)}{p_j(\mathbf{x}^n|\boldsymbol{\theta}_j^{\text{new}})} \frac{\partial p_j(\mathbf{x}^n|\boldsymbol{\theta}_j^{\text{new}})}{\partial \sigma_j^{\text{new}}} \\ &= -\sum_n h_j(\mathbf{x}^n) \left[\frac{||\mathbf{x}^n - \boldsymbol{\mu}_j^{\text{new}}||^2}{(\sigma_j^{\text{new}})^3} - \frac{d}{\sigma_j^{\text{new}}}\right].\end{aligned}$$

Setting this partial derivative to zero, the new estimate for the variances is:

$$(\sigma_j^{\text{new}})^2 = \frac{1}{d}\frac{\sum_n h_j(\mathbf{x}^n)||\mathbf{x}^n - \boldsymbol{\mu}_j^{\text{new}}||^2}{\sum_n h_j(\mathbf{x}^n)},$$

which completes the M-step for the Gaussian kernels.

References

Aizerman, M., E. Braverman, and L. Rozonoer (1964). Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control 25*, 821–837.

Alpaydin, E. (1999, November). Combined 5x2cv F test for comparing supervised classification learning algorithms. *Neural Computation 11*(8), 1885–1892.

Alpaydin, E. and M. I. Jordan (1996, May). Local linear perceptrons for classification. *IEEE Transactions on Neural Networks 7*(3), 788–792.

Auer, P., M. Hebster, and M. K. Warmuth (1996). Exponentially many local minima for single neurons. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo (Eds.), *Advances in Neural Information Processing Systems*, Volume 8, pp. 316–322. Cambridge, MA: MIT Press.

Avnimelech, R. and N. Intrator (1999, February). Boosted mixture of experts: an ensemble learning scheme. *Neural Computation 11*(2), 483–498.

Bartholomew, D. J. and M. Knott (1999). *Latent Variable Models and Factor Analysis* (2nd ed.). London: Arnold.

Bauer, E. and R. Kohavi (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning 36*, 105–146.

Bellegarda, J. and D. Nahamoo (1990). Tied mixture continuous parameter modeling for speech recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing 38*, 2033–2045.

Ben-Yacoub, S. (1997). Fast object detection using MLP and FFT. Technical Report IDIAP-RR 97-11, IDIAP, Martigny, Switzerland.

Berger, J. O. (1985). *Statistical Decision Theory and Bayesian Analysis* (2nd ed.). New York: Springer-Verlag.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford: Oxford University Press.

Bishop, C. M. (1999a). Bayesian PCA. In M. S. Kearns, S. A. Solla, and D. A. Cohn (Eds.), *Advances in Neural Information Processing Systems*, Volume 11, pp. 382–388. Cambridge, MA: MIT Press.

Bishop, C. M. (1999b). Latent variable models. See Jordan (1999), pp. 371–403.

Bishop, C. M. (1999c). Variational principal components. In *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN'99)*, Volume 1, pp. 509–514. London: IEE.

Bishop, C. M. and M. E. Tipping (1998, March). A hierarchical latent variable model for data visualization. *IEEE Transactions on Pattern Analysis and Machine Intelligence 20*(3), 281–293.

Blake, C., E. Keogh, and C. J. Merz (1998). UCI repository of machine learning databases. Irvine: University of California, Department of Information and Computer Sciences. `www.ics.uci.edu/~mlearn/MLRepository.html`.

Bourlard, H. and N. Morgan (1994). *Connectionist Speech Recognition: A Hybrid Approach*. Kluwer Academic Publishers.

Bradshaw, N. P., A. Duchâteau, and H. Bersini (1997). Global least-squares vs. EM training for the Gaussian mixture of experts. In W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud (Eds.), *Artificial Neural Networks - ICANN'97*, Number 1327 in Lecture Notes in Computer Science, pp. 295–300. Berlin: Springer-Verlag.

Breiman, L. (1996). Bagging predictors. *Machine Learning 26*(2), 123–140.

Breiman, L. (1997). Arcing the edge. Technical Report 486, Statistics Department, University of California, Berkeley. `ftp://ftp.stat.berkeley.edu/pub/users/breiman/arcing-the-edge.ps.Z`.

Breiman, L., J. H. Friedman, R. A. Olshen, and C. Stone (1984). *Classification and Regression Trees*. Belmont, California: Wadsworth.

Bridle, J. S. (1990). Probabilistic interpretation of feedforward classification network outputs with relationships to statistical pattern recognition. In F. F. Soulié and J. Hérault (Eds.), *Neurocomputing: Algorithms, Architectures, and Applications*, pp. 227–236. New York: Springer Verlag.

Carreira-Perpiñán, M. Á. and S. J. Renals (1998, December). Dimensionality reduction of electropalatographic data using latent variable models. *Speech Communication 26*(4), 259–282.

Cover, T. M. and J. A. Thomas (1991). *Elements of Information Theory*. New York: John Wiley & Sons.

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, B 39*(1), 1–38.

Diamantaras, K. I. and S. Y. Kung (1996). *Principal Component Neural Networks: Theory and Applications*. New York: John Wiley & Sons, Inc.

Dietterich, T. G. (1998a). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation 10*(7), 1895–1923.

Dietterich, T. G. (1998b). An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. *Machine Learning*. To appear.

Duda, R. O. and P. E. Hart (1973). *Pattern Classification and Scene Analysis*. New York: John Wiley & Sons.

Everitt, B. S. (1977). *The Analysis of Contingency Tables*. London: Chapman and Hall.

Everitt, B. S. (1984). *An Introduction to Latent Variable Models*. London: Chapman and Hill.

Feller, W. (1970). *An Introduction to Probability Theory and Its Applications* (Revised printing, 3rd ed.), Volume I. John Wiley & Sons.

Fisher, R. A. (1952). *Contributions to Mathematical Statistics*. New York: J. Wiley.

Fournier, R. (1998, October). Prosodie als Hilfsmittel für die automatische Spracherkennung. Master's thesis, Universität Freiburg, Knüslistraße 6, Zürich, Switzerland.

Freund, Y. (1995). Boosting a weak learning algorithm by majority. *Information and Computation 121*(2), 256–285.

Freund, Y. and R. Schapire (1996). Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, pp. 148–156. San Francisco, CA: Morgan Kaufmann.

Freund, Y. and R. E. Schapire (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences 55*(1), 119–139.

Frey, B. J. (1998). *Graphical Models for Machine Learning and Digital Communication*. Cambridge, MA: MIT Press.

Frey, B. J. (1999). Turbo factor analysis. Technical Report 99-1, Dept. of Computer Science, University of Waterloo. `http://www.cs.toronto.edu/~frey/papers/tfa-nc99.ps.Z`.

Frey, B. J., A. Colmenarez, and T. S. Huang (1998). Mixtures of local linear subspaces for face recognition. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition 1998*. Los Alamitos, CA: IEEE Comp. Soc. Press.

Friedman, J., T. Hastie, and R. Tibshirani (1998). Additive logistic regression: a statistical view of boosting. Technical report, Department of Statistics, Stanford University. `ftp://utstat.toronto.edu/pub/tibs/boost.ps.Z`.

Fritsch, J. (1996). Modular neural networks for speech recognition. Master's thesis, Carnegie Mellon University. `ftp://reports.adm.cs.cmu.edu/usr/anon/1996/CMU-CS-96-203.ps.gz`.

Garris, M. D. and R. D. Wilkinson (1992, February). NIST special database 3: handwritten segmented characters. Technical report, National Insitute of Standards and Technology.

Ghahramani, Z. and M. J. Beal (2000). Variational inference for Bayesian mixture of factor analysers. In S. A. Solla, T. K. Leen, and K.-R. Müller (Eds.), *Advances in Neural Information Processing Systems*, Volume 12. Cambridge, MA: MIT Press. `http://www.gatsby.ucl.ac.uk/~zoubin/papers/nips99.ps.gz`.

Ghahramani, Z. and G. E. Hinton (1996). The EM algorithm for mixtures of factor analyzers. Technical Report CRG-TR-96-1, University of Toronto. `ftp://ftp.cs.toronto.edu/pub/zoubin/tr-96-1.ps.gz`.

Ghahramani, Z. and M. I. Jordan (1994). Supervised learning from incomplete data via an EM approach. In J. D. Cowan, G. T. Tesauro, and J. Alspector (Eds.), *Advances in Neural Information Processing Systems*, Volume 6, pp. 120–127. San Mateo, CA: Morgan Kaufmann.

Golub, G. H. and C. F. Van Loan (1996). *Matrix Computations* (3rd ed.). Baltimore: The Johns Hopkins University Press.

Hampshire, J. B. and B. Pearlmutter (1990). Equivalence proofs for multilayer perceptron classifiers and the Bayesian discriminant function. In D. S. Touretzky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton (Eds.), *Proceedings of the 1990 Connectionist Models Summer School*, pp. 159–172. San Mateo, CA: Morgan Kaufmann.

Hastie, T. and W. Stuetzle (1989). Principal curves. *Journal of the American Statistical Association 84*, 502–516.

Hastie, T. and R. Tibshirani (1996). Discriminant analysis by Gaussian mixtures. *Journal of the Royal Statistical Society (Series B) 58*, 155–176.

Hastie, T., R. Tibshirani, and A. Buja (1999). Flexible discriminant and mixture models. In J. Kay and D. Titterington (Eds.), *Statistics and Neural Networks*. Oxford: Oxford University Press.

Hinton, G. E., P. Dayan, and M. Revow (1997). Modelling the manifolds of images of handwritten digits. *IEEE Transactions on Neural Networks 8*(1), 65–74.

Hoffman, T. (1999). Probabilistic latent semantic analysis. In *Proceedings of UAI'99*.

Jacobs, R. A., M. I. Jordan, S. J. Nowlan, and G. E. Hinton (1991). Adaptive mixtures of local experts. *Neural Computation 3*(1), 79–87.

Jebara, T. and A. Pentland (1999). Maximum conditional likelihood via bound maximization and the CEM algorithm. In M. S. Kearns, S. A. Solla, and D. A. Cohn (Eds.), *Advances in Neural Information Processing Systems*, pp. 494–500. Cambridge, MA: MIT Press.

Jiang, W. and M. A. Tanner (1999a). Hierarchical mixtures-of-experts for generalized linear models: Some results on densenes and consistency. In D. Heckerman and J. Whittaker (Eds.), *Proceedings of the Seventh International Workshop on Artificial Intelligence and Statistics*. San Francisco, CA: Morgan Kaufmann Publishers.

Jiang, W. and M. A. Tanner (1999b). On the approximation rate of hierarchical mixtures-of-experts for generalized linear models. *Neural Computation 11*(5), 1183–1198.

Jojic, N. and B. J. Frey (2000). Topographic transformation as a discrete latent variable. In S. A. Solla, T. K. Leen, and K.-R. Müller (Eds.), *Advances in Neural Information Processing Systems*, Volume 12. Cambridge, MA: MIT Press. `http://www.cs.toronto.edu/~frey/papers/ttdlv-nips99.ps.Z`.

Jollife, I. T. (1986). *Principal Component Analysis*. New York: Springer-Verlag.

Jordan, M. I. (Ed.) (1999). *Learning in Graphical Models*. Adaptive Computation and Machine Learning. Cambridge, MA: MIT Press.

Jordan, M. I., Z. Ghahramani, T. S. Jaakkola, and L. Saul (1999). An introduction to variational methods for graphical models. See Jordan (1999), pp. 105–161.

Jordan, M. I. and R. A. Jacobs (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation 6*(2), 181–214.

Jordan, M. I. and L. Xu (1995). Convergence results for the EM approach to mixtures of experts architectures. *Neural Networks 8*(9), 1409–1431.

Jost, J. and X. Li-Jost (1998). *Calculus of Variations*. Cambridge: Cambridge University Press.

Kambhatla, N. and T. K. Leen (1995). Classifying with Gaussian mixtures and clusters. In G. Tesauro, D. Touretzky, and T. Leen (Eds.), *Advances in Neural Information Processing Systems*, Volume 7, pp. 681–688. Cambridge, MA: MIT Press.

Kambhatla, N. and T. K. Leen (1997). Dimension reduction by local principal component analysis. *Neural Computation 9*(7), 1493–1516.

Kirby, M. and L. Sirovich (1990). Application of the karhunen-loève procedure for the characterization of human faces. *IEEE Transactions on Pattern Analysis and Machine Intelligence 12*(1), 103–108.

Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AICHE Journal 37*(2), 233–243.

Kreßel, U. H.-G. (1999). Pairwise classification and support vector machines. In B. Schölkopf, C. Burges, and A. Smola (Eds.), *Advances in Kernel Methods - Support Vector Learning*, pp. 255–268. Cambridge, MA: MIT Press.

Kreyszig, E. (1999). *Advanced Engineering Mathematics* (6th ed.). New York: Wiley.

LeCun, Y. (2000). The MNIST database of handwritten digits. `http://www.research.att.com/~yann/ocr/mnist/index.html`.

LeCun, Y., L. D. Jackel, L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Muller, E. Sackinger, P. Simard, and V. Vapnik (1995). Learning algorithms for classification: A comparison on handwritten digit recognition. In J. H. Oh, C. Kwon, and S. Cho (Eds.), *Neural Networks: The Statistical Mechanics Perspective*, pp. 261–276. Singapore: World Scientific.

MacKay, D. J. C. (1991). *Bayesian Methods for Adaptive Models*. Ph. D. thesis, California Institute of Technology. `wol.ra.phy.cam.ac.uk/mackay/README.html`.

MacKay, D. J. C. (1994a). Bayesian methods for backpropagation networks. In E. Domany, J. L. van Hemmen, and K. Schulten (Eds.), *Models of Neural Networks III*, Chapter 6. New York: Springer-Verlag.

MacKay, D. J. C. (1994b). Bayesian non-linear modeling for the energy prediction competition. *ASHRAE Transactions 100*(2), 1053–1062.

Mangasarian, O. L. and W. H. Wolberg (1990). Cancer diagnosis via linear programming. *SIAM News 23*(5), 1–18.

Marcus, M. and H. Minc (1992). *A Survey of Matrix Theory and Matrix Inequalities*. New York: Dover Publications.

Mardia, K. V., J. Kent, and J. M. Bibby (1979). *Multivariate Analysis*. London: Academic Press.

Mason, L., J. Baxter, P. Bartlett, and M. Frean (1999). Boosting algorithms as gradient descent in function space. Technical report, Australian National University, Canberra, Australia. `wwwsyseng.anu.edu.au/~jon/papers/doom2.ps.gz`.

McCullagh, P. and J. A. Nelder (1989). *Generalized Linear Models* (2nd ed.). London: Chapman and Hall.

McLachlan, G. J. and K. E. Basford (1988). *Mixture Models: Inference and Applications to Clustering*. New York: Marcel Dekker, Inc.

Meinicke, P. and H. Ritter (1999). Local PCA learning with resolution-dependent mixtures of Gaussians. In *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN'99)*, pp. 497–502. London: IEE. Extended version: `http://www.techfak.uni-bielefeld.de/gk/papers/meinick99.html`.

Michie, D., D. J. Spiegelhalter, and C. C. Taylor (Eds.) (1994). *Machine Learning, Neural and Statistical Classification*. New York: Ellis Horwood.

Mika, S. (1998, November). Kernalgorithmen zur nichtlinearen Signalverarbeitung in Merkmalsräumen. Master's thesis, Technische Universität Berlin. `http://www.first.gmd.de/~mika/diplom.ps.gz`.

Mika, S., G. Rätsch, J. Weston, B. Schölkopf, and K.-R. Müller (1999). Fisher discriminant analysis with kernels. In Y.-H. Hu, J. Larsen, E. Wilson, and S. Douglas (Eds.), *Neural Networks for Signal Processing IX*, pp. 41–48. IEEE.

Mitchell, T. (1997). *Machine Learning*. McGraw Hill.

Moerland, P. (1997a). Mixtures of experts estimate a posteriori probabilities. In W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud (Eds.), *Proceedings of the International Conference on Artificial Neural Networks (ICANN'97)*, Number 1327 in Lecture Notes in Computer Science, pp. 499–504. Berlin: Springer-Verlag.

Moerland, P. (1997b). Mixtures of experts estimate a posteriori probabilities. IDIAP-RR 97-07, IDIAP, Martigny, Switzerland, `ftp://ftp.idiap.ch/pub/reports/1997/rr97-07.ps.gz`.

Moerland, P. (1997c, November). Some methods for training mixtures of experts. IDIAP-Com 97-05, IDIAP, `ftp://ftp.idiap.ch/pub/reports/1997/com97-05.ps.gz`.

Moerland, P. (1998). Localized mixtures of experts. IDIAP-RR 98-14, IDIAP, `ftp://ftp.idiap.ch/pub/reports/1998/rr98-14.ps.gz`.

Moerland, P. (1999a). Classification using localized mixtures of experts. In *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN'99)*, Volume 2, pp. 838–843. London: IEE.

Moerland, P. (1999b). A comparison of mixture models for density estimation. In *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN'99)*, Volume 1, pp. 25–30. London: IEE.

Moerland, P. and E. Mayoraz (1999). Dynaboost: Combining boosted hypotheses in a dynamic way. IDIAP-RR 9, IDIAP. Submitted for publication: `ftp://ftp.idiap.ch/pub/reports/1999/rr99-09.ps.gz`.

Møller, M. (1993). A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks 6*(4), 525–533.

Moody, J. and C. J. Darken (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation 1*, 281–294.

Nabney, I. T. (1999). Efficient training of RBF networks for classification. In *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN'99)*, Volume 1, pp. 210–215. London: IEE.

Neal, R. M. (1996). *Bayesian Learning for Neural Networks*. Number 118 in Lecture Notes in Statistics. New York: Springer-Verlag.

Neal, R. M. (1998). Assessing relevance determination methods using Delve. In C. M. Bishop (Ed.), *Generalization in Neural Networks and Machine Learning*. Springer-Verlag.

Neal, R. M. and P. Dayan (1997). Factor analysis using delta-rule wake-sleep learning. *Neural Computation 9*, 1781–1803.

Neal, R. M. and G. E. Hinton (1999). A view of the EM algorithm that justifies incremental, sparse and other variants. See Jordan (1999), pp. 355–368.

Nilsson, N. J. (1965). *Learning Machines*. New York: McGraw-Hill.

Nowlan, S. J. (1991). *Soft Competitive Adaptation*. Ph. D. thesis, Carnegie Mellon University, School of Computer Science, CMU, Pittsburgh, PA.

Oja, E. (1982). A simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology 15*, 267–273.

Ormoneit, D. and V. Tresp (1998). Averaging, maximum penalized likelihood and Bayesian estimation for improving Gaussian mixture probability density estimates. *IEEE Transactions on Neural Networks 9*(4), 639–650.

Papoulis, A. (1991). *Probability, Random Variables, and Stochastic Processes* (3rd ed.). New York: McGraw-Hill.

Pearson, K. (1894). Contribution to the mathematical theory of evolution. *Philosophical Transactions of the Royal Society A*(185), 71–110.

Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery (1992). *Numerical Recipes in C: the Art of Scientific Computing* (2nd ed.). Cambridge: Cambridge University Press.

Quinlan, J. R. (1996). Bagging, boosting and C4.5. In *Proceedings of the Thirteenth National on Artificial Intelligence*, pp. 725–730. Cambridge, MA: AAAI Press/MIT Press.

Ramamurti, V. and J. Ghosh (1999, January). Structurally adaptive modular networks for nonstationary environments. *IEEE Transactions on Neural Networks 10*(1), 152–161.

Rasmussen, C. E., R. M. Neal, G. E. Hinton, D. van Camp, M. Revow, Z. Ghahramani, R. Kustra, and R. Tibshirani (1996). The DELVE manual (version 1.1). Technical report, University of Toronto, `www.cs.utoronto.ca/~delve`.

Rätsch, G., T. Onoda, and K. R. Müller (1998). Soft margins for AdaBoost. *Machine Learning*. To appear: `www.first.gmd.de/~raetsch/ps/Neurocolt_Margin.ps.gz`.

Richard, M. D. and R. P. Lippmann (1991). Neural network classifiers estimate Bayesian a posteriori probabilities. *Neural Computation 3*, 461–483.

Rida, A., A. Labbi, and C. Pellegrini (1999). Local experts combination through density decomposition. In D. Heckerman and J. Whittaker (Eds.), *Proceedings of the Seventh International Workshop on Artificial Intelligence and Statistics*. San Francisco, CA: Morgan Kaufmann Publishers.

Ripley, B. D. (1996). *Pattern recognition and Neural networks*. Cambridge, UK: Cambridge University Press.

Roweis, S. (1998). EM algorithms for PCA and SPCA. In M. I. Jordan, M. J. Kearns, and S. A. Solla (Eds.), *Advances in Neural Information Processing Systems*, Volume 10, pp. 626–632. Cambridge, MA: MIT Press.

Roweis, S. and Z. Ghahramani (1999). A unifying review of linear Gaussian models. *Neural Computation 11*(2), 305–345.

Rubin, D. B. and D. T. Thayer (1982). EM algorithms for ML factor analysis. *Psychometrika 47*(1), 69–76.

Ruck, D. W., S. K. Rogers, M. Kabrisky, M. E. Oxley, and B. W. Suter (1990, December). The multilayer perceptron as an approximation to a Bayes optimal discriminant function. *IEEE Transactions on Neural Networks 1*(4), 296–298.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland, and the PDP Research Group (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Volume 1 : Foundations, Chapter 8, pp. 318–362. Cambridge, MA: MIT Press.

Sanger, T. D. (1989). Optimal unsupervised learning in a single-layer linear feedforward network. *Neural Networks 2*, 459–473.

Sato, M. and S. Ishii (2000). On-line EM algorithm for the normalized Gaussian network. *Neural Computation 12*(2), 407–432.

Saul, L. and M. Rahim (1998). Modeling acoustic correlations by factor analysis. In M. I. Jordan, M. J. Kearns, and S. A. Solla (Eds.), *Advances in Neural Information Processing Systems*, Volume 10, pp. 749–755. Cambridge, MA: MIT Press.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning 5*(2), 197–227.

Schapire, R. E., Y. Freund, P. Bartlett, and W. S. Lee (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics 26*(5), 1651–1686.

Schölkopf, B., A. Smola, and K.-R. Müller (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation 10*(5), 1299–1319.

Schölkopf, B., S. Mika, C. Burges, P. Knirsch, K.-R. Müller, G. Rätsch, and A. Smola (1999). Input space vs. feature space in kernel-based methods. *IEEE Transactions on Neural Networks 10*(5), 1000–1017.

Schwenk, H. and Y. Bengio (1998). Training methods for adaptive boosting of neural networks. In M. I. Jordan, M. J. Kearns, and S. A. Solla (Eds.), *Advances in Neural Information Processing Systems*, Volume 10, pp. 647–653. Cambridge, MA: MIT Press.

Smola, A. J., O. L. Mangasarian, and B. Schölkopf (1999). Sparse kernel feature analysis. Technical Report 99-03, University of Wisconsin, Data Mining Institute, Madison.

Thiesson, B., C. Meek, and D. Heckerman (1999, May). Accelerating EM for large databases. Technical Report MSR-TR-99-31, Microsoft Research. `ftp://ftp.research.microsoft.com/pub/tr/tr-99-31.ps`.

Tipping, M. E. (1999). Probabilistic visualisation of high-dimensional binary data. In M. S. Kearns, S. A. Solla, and D. A. Cohn (Eds.), *Advances in Neural Information Processing Systems*, Volume 11, pp. 592–598. Cambridge, MA: MIT Press.

Tipping, M. E. (2000). The relevance vector machine. In S. A. Solla, T. K. Leen, and K.-R. Müller (Eds.), *Advances in Neural Information Processing Systems*, Volume 12, pp. 652–658. Cambridge, MA: MIT Press.

Tipping, M. E. and C. M. Bishop (1999, February). Mixtures of probabilistic principal component analysers. *Neural Computation 11*(2), 443–482.

Titterington, D., A. F. M. Smith, and U. E. Makov (1985). *Statistical Analysis of Finite Mixture Distributions*. Chichester: John Wiley & Sons.

Turk, M. A. and A. P. Pentland (1991). Eigenfaces for recognition. *Journal of Cognitive Neuroscience 3*(1), 71–86.

Ueda, N., R. Nakano, Z. Ghahramani, and G. E. Hinton (1999). SMEM algorithm for mixture models. In M. S. Kearns, S. A. Solla, and D. A. Cohn (Eds.), *Advances in Neural Information Processing Systems*, Volume 11, pp. 599–605. Cambridge, MA: MIT Press.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. New York: Springer-Verlag.

Wan, E. A. (1990, December). Neural network classification: A Bayesian interpretation. *IEEE Transactions on Neural Networks 1*(4), 303–305.

Waterhouse, S. R. (1997, October). *Classification and Regression Using Mixtures of Experts*. Ph. D. thesis, Cambridge University Engineering Department. `svr-www.eng.cam.ac.uk/~srw1001/`.

Waterhouse, S. R. and G. D. Cook (1997). Ensemble methods for phoneme classification. In M. C. Mozer, M. I. Jordan, and T. Petsche (Eds.), *Advances in Neural Information Processing Systems*, Volume 9, pp. 800–806. Cambridge, MA: MIT Press.

Waterhouse, S. R., D. J. C. MacKay, and A. J. Robinson (1996). Bayesian methods for mixtures of experts. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo (Eds.), *Advances in Neural Information Processing Systems*, Volume 8, pp. 351–357. Cambridge, MA: MIT Press.

Weigend, A. S., M. Mangeas, and A. N. Srivastava (1995). Nonlinear gated experts for time series: Discovering regimes and avoiding overfitting. *International Journal of Neural Systems 6*, 373–399.

Xu, L., M. I. Jordan, and G. E. Hinton (1995). An alternative model for mixtures of experts. In G. Tesauro, D. S. Touretzky, and T. K. Leen (Eds.), *Advances in Neural Information Processing Systems*, Volume 7, pp. 633–640. Cambridge, MA: MIT Press.

Zeevi, A., R. Meir, and V. Maiorov (1998). Error bounds for functional approximation and estimation using mixtures of experts. *IEEE Transactions on Information Theory 44*, 1010–1025.

Index

Curriculum Vitae

Personal Data

Name:	Pieter Daniël Moerland
Date of birth:	September 9, 1969
Place of birth:	Oud-Vossemeer, the Netherlands

Education

September '81 - June '87: Gymnasium β (preparing for university entrance in the exact sciences) at the Mollerlyceum in Bergen op Zoom, the Netherlands.

September '87 - February '93: Engineering degree in Computing Science (with highest distinction) from the Eindhoven University of Technology.

October'93 - March'94: Postgraduate course in discrete mathematics, information theory, cryptology, and error-correcting codes at the Eindhoven University of Technology.

Professional experience

May '93 - April '94: Research assistant (as a fulfilment of my social service) at the discrete mathematics group, Department of Mathematics and Computing Science, Eindhoven University of Technology.

May '94 - April '96 Research engineer at the Dalle Molle Institute of Perceptual Artificial Intelligence (IDIAP) in Martigny, Switzerland.

May '96 - now Working towards Ph.D. degree at IDIAP in Martigny, Switzerland.

Publications

Book chapters

P. Moerland and E. Fiesler. Neural network adaptations to hardware implementations. In E. Fiesler and R. Beale, editors, *Handbook of Neural Computation*, pages E1.2:1–13. Institute of Physics Publishing and Oxford University Press, Bristol and Oxford, 1997.

International journals

P. Moerland, E. Fiesler, and I. Saxena. Discrete all-positive multilayer perceptrons for optical implementation. *Optical Engineering*, 37(4):1305–1315, 1998.

P. Moerland, E. Fiesler, and I. Saxena. Incorporation of liquid-crystal light valve non-linearities in optical multilayer neural networks. *Applied Optics*, 35(26):5301–5307, 1996.

P. Moerland. A review of MicroNeuro'96, February 12-14, 1996, Lausanne, Switzerland. *Neurocomputing*, 12(4):371–373, 1996.

G. Thimm, P. Moerland, and E. Fiesler. The interchangeability of learning rate and gain in backpropagation neural networks. *Neural Computation*, 8(2):451–460, 1996.

International conferences

P. Moerland. A comparison of mixture models for density estimation. In *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN'99)*, volume 1, pages 25–30. IEE, London, 1999.

P. Moerland. Classification using localized mixtures of experts. In *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN'99)*, volume 2, pages 838–843. IEE, London, 1999.

P. Moerland. Mixtures of experts estimate a posteriori probabilities. In W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, editors, *Proceedings of the International Conference on Artificial Neural Networks (ICANN'97)*, number 1327 in Lecture Notes in Computer Science, pages 499–504. Springer-Verlag, Berlin, 1997.

I. Saxena, P. Moerland, E. Fiesler, and A. Pourzand. Handwritten digit recognition with binary optical perceptron. In W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, editors, *Proceedings of the International Conference on Artificial Neural Networks (ICANN'97)*, number 1327 in Lecture Notes in Computer Science, pages 1253–1258. Springer-Verlag, Berlin, 1997.

I. Saxena, P. Moerland, E. Fiesler, A. R. Pourzand, and N. Collings. An optical thresholding perceptron. In *Proceedings of the 11th IPPS Symposium Workshop: Optics and Computer Science*, 1997.

I. Saxena, E. Fiesler, and P. Moerland. A method for all-positive optical multilayer perceptrons. In *Proceedings of the Third IEEE International Conference on Electronics, Circuits, and Systems*, volume 1, pages 448–451. IEEE, Piscataway, NJ, 1996.

P. Moerland and E. Fiesler. Hardware-friendly learning algorithms for neural networks: An overview. In *Proceedings of the Fifth International Conference on Microelectronics for Neural Networks and Fuzzy Systems: MicroNeuro'96*, pages 117–124. IEEE Computer Society Press, Los Alamitos, CA, 1996.

G. Thimm, E. Fiesler, and P. Moerland. Gain elimination form backpropagation neural networks. In *Proceedings of the International Conference on Neural Networks*, volume 1, pages 365–368. IEEE, Piscataway, NJ, 1995.

P. Moerland, E. Fiesler, and I. Saxena. The effects of optical thresholding in backpropagation neural networks. In F. Fogelman-Soulié and P. Gallinari, editors, *Proceedings of the International Conference on Artificial Neural Networks (ICANN'95 and NeuroNîmes'95)*, volume 2, pages 339–343. EC2 & Cie, Paris La Défense, France, 1995.

Other publications

T. Lundin, E. Fiesler, and P. Moerland. Connectionist quantization functions. In *Proceedings of the '96 SIPAR-Workshop on Parallel and Distributed Computing*, pages 33–36. Scientific and Parallel Computing Group, University of Geneva, Geneva, Switzerland, 1996.

P. Moerland, E. Fiesler, and I. Saxena. Overcoming inaccuracies in optical multilayer perceptrons. In *Proceedings of the First International Symposium on Neuro-Fuzzy Systems (AT'96)*, pages 55–62. AATI, Lausanne, Switzerland, 1996.

P. Moerland, G. Thimm, and E. Fiesler. Results on the steepness in backpropagation neural networks. In Marc Aguilar, editor, *Proceedings of the '94 SIPAR-Workshop on Parallel and*

Distributed Computing, pages 91–94, Institute of Informatics, University Pérolles, Fribourg, Switzerland, 1994.

Technical reports and submitted papers

P. Moerland. Mixtures of latent variable models for density estimation and classification. Submitted for publication, May 2000.

P. Moerland and E. Mayoraz. Dynaboost: Combining boosted hypotheses in a dynamic way. IDIAP-RR 99-09, IDIAP, 1999. `ftp://ftp.idiap.ch/pub/reports/1999/rr99-09.ps.gz`.

P. Moerland. Localized mixtures of experts. IDIAP-RR 98-14, IDIAP, 1998. `ftp://ftp.idiap.ch/pub/reports/1998/rr98-14.ps.gz`.

P. Moerland. Some methods for training mixtures of experts. IDIAP-Com 97-05, IDIAP, November 1997. `ftp://ftp.idiap.ch/pub/reports/1997/com97-05.ps.gz`.

T. Lundin and P. Moerland. Quantization and pruning of multilayer perceptrons: Towards compact neural networks. IDIAP-Com 97-02, IDIAP, March 1997. `ftp://ftp.idiap.ch/pub/reports/1997/com97-02.ps.gz`.

P. Moerland. Exercises in multiprogramming. Computing Science Note 93/07, Dept. of Math. and Comp. Science, Eindhoven University of Technology, The Netherlands, 1993.