Thesis Submitted in Partial Fulfilment of the Requirements for the
Degree of Diplom Informatiker

A Neural Architecture for Learning Uncertainty

Hauke Tschach

Supervisor: Dr. Christian Perwass

Christian-Albrechts-University of Kiel
Faculty of Engineering
Institute of Computer Science and Applied Mathematics
Chair of Cognitive Systems
Prof. Dr. Gerald Sommer

November 2005

Abstract

This text presents a novel architecture for binary classification. The idea is the combination of the Gauss-Helmert model with Fisher's linear discriminant/least squares techniques. The result of this combination is a linear classifier that is able to use prior knowledge in the form of information about the uncertainty of the input data points for the learning of the decision boundary. This uncertainty information is usually given in the form of a covariance matrix and it is possible to use different covariance matrices for different points. The use of error propagation enables this system not only to use uncertainty information for the learning, it also results in uncertainty information for the decision boundary. The uncertainty information of the decision boundary can be used to calculate a measure of confidence for the classification result for a new point. Alternatively it can be used by a statistical test to determine whether a new point can be classified with sufficient reliability or not. Besides the linear classifier, an extension to nonlinear classification is presented too. This extension is based on the ideas of radial basis function (RBF) networks. Some initial experiments giving a proof of concept are reported as well.

Contents

Chapter 1

Introduction

As humans we are making many decisions everyday. Many of these are based on observations. This means that we gain some knowledge from our environment and make decisions based on this knowledge. This can be decisions which we recognise as such like the decision of taking an umbrella with us when the sky is grey or even more serious decisions where it may take us several days to reach a decision.

Although these are the kind of decisions recognised by us as such, we are in fact making a far greater number of decisions without thinking of it. When riding a bicycle, for example, we are constantly turning the handlebar in order not to fall over to one side. When reading e.g. numbers, we are constantly looking at pictures of digits deciding which of the ten digits of our decimal system is represented by the current picture.

In order to enable machines to do many jobs formerly done by humans, we must enable the machines to make decisions. Making a decision can be seen as getting a portion of – possibly noisy – data and deciding to which of two or more classes the data belongs and subsequently reporting the decision or taking some more sophisticated action. This process is called pattern recognition (*'the act of taking in raw data and taking an action based on the "category" of the pattern'*, [8]).

Since computers have been used for pattern classification for several decades now, there are many different algorithms which have been developed over the years. These range from relatively simple algorithms – like a perceptron, Fisher's linear discriminant, a support vector machine (SVM) with a linear kernel, etc. – that implement linear decision functions, up to very sophisticated algorithms – like multi-layer perceptrons, radial basis function (RBF) networks, SVMs with non-linear kernels, etc. – that can learn virtually every imaginable decision function as long as the training is started with the correct parameters and there is enough training data available.

Since the available training data is often contaminated with noise, most of these algorithms do not try to learn the training data perfectly. Instead they try to learn a smooth decision surface where the degree of smoothness

usually has to be controlled by the user. The smoothing has the advantage that the resulting decision surface will not be mislead by a few outliers and often the amount of training data needed is also reduced. However, although most of these pattern classification algorithms assume some noise on the input data, most of them do not take into account a possibly known distribution of this noise.

It appears as if only recently there have evolved some algorithms that take into account information of different variances for different data points. These are mainly the algorithms presented in [2] and [1]. They make use of the training data uncertainty information for learning the decision boundary. However they do not supply the user with information about the confidence of a classification decision. Another algorithm that could be interpreted as using information of different uncertainties on the two classes, but the same uncertainty for all points from one class, is the perceptron with uneven margins, see [23], although its intended purpose is to cope with uneven data sets.

The architecture proposed in this text will take into account uncertainty information of the training data not only for learning the decision boundary but also for calculating a variance for the parameters of the decision boundary. This can then be used to calculate the probability for a new point to lie on one side or the other of a decision boundary drawn from the distribution of the decision boundaries (given by the parameters and its variances). This probability can be seen as a measure of confidence for the classification result.

This text is organised as follows: in the second chapter we will briefly review some important traditional approaches for linear separation of data. In the following chapter we will introduce the Gauss-Helmert model for line fitting before we present its adaptation for the task of linear discrimination in Chapter 4. This is the main part of this text. It follows a review of some traditional architectures for non-linear discrimination before we present an adaptation of our approach using RBF nodes. In Chapter 7 we show a possibility to use a statistical test to determine whether a pattern can be classified with sufficient reliability or not. It follows a short presentation of some other recent architectures that make use of known training data uncertainty before we arrive at the final chapter of this text summarising the benefits and problems of the presented approach.

— ∗ —

Chapter 2

Classic neural architectures for linear discrimination

The task of a pattern recognition system is to assign a previously unknown pattern x to one of c classes $C_1, \ldots, C_c$. If this is done consistently, this corresponds to partitioning the input space into c parts $R_1, \ldots, R_c$, each region R_j corresponding to class C_j. A new pattern is now assigned to that class that the region containing the pattern corresponds to.

Since many multi-class discrimination algorithms are based on binary discrimination algorithms and for the ease of understanding, in this text we will only consider binary classification problems with two classes C_1 and C_2 which we will also call C_+ and C_-. Therefore we will often use the terms positive and negative examples to denote the elements of C_1 and C_2. In most examples we even go a step further and assume that both classes have the same prior probability, i.e. $P(C_1) = P(C_2) = 0.5$. The prior probability is the probability for an unobserved sample to belong to a class C_i. This is approximately equal to the fraction of patterns belonging to class C_i in a sufficiently large sample of training data.

For the reader interested in multi-class discrimination, the literature (e.g. [8]) might be an aid to find ways of extending the presented algorithms.

2.1 Notation

In this text we will, if not stated otherwise, use small bold letters (e.g. $\mathbf{x}^i, \mathbf{w}$) for column vectors and capital bold letters for matrices. Non-bold characters usually denote scalars. The transpose of a vector or matrix will be written as $\mathbf{w}^\top = (w_1, \ldots, w_d)$ or $\mathbf{A}^\top$. Probabilities will be written as P and for probability densities we will use p. In addition we define the signum function as

$$\text{sgn} : x \mapsto \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{otherwise.} \end{cases}$$

We will use $X \subset \mathbb{R}^d$ to denote the input space and Y to denote the output domain. If not stated otherwise, we assume that $Y = \{-1, +1\}$. The training of our algorithms will be done using batch learning. For this we will use a finite training set $S = \{(\mathbf{x}^1, y_1), \ldots, (\mathbf{x}^N, y_N)\} \subset X \times Y$ with labels y_i to denote the desired output for an example $\mathbf{x}^i$, i.e. $y_i = +1$ meaning that $\mathbf{x}^i$ should be assigned to class C_+. The dimension of the input space is d, the size of the training set is N, the number of training vectors with the desired output i is N_i (i.e. $N_+ = N_1$ denotes the number of positive examples in the training set). If $d = 1$, we will write x^i instead of $\mathbf{x}^i$. A new example which is not part of the training set will be written as $\mathbf{x}$ or x. If there is no danger of confusing indices of vectors and indices of vector entries, we will also use the more common notation $\mathbf{x}_i$ instead of $\mathbf{x}^i$.

2.2 Why linear classification does make sense

A misclassification occurs each time an example belonging to class C_i is classified as belonging to class $C_j, j \neq i$. If all examples lying in a region R_1 are classified as positive examples and all examples lying in $R_2 = X \setminus R_1$ as negative examples, we can specify the probability of misclassifying a new example as

$$\begin{aligned} P(\text{error}) &= P(\mathbf{x} \in R_2, C_1) + P(\mathbf{x} \in R_1, C_2) \\ &= \int_{R_2} p(\mathbf{x}|C_1)P(C_1)d\mathbf{x} + \int_{R_1} p(\mathbf{x}|C_2)P(C_2)d\mathbf{x}, \end{aligned}$$

where $P(\mathbf{x} \in R_i, C_j)$ describes the probability that a pattern belongs to class C_j and lies in region R_i.

If our observations are one-dimensional and the regions R_1 and R_2 are specified as e.g. $R_1 = \{x|x >= b\}$ and $R_2 = \{x|x < b\}$, we can rewrite the probability of misclassification as

$$P(\text{error}) = \int_{-\infty}^{b} p(x|C_1)P(C_1)dx + \int_{b}^{\infty} p(x|C_2)P(C_2)dx. \tag{2.1}$$

In order to minimise the number of misclassified examples, each example x has to be assigned to the class C_i whose joint probability density $p(x, C_i)$ is maximal for x. The decision boundaries therefore have to lie where the joint probability densities for both classes are equal. In the one-dimensional case the decision boundary has to lie where the curves of the joint probability densities for C_- and C_+ cross. This is illustrated in Figure 2.1 for normal distributed data.

Since the joint probability densities can be written as

$$p(x, C_i) = P(C_i|x)p(x), \tag{2.2}$$

with the factor $p(x)$ being independent of the class, we get the same classification if we assign each pattern to the class whose posterior probability $P(C_i|x)$ is maximal as illustrated in Figure 2.2.

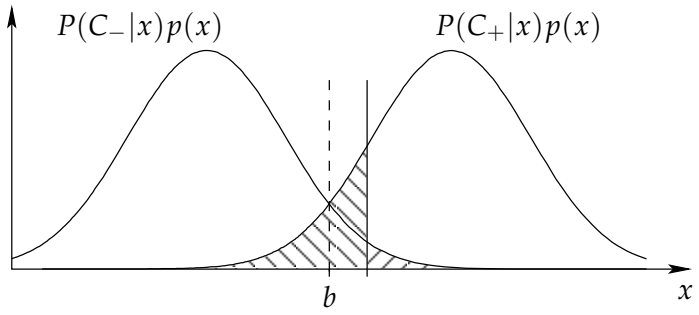

Figure 2.1: Given the joint probability density functions for two classes, the hatched areas show the classification error when the place indicated by the solid vertical line would be used as the boundary between the two classes. Using b for this boundary instead, illustrated by the dashed line, would minimise this error.

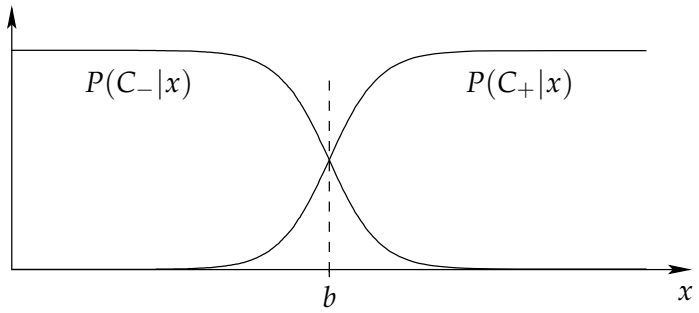

Figure 2.2: Using the point of equal posterior probabilities leads to the same boundary as using the point of equal joint probability density functions as in Figure 2.1.

If both classes are multivariate normal distributed with the same covariance matrices, and both have the same prior probability, the optimal decision boundary will be a hyperplane. Figure 2.3 illustrates this for the two-dimensional case.

Since the exact distributions of the two classes are often unknown, it is convenient to assume that they are normal distributions with the same covariance matrices. As mentioned above, this naturally leads to a linear separation of the input space. A disadvantage of linear discrimination is its inability to accurately discriminate between classes that are not linearly separable. However, for many neural architectures that learn a linear discrimination of the input space, this simplicity can also be an advantage. This is because it reduces the risk of over-fitting and keeps the number of tuning parameters, which have to be chosen by the user, to a minimum, making it even more robust against improper use. This robustness and simplicity of linear separation together with the fact that it is used in more complex algorithms like multi layer perceptrons (MLP), radial basis func-

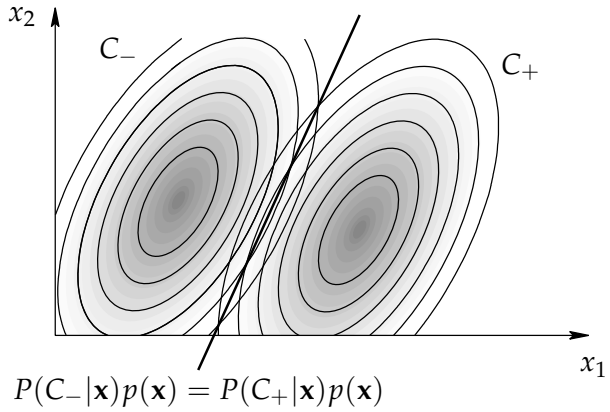

Figure 2.3: The plot shows the distributions of data belonging to two classes C_+ and C_-. The ellipses show lines of same joint probability density. The solid line runs through the points where both joint probability densities are equal and therefore shows the best possible decision boundary realising the smallest classification error.

tion networks (RBF networks) or support vector machines (SVM) explains its importance.

To be able to linearly discriminate between two classes one first has to calculate a separating hyperplane. For this task there exist a variety of algorithms of which we will present some in the remainder of this chapter.

2.3 The perceptron

The perceptron algorithm is one of the simplest algorithms for learning a separating hyperplane. Due to its simplicity it is easy to understand and to implement. Its main deficiencies are the facts that the algorithm runs into an infinite loop if the training data is not linearly separable and that the final hyperplane depends on the order of the training data and a possibly random initialisation of the weight vector. However, if the training set is separable it will always find a solution (for a proof see e.g. the proof of Theorem 5.1 in [8]).

The idea of the perceptron algorithm is to start with an initial weight vector $\mathbf{w}_0$, go through a finite training set, and update the weight vector each time a training vector is misclassified. Once the end of the training set is reached, the process is restarted at the beginning of the set, updating the weight vector further. This process is continued until all training vectors are classified correctly, i.e. it will never end if the training data is not linearly separable.

The update process can be expressed as

$$\mathbf{w}_{t+1} \leftarrow \begin{cases} \mathbf{w}_t + \eta y_i \mathbf{x}^i & \text{if } \operatorname{sgn}(\mathbf{w}_t^\top \mathbf{x}^i) \neq y_i, \\ \mathbf{w}_t & \text{otherwise,} \end{cases} \tag{2.3}$$

where $\eta \in \mathbb{R}_{>0}$ is called the learning rate. The learning of two positive training samples is illustrated in Figure 2.4.

Note that the algorithm as presented above will only learn hyperplanes going through the origin. To learn arbitrary hyperplanes with a distance to the origin, it is necessary to learn an additional parameter b specifying the distance to the origin. This can be done by adding an additional component $x_0 = 1$ to each (training) vector. The w_0 component of the new weight vector can now be seen as specifying the distance of the separating hyperplane to the origin. The Euclidean distance to the origin can be calculated as $|-w_0/\|(w_1,\ldots,w_d)^\top\|_2|$.

Variants

Its introduction being about half a century ago, there are many variants of the perceptron algorithm, which have been developed over the decades.

One variation is the random selection of patterns from the training set. This would make the resulting classifier independent of the order of the training set. On the other hand, several training sessions can result in different classifiers. This can be a disadvantage, since the training is not reproducible, or an advantage, since one could train several times and choose the best of the resulting classifiers.

Another possible variant is the use of a varying learning rate with the most popular choice being a decreasing learning rate resulting in smaller updates as the training progresses.

A variation able to deal with not linearly separable training sets would be to stop the training process when the number of misclassified training examples has decreased to a non-zero value chosen by the user.

A further possibility is the introduction of margins, i.e. a pattern $\mathbf{x}^i$ is counted as misclassified not only if $\operatorname{sgn}(\mathbf{w}^\top\mathbf{x}^i) \neq y_i$, which is equivalent to $y_i\mathbf{w}^\top\mathbf{x}^i < 0$, but even if $y_i\mathbf{w}^\top\mathbf{x}^i < \tau$ with $\tau \in \mathbb{R}$ being the margin. If $\tau < 0$, it can become possible to train with a not linearly separable training set as long as it can be made linearly separable by moving the two classes by at most $2|\tau|$ apart. If $\tau > 0$, each point will have a Euclidean distance to the separating hyperplane of at least $\tau/\|\mathbf{w}\|_2$. If there is more noise on the points from one class than on the points from the other class or if the training set is very unbalanced, it is also possible to choose different margins τ_+/τ_- for the different classes as proposed in [23].

2.4 Least-squares

Unlike the perceptron, where a possible error function would count the number of misclassified examples and would therefore be discontinuous, least-squares techniques rely on a continuously differentiable error func-

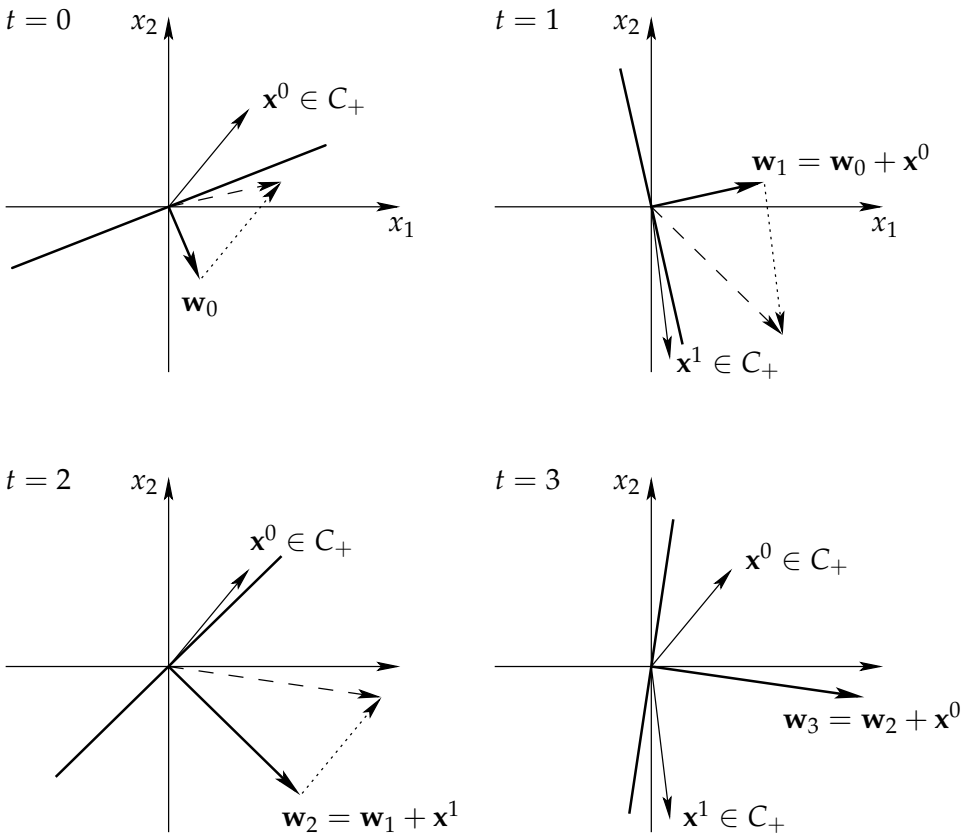

Figure 2.4: Perceptron learning starting with a weight vector $\mathbf{w}_0$ and a training set consisting of two positive examples $\mathbf{x}^0$ and $\mathbf{x}^1$. The first training vector $\mathbf{x}^0$ is mistakenly classified as negative. Therefore the weight vector is updated to $\mathbf{w}_1 = \mathbf{w}_0 + \mathbf{x}^0$. Now the second vector $\mathbf{x}^1$ is misclassified such that the weight vector is updated to $\mathbf{w}_2 = \mathbf{w}_1 + \mathbf{x}^1$. Due to the misclassification of $\mathbf{x}^0$ using $\mathbf{w}_2$, the weight vector is updated to $\mathbf{w}_3 = \mathbf{w}_2 + \mathbf{x}^0$. Since the hyperplane given by $\mathbf{w}_3$ classifies all training vectors correctly, no further updates are necessary: the solution found by the perceptron algorithm is the hyperplane given by the weight vector $\mathbf{w}_3$.

tion. The sum-of-squares error function

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(\mathbf{w}^\top \mathbf{x^n} - y_n)^2 \tag{2.4}$$

is formed as the sum of the squared differences between the product of the weight vector with a vector $\mathbf{x}^n$ and its target label y_n. Minimisation of the error function gives the weight vector $\mathbf{w}$ which specifies the separating hyperplane.

Geometrically, the search for a weight vector $\mathbf{w}$ using least-squares techniques can be interpreted as finding a hyperplane h given by $\mathbf{w}$ such that the two hyperplanes h_+ and h_-, parallel to h, each with distance $1/||\mathbf{w}||_2$, best fit the training data points from class C_+ (h_+) and C_- (h_-). This means that we try to minimise the squared distances of the training data points to the parallel that corresponds to their class label. Figure 2.5 illustrates this. To learn an additional parameter b, one can use the approach described in the previous section.

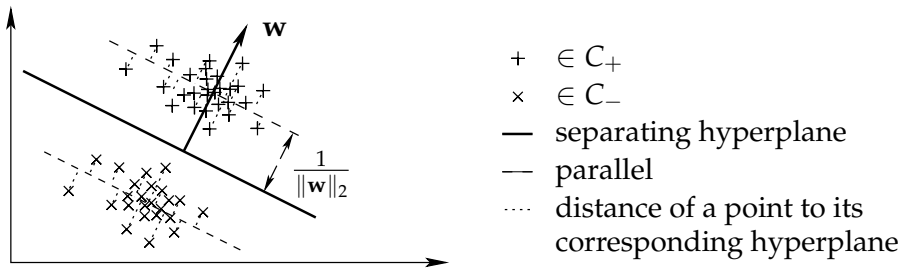

Figure 2.5: Least-squares techniques used for discrimination find a separating hyperplane by fitting two parallels (dashed) through the training data points: one through the points from class C_+ and one through the points from class C_-. The solid parallel in the centre is then used as the separating hyperplane. The distance of the dashed parallels to the separating hyperplane is given by the reciprocal of the norm of $\mathbf{w}$.

2.5 Fisher's linear discriminant

In its original form, Fisher's linear discriminant is not a method for learning a discriminating hyperplane but a method for dimensionality reduction to just one dimension. The idea is to project the data onto a line and to separate the one-dimensional projected data.

For the best possible discrimination one has to find a projection such that the overlap of the joint probability densities of the projected classes is as small as possible (see Figure 2.7) since this corresponds to the error that even the best classifier will make when applied to the projected data (see Figure 2.1).

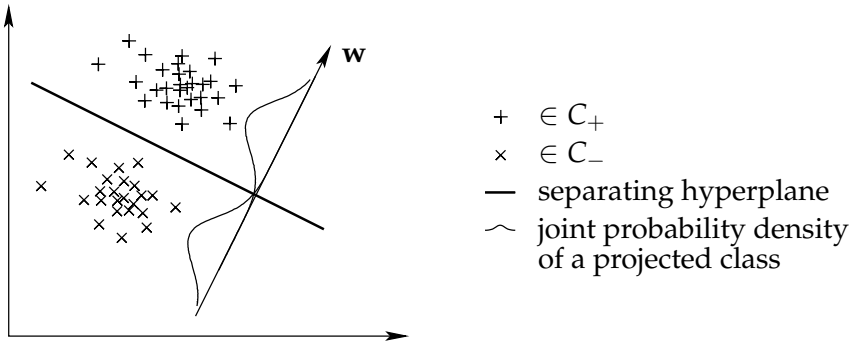

Figure 2.6: Projection of the original data onto a line and discrimination of the one-dimensional projected data.

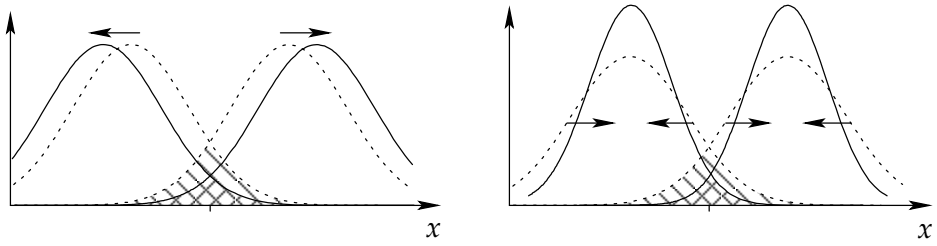

Figure 2.7: Increasing the distance of the projected class means (left) and decreasing the variance of the projected within-class variances (right) both reduces the classification error of the best possible classifier on the one-dimensional projected data.

2.5.1 The Fisher criterion

For finding an appropriate projection, the technique called Fisher's linear discriminant analysis makes use of the Fisher criterion. This is a function of the weight vector $\mathbf{w}$ which has to be maximised in order to find the 'best' estimate for $\mathbf{w}$. The idea is to maximise the distance of the projected class means while at the same time keeping the variances of the projected classes small.

Having a finite training set of labelled data points from two classes with unknown distributions, we can follow [3] or [8] by defining the means of the projected classes as

$$m_1 = \frac{1}{N_1} \sum_{n \in C_1} \mathbf{w}^\top \mathbf{x}_n \quad \text{and} \quad m_2 = \frac{1}{N_2} \sum_{n \in C_2} \mathbf{w}^\top \mathbf{x}_n, \tag{2.5}$$

the within-class scatter as

$$s_i{}^2 = \sum_{n \in C_i} (\mathbf{w}^\top \mathbf{x_n} - m_i)^2, \qquad i \in \{1, 2\}, \tag{2.6}$$

and the Fisher criterion as

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2}. \tag{2.7}$$

The within-class scatter (2.6) is a measure for the within class variance scaled by the size of the class. This means that, if the numbers of the given samples from the two classes differ considerably, the within-class scatter of the larger class has a greater impact to the denominator of the Fisher criterion (2.7) than the scatter of the other class. If this is not desired, one could follow [12] and replace the within-class scatters, which depend on the sizes of the classes, by the within-class variances of the projected data.

Being a method for dimensionality reduction with a function assessing the quality of a projection, Fisher's linear discriminant could also be seen as a projection index for projection pursuit ([11]). In [14] however, the two-sample projection index searching *'for the best discriminating hyperplane in the classical, Fisherian sense'* differs slightly from the definitions above.[1]

2.5.2 Relation to least-squares

Although looking differently and being the result of different ideas, the solutions found by the least-squares approach (Section 2.4) are often identical to the solutions found by Fisher's linear discriminant – provided that the training set is balanced.

This is due to the fact that the Fisher criterion can be seen as a special case of the least-squares approach. The trick is the use of special target values for the least-squares minimisation. If the target values $+1$ and -1 are replaced by N/N_+ and $-N/N_-$, with N_+ and N_- being the sizes of the two classes C_+ and C_-, it can be shown (see e.g. [3] pp 107 – 110) that the least-squares approach results in the same normal vector for the separating hyperplane as Fisher's linear discriminant does. If the number of examples from both classes are equal, the separating hyperplane will be exactly the same as the one found by minimising the sum-of-squares error function (2.4).

This gives an explanation for the identical solutions mentioned above. Given the fact that there are several – at least two – well established techniques for binary discrimination which effectively rely on fitting parallels through the data, we will take up this approach again in Chapter 4 and base our new architecture on this idea.

[1] The main difference is, that the denominator does not contain the sum of the variances of the two classes but the standard deviation of the set of all projected samples (from both classes). The nominator contains the difference of the class averages – not squared as in the definitions above. The index is therefore given as $(m_1 - m_2)/(\mathrm{sdv}\{\mathbf{w}^\top \mathbf{x}_n | \mathbf{x}_n \in C_1 \cup C_2\})$.

2.6 The support vector machine

The support vector machine ([4]) is the most recent approach for linear discrimination presented in this chapter. The idea is to find a separating hyperplane with maximal margin.

The (geometric) margin of a point $\mathbf{x}_n$ with label $y_n \in \{-1, 1\}$, with respect to a hyperplane given by a weight vector $\mathbf{w}$ and a parameter b, is given as

$$\gamma_n = y_n(\mathbf{w}^\top \mathbf{x}_n + b) / \|\mathbf{w}\|_2 \tag{2.8}$$

and therefore measures the Euclidean distance of $\mathbf{x}_n$ to the separating hyperplane if $\mathbf{x}_n$ lies on the correct side and the negative of the Euclidean distance if $\mathbf{x}_n$ lies on the wrong side. The (geometric) margin γ with respect to a training set is now defined as the minimum of the (geometric) margins of all points. Omitting the factor $1/\|\mathbf{w}\|_2$ in equation (2.8) gives the functional margin.

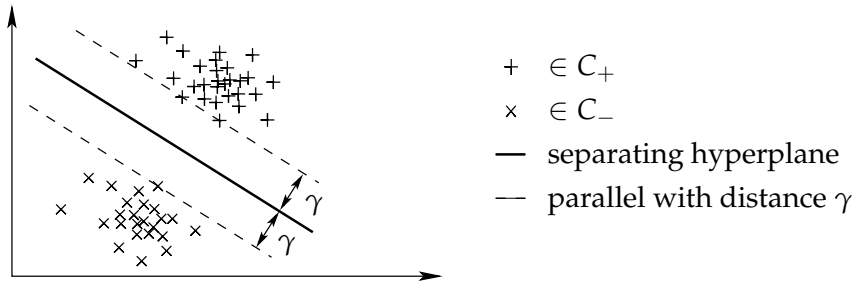

Figure 2.8: The separating hyperplane with maximal (geometric) margin. Rotation or translation of the hyperplane would result in a decrease of the (geometric) margin.

The separating hyperplane is now found by maximising the geometric margin. Since this is equivalent to minimising (the square of) the weight vector while fixing the functional margin to 1, the optimisation problem solved for training a support vector machine can be formulated as

$$\begin{array}{ll} \text{minimise} & \|\mathbf{w}\|_2 \\ \text{subject to} & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \\ & i = 1, \ldots, N. \end{array} \tag{2.9}$$

The main problem of this hard margin support vector machine is the fact that, like the perceptron, it cannot find a solution if the training set is not linearly separable, since this would result in negative margins which cannot satisfy the constraints in the optimisation problem (2.9).

To deal with this problem, there are mainly two approaches: the non-linear projection of the data into a usually higher-dimensional space using a non-linear kernel with the hope of linear separability of the projected data (see Section 5.2), or the introduction of slack variables leading to the

soft margin SVM (see [6]). The problem of the first approach is the increased danger of over-fitting the training set besides the fact that there is no guarantee that the projected data will be linearly separable. The second approach has the drawback that an additional parameter C (see e.g. [7]), which controls the influence of outliers to the solution, has to be tuned.

Chapter 3

The Gauss-Helmert method

The Gauss-Helmert method (for a more general presentation see [27]) can be used as a quite universal tool for estimating the parameters of a model from uncertain observations. For linear problems it gives a best linear unbiased estimate.

3.1 Properties of the Gauss-Helmert estimation

Compared to most other methods for parameter estimation, like the minimisation of an error function in classical least-squares techniques, there are two main differences:

1. the adaptation process does not only adapt the parameters but the observations too,

2. the parameters are estimated together with a covariance matrix describing the uncertainty of the parameters.

A prerequisite for using the Gauss-Helmert model is that the observations are given together with uncertainty information in the form of covariance matrices. Since the estimation is done by iteratively adapting both the parameters *and* the observations, it is also necessary to start with a suitable initial estimate for the parameter vector. Otherwise it can happen that the performance is very poor. Figures 3.3 and 3.4 give an example of such a poor performance.

3.2 The Gauss-Helmert estimation to fit straight lines

To demonstrate how the Gauss-Helmert estimation works, we present the example of fitting a straight line through a set of points in $\mathbb{R}^2$. This does not demonstrate the full potential of the model but shows all properties we will use for our new architecture presented in the next section.

The parameters of our model are represented by a three-dimensional vector $\mathbf{w} = \begin{pmatrix} \mathbf{n} \\ b \end{pmatrix}$ consisting of the normal vector $\mathbf{n}$ and the negative distance b of the line to the origin. When the learning is finished, we will have also estimated a covariance matrix $\mathbf{\Sigma}_{\Delta\mathbf{w},\Delta\mathbf{w}}$ describing the uncertainty of the parameter vector and thus the uncertainty of the line itself.

The observations are given by N two-dimensional vectors $\mathbf{x}_i \in \mathbb{R}^2$ together with N covariance matrices $\mathbf{\Sigma}_{\mathbf{x}_i,\mathbf{x}_i} \in \mathbb{R}^{2\times 2}$.

We group all possible constraints into two types:

1. constraints on the observations and the parameters: $\mathbf{g}_i(\mathbf{x}_i, \mathbf{w}) = \mathbf{0}$,
2. constraints on the parameters alone: $\mathbf{h}(\mathbf{w}) = \mathbf{0}$.

3.2.1 Constraints on the observations and the parameters

The first set of constraints is of the first type and ensures that all points will finally lie on the line:

$$\mathbf{g}_i(\mathbf{x}_i, \mathbf{w}) = \mathbf{n}^\top \mathbf{x}_i + b = 0. \tag{3.1}$$

If we write the observations $\mathbf{x}_i$ as a sum

$$\mathbf{x}_i = \hat{\mathbf{x}}_i + \Delta\mathbf{x}_i$$

of an estimate $\hat{\mathbf{x}}_i$ and an error in the estimate $\Delta\mathbf{x}_i$ and similarly for $\mathbf{w}$ as

$$\mathbf{w} = \hat{\mathbf{w}} + \Delta\mathbf{w},$$

we can rewrite the constraints of type one as

$$\mathbf{g}_i(\hat{\mathbf{x}}_i + \Delta\mathbf{x}_i, \hat{\mathbf{w}} + \Delta\mathbf{w}) = 0$$

and use a Taylor series expansion to approximate them as

$$\mathbf{g}_i(\hat{\mathbf{x}}_i, \hat{\mathbf{w}}) + (\partial_{\mathbf{x}i}\mathbf{g}_i)(\hat{\mathbf{x}}_i, \hat{\mathbf{w}})\Delta\mathbf{x}_i + (\partial_{\mathbf{w}}\mathbf{g}_i)(\hat{\mathbf{x}}_i, \hat{\mathbf{w}})\Delta\mathbf{w} \approx 0. \tag{3.2}$$

Defining

$$\mathbf{c}_{\mathbf{g}_i} := -\mathbf{g}_i(\hat{\mathbf{x}}_i, \hat{\mathbf{w}}), \qquad \mathbf{A} := \begin{pmatrix} (\partial_{\mathbf{w}}\mathbf{g}_1)(\hat{\mathbf{x}}_1, \hat{\mathbf{w}}) \\ \vdots \\ (\partial_{\mathbf{w}}\mathbf{g}_N)(\hat{\mathbf{x}}_N, \hat{\mathbf{w}}) \end{pmatrix},$$

$$\mathbf{B}^\top := \begin{pmatrix} (\partial_{\mathbf{x}_i}\mathbf{g}_1)(\hat{\mathbf{x}}_1, \hat{\mathbf{w}}) & & 0 \\ & \ddots & \\ 0 & & (\partial_{\mathbf{w}}\mathbf{g}_N)(\hat{\mathbf{x}}_N, \hat{\mathbf{w}}) \end{pmatrix}, \text{ and } \Delta\mathbf{x} := \begin{pmatrix} \Delta\mathbf{x}_1 \\ \vdots \\ \Delta\mathbf{x}_N \end{pmatrix},$$

we can rewrite the equations from (3.2) as

$$\mathbf{A}\,\Delta\mathbf{w} + \mathbf{B}^\top\,\Delta\mathbf{x} = \mathbf{c}_{\mathbf{g}}. \tag{3.3}$$

3.2.2 Constraint on the parameters alone

The last constraint fixes the norm of the normal vector $\mathbf{n}$ to one:

$$\mathbf{h}(\mathbf{w}) = \frac{1}{2}(\mathbf{n}^\top \mathbf{n} - 1) = 0 \tag{3.4}$$

$$\Longleftrightarrow \mathbf{h}(\hat{\mathbf{w}}) + (\partial_{\mathbf{w}}\mathbf{h})(\hat{\mathbf{w}})\Delta\mathbf{w} \approx 0.$$

Defining

$$\mathbf{H}^\top(\hat{\mathbf{w}}) := (\partial_{\mathbf{w}}\mathbf{h})(\hat{\mathbf{w}}) \quad \text{and} \quad \mathbf{c}_\mathbf{h} := -\mathbf{h}(\hat{\mathbf{w}}),$$

this constraint can be rewritten as

$$\mathbf{H}^\top(\hat{\mathbf{w}})\,\Delta\mathbf{w} = \mathbf{c}_\mathbf{h}. \tag{3.5}$$

3.2.3 The minimisation

Starting with the given observations $\mathbf{x}_i$ and an initial parameter vector $\mathbf{w}$, the new parameter vector is calculated by minimising the adaptations of the observations, weighted by their inverse covariance matrices, observing the constraints specified above. After defining the covariance matrix for all observations as

$$\mathbf{\Sigma}_{\mathbf{x},\mathbf{x}} := \begin{pmatrix} \mathbf{\Sigma}_{\mathbf{x}_1,\mathbf{x}_1} & & 0 \\ & \ddots & \\ 0 & & \mathbf{\Sigma}_{\mathbf{x}_N,\mathbf{x}_N} \end{pmatrix},$$

we can write our optimisation problem as

$$\begin{array}{ll} \text{minimise} & \frac{1}{2}\Delta\mathbf{x}^\top\mathbf{\Sigma}_{\mathbf{x},\mathbf{x}}^{-1}\Delta\mathbf{x} \\ \text{subject to} & \mathbf{A}\,\Delta\mathbf{w} + \mathbf{B}^\top\,\Delta\mathbf{x} = \mathbf{c}_\mathbf{g}, \\ & \mathbf{H}^\top(\hat{\mathbf{w}})\,\Delta\mathbf{w} = \mathbf{c}_\mathbf{h}. \end{array} \tag{3.6}$$

To find a solution for this problem, we use the method of Lagrange multipliers where the Lagrangian function is given as

$$\begin{array}{rcl} L(\Delta\mathbf{w}, \Delta\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) & = & \frac{1}{2}\Delta\mathbf{x}^\top\mathbf{\Sigma}_{\mathbf{x},\mathbf{x}}^{-1}\Delta\mathbf{x} \\ & & -(\mathbf{A}\,\Delta\mathbf{w} + \mathbf{B}^\top\,\Delta\mathbf{x} - \mathbf{c}_\mathbf{g})^\top\boldsymbol{\lambda} \\ & & +(\mathbf{H}^\top(\hat{\mathbf{w}})\,\Delta\mathbf{w} - \mathbf{c}_\mathbf{h})^\top\boldsymbol{\mu}. \end{array} \tag{3.7}$$

Calculating the partial derivatives with respect to $\Delta\mathbf{x}$, $\Delta\mathbf{w}$, $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ gives the following set of equations:

$$\mathbf{\Sigma}_{\mathbf{x},\mathbf{x}}^{-1}\Delta\mathbf{x} - \mathbf{B}\boldsymbol{\lambda} = 0, \tag{3.8}$$

$$-\mathbf{A}^\top\boldsymbol{\lambda} + \mathbf{H}\boldsymbol{\mu} = 0, \tag{3.9}$$

$$\mathbf{A}\Delta\mathbf{w} + \mathbf{B}^\top\Delta\mathbf{x} - \mathbf{c}_\mathbf{g} = 0, \tag{3.10}$$

$$\mathbf{H}^\top\Delta\mathbf{w} - \mathbf{c}_\mathbf{h} = 0. \tag{3.11}$$

Writing equation (3.8) as $\Delta\mathbf{x} = \boldsymbol{\Sigma}_{\mathbf{x},x}\mathbf{B}\boldsymbol{\lambda}$ and substituting this into equation (3.10), we get

$$\boldsymbol{\lambda} = (\mathbf{B}^\top\boldsymbol{\Sigma}_{\mathbf{x},x}\mathbf{B})^{-1}(\mathbf{c}_\mathbf{g} - \mathbf{A}\Delta\mathbf{w}). \tag{3.12}$$

Substituting this equation into equation (3.9) gives

$$\mathbf{A}^\top(\mathbf{B}^\top\boldsymbol{\Sigma}_{\mathbf{x},x}\mathbf{B})^{-1}\mathbf{A}\Delta\mathbf{w} + \mathbf{H}\boldsymbol{\mu} = \mathbf{A}^\top(\mathbf{B}^\top\boldsymbol{\Sigma}_{\mathbf{x},x}\mathbf{B})^{-1}\mathbf{c}_\mathbf{g}. \tag{3.13}$$

If we define

$$\mathbf{N} := \mathbf{A}^\top(\mathbf{B}^\top\boldsymbol{\Sigma}_{\mathbf{x},x}\mathbf{B})^{-1}\mathbf{A} \quad \text{and} \quad \mathbf{c}_\mathbf{n} := \mathbf{A}^\top(\mathbf{B}^\top\boldsymbol{\Sigma}_{\mathbf{x},x}\mathbf{B})^{-1}\mathbf{c}_\mathbf{g}, \tag{3.14}$$

we can combine equations (3.13) and (3.11) into a linear system of equations:

$$\begin{pmatrix} \mathbf{N} & \mathbf{H} \\ \mathbf{H}^\top & 0 \end{pmatrix}\begin{pmatrix} \Delta\mathbf{w} \\ \boldsymbol{\mu} \end{pmatrix} = \begin{pmatrix} \mathbf{c}_\mathbf{n} \\ \mathbf{c}_\mathbf{h} \end{pmatrix}. \tag{3.15}$$

Substituting equation (3.12) into (3.8) gives a way to calculate $\Delta\mathbf{x}$ after solving (3.15) as

$$\Delta\mathbf{x} = \boldsymbol{\Sigma}_{\mathbf{x},x}\mathbf{B}(\mathbf{B}^\top\boldsymbol{\Sigma}_{\mathbf{x},x}\mathbf{B})^{-1}(\mathbf{c}_\mathbf{g} - \mathbf{A}\Delta\mathbf{w}).$$

Now we can update the estimates $\hat{\mathbf{w}}$ for $\mathbf{w}$ and $\hat{\mathbf{x}}$ for $\mathbf{x}$ by

$$\hat{\mathbf{w}}' = \hat{\mathbf{w}} + \Delta\mathbf{w} \quad \text{and} \quad \hat{\mathbf{x}}' = \hat{\mathbf{x}} + \Delta\mathbf{x}.$$

Iterating these calculations will finally lead to a solution of the minimisation problem (3.6).

3.2.4 Numerical calculations

Since we used the matrix based environment Matlab for conducting our experiments, we have added an extra component to each vector, writing the new vectors as

$$\mathbf{x}_i := \begin{pmatrix} \tilde{\mathbf{x}}_i \\ 1 \end{pmatrix}$$

with $\tilde{\mathbf{x}}_i$ being the original observed vector. This enables us to rewrite the constraints from equation (3.1) as

$$\mathbf{g}_i(\mathbf{x}_i, \mathbf{w}) = (\mathbf{n}^\top, b)\begin{pmatrix} \tilde{\mathbf{x}}_i \\ 1 \end{pmatrix} = \mathbf{w}^\top\mathbf{x}_i = 0, \tag{3.16}$$

and the constraints from equation (3.4) as

$$\mathbf{h}(\mathbf{w}) = \frac{1}{2}\left(\mathbf{w}^\top\mathbf{M}_\mathbf{h}\mathbf{w} - 1\right) = 0 \tag{3.17}$$

with

$$\mathbf{M}_\mathbf{h} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & \vdots \\ 0 & \cdots & 0 \end{pmatrix}.$$

The covariance matrices for the new $\mathbf{x}_i$ become

$$\mathbf{\Sigma}_{\mathbf{x}_i,\mathbf{x}_i} = \begin{pmatrix} \mathbf{\Sigma}_{\tilde{\mathbf{x}}_i,\tilde{\mathbf{x}}_i} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{pmatrix},$$

with $\mathbf{0} = (0,\ldots,0)^\top \in \mathbb{R}^2$.

Observing that $\mathbf{\Sigma}_{\mathbf{x},\mathbf{x}}$ is a block diagonal matrix and taking into account the sparse structure of $\mathbf{B}$, it is possible to calculate $\mathbf{N}$ and $\mathbf{c_n}$ without explicitly using huge matrices as equation (3.14) might suggest:

Since

$$(\mathbf{B}^\top \mathbf{\Sigma}_{\mathbf{x},\mathbf{x}} \mathbf{B})^{-1} = \begin{pmatrix} \mathbf{E}^{-1}_{\mathbf{x}_1,\mathbf{x}_1} & & 0 \\ & \ddots & \\ 0 & & \mathbf{E}^{-1}_{\mathbf{x}_N,\mathbf{x}_N} \end{pmatrix}$$

with

$$\mathbf{E}_{\mathbf{x}_i,\mathbf{x}_i} = (\partial_{\mathbf{x}_i}\mathbf{g}_i)(\hat{\mathbf{x}}_i,\hat{\mathbf{w}})\mathbf{\Sigma}_{\mathbf{x}_i,\mathbf{x}_i}[(\partial_{\mathbf{x}_i}\mathbf{g}_i)(\hat{\mathbf{x}}_i,\hat{\mathbf{w}})]^\top,$$

we can calculate $\mathbf{N}$ and $\mathbf{c_n}$ as

$$\mathbf{N} = \sum_{i=1}^{N} [(\partial_{\mathbf{w}}\mathbf{g}_i)(\hat{\mathbf{x}}_i,\hat{\mathbf{w}})]^\top \mathbf{E}^{-1}_{\mathbf{x}_i,\mathbf{x}_i} (\partial_{\mathbf{w}}\mathbf{g}_i)(\hat{\mathbf{x}}_i,\hat{\mathbf{w}})$$

and

$$\mathbf{c_n} = -\sum_{i=1}^{N} [(\partial_{\mathbf{w}}\mathbf{g}_i)(\hat{\mathbf{x}}_i,\hat{\mathbf{w}})]^\top \mathbf{E}^{-1}_{\mathbf{x}_i,\mathbf{x}_i} \mathbf{g}_i(\hat{\mathbf{x}}_i,\hat{\mathbf{w}}).$$

Using the updated definitions (3.16) and (3.17), we get:

$$\begin{aligned} \mathbf{N} &= \sum_{i=1}^{N} \hat{\mathbf{x}}_i \left(\hat{\mathbf{w}}^\top \mathbf{\Sigma}_{\mathbf{x}_i,\mathbf{x}_i} \hat{\mathbf{w}}\right)^{-1} \hat{\mathbf{x}}_i^\top, \\ \mathbf{c_n} &= \sum_{i=1}^{N} \hat{\mathbf{x}}_i \left(\hat{\mathbf{w}}^\top \mathbf{\Sigma}_{\mathbf{x}_i,\mathbf{x}_i} \hat{\mathbf{w}}\right)^{-1} \hat{\mathbf{x}}_i^\top \hat{\mathbf{w}}, \\ \mathbf{H} &= \mathbf{M_h}\hat{\mathbf{w}}, \quad \text{and} \\ \mathbf{c_h} &= \frac{1}{2}\left(1 - \hat{\mathbf{w}}^\top \mathbf{M_h} \hat{\mathbf{w}}\right). \end{aligned}$$

3.2.5 Derivation of the covariance matrix $\mathbf{\Sigma}_{\mathbf{w},\mathbf{w}}$

Taking the system of equations (3.15), we can calculate $\Delta\mathbf{w}$ by inverting the left matrix:

$$\begin{pmatrix} \Delta\mathbf{w} \\ \boldsymbol{\mu} \end{pmatrix} = \begin{pmatrix} \mathbf{N} & \mathbf{H} \\ \mathbf{H}^\top & 0 \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{c_n} \\ \mathbf{c_h} \end{pmatrix}.$$

Using a derivation from [16], we express the inverse as

$$\begin{pmatrix} \mathbf{N} & \mathbf{H} \\ \mathbf{H}^\top & 0 \end{pmatrix}^{-1}$$

$$= \begin{pmatrix} \mathbf{N}^{-1} - \mathbf{N}^{-1}\mathbf{H}(\mathbf{H}^\top\mathbf{N}^{-1}\mathbf{H})^{-1}\mathbf{H}^\top\mathbf{N}^{-1} & \mathbf{N}^{-1}\mathbf{H}(\mathbf{H}^\top\mathbf{N}^{-1}\mathbf{H})^{-1} \\ (\mathbf{H}^\top\mathbf{N}^{-1}\mathbf{H})^{-1}\mathbf{H}^\top\mathbf{N}^{-1} & -(\mathbf{H}^\top\mathbf{N}^{-1}\mathbf{H})^{-1} \end{pmatrix}$$

and can express $\Delta\mathbf{w}$ as

$$\Delta\mathbf{w} = \mathbf{N}^{-1} - \mathbf{N}^{-1}\mathbf{H}(\mathbf{H}^\top\mathbf{N}^{-1}\mathbf{H})^{-1}\mathbf{H}^\top\mathbf{N}^{-1}\mathbf{c_n} + \mathbf{N}^{-1}\mathbf{H}(\mathbf{H}^\top\mathbf{N}^{-1}\mathbf{H})^{-1}\mathbf{c_h}.$$

Using error propagation and following [27], we compute the covariance matrix $\mathbf{\Sigma}_{\Delta\mathbf{w},\Delta\mathbf{w}}$ as

$$\mathbf{\Sigma}_{\Delta\mathbf{w},\Delta\mathbf{w}} = \mathbf{N}^{-1}(\mathbf{I} - \mathbf{H}(\mathbf{H}^\top\mathbf{N}^{-1}\mathbf{H})^{-1}\mathbf{H}^\top\mathbf{N}^{-1}) \tag{3.18}$$

with $\mathbf{I}$ being an identity matrix.

3.2.6 Visualisation of the variance

Having calculated the parameter vector $\mathbf{w} = (\mathbf{n}^\top, b)$ together with the accompanying covariance matrix $\mathbf{\Sigma}_{\mathbf{w},\mathbf{w}}$, we have calculated not only one line l, but a whole distribution of lines with the line given by $\mathbf{w}$ being the mean of the distribution.

Taking a point $\mathbf{x}$ from $\mathbb{R}^2$, we can calculate the distance d to the line l as $d = \mathbf{n}^\top\mathbf{x} - b$. Using error propagation we can calculate the variance for this distance. For the visualisation of the variance of the hyperplanes we have taken some points on the line and plotted bars with length proportional to the variance vertical to the line.

3.2.7 Examples

To illustrate the process of line fitting, we have generated some figures. Figure (3.1) (together with Figure (3.2)) shows the successful learning of the parameters.

Figure (3.3) (together with Figure (3.4)) shows that, if the initial estimate is too bad, it can happen that, due to adopting the original observations towards a bad estimate of the line during early updates, it becomes impossible to learn the optimal line in the later updates. However, it is important to note that in this example the initial estimate is perpendicular to the line used to generate the points. This means that it is approximately the worst possible estimate. But, although this is the worst possible initial estimate, rerunning the experiment with this bad estimate, we found that in the majority of cases the results were satisfactory. This shows that, when a bad initial estimate is used, a failure as depicted in Figures (3.3) and (3.4) can happen, but this very much depends on the actual points used. With a sufficiently good initial estimate, however, we only got good results in our experiments.

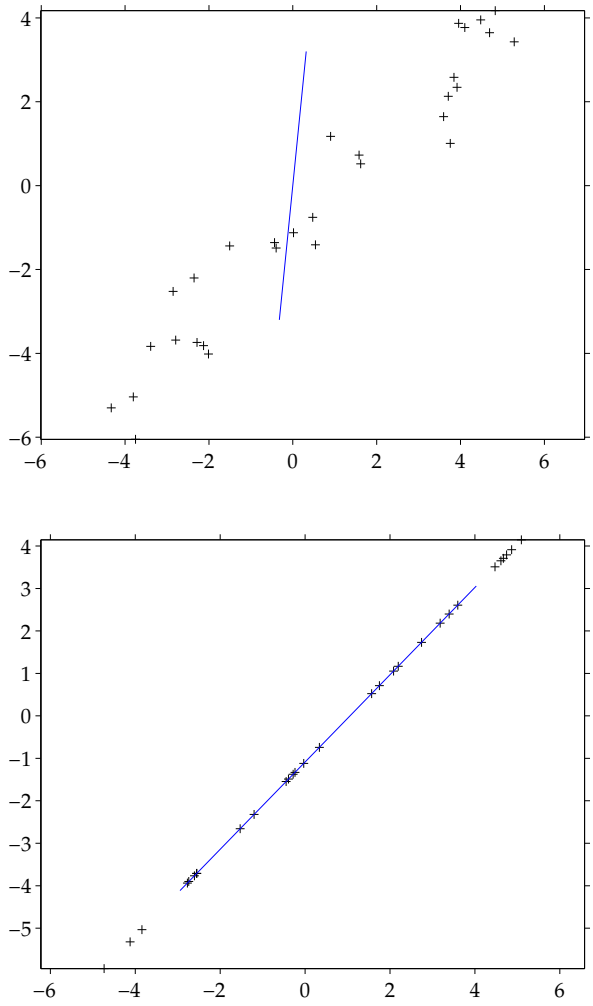

Figure 3.1: The pictures above together with the picture from Figure 3.2 show the use of the Gauss-Helmert model for line-fitting with a sufficiently good initial guess for the parameter vector leading to good performance. The first picture (at the top of this page) depicts the observed data and the initial estimate for the line. The second drawing (centre of this page) shows the result of the learning after the third and final update. The last picture in Figure 3.2 shows the learned line together with the original points to illustrate the quality of the final estimate. The data for this example consists of 30 points uniformly distributed on a line segment and deviated perpendicular to the line with normal distributed noise.

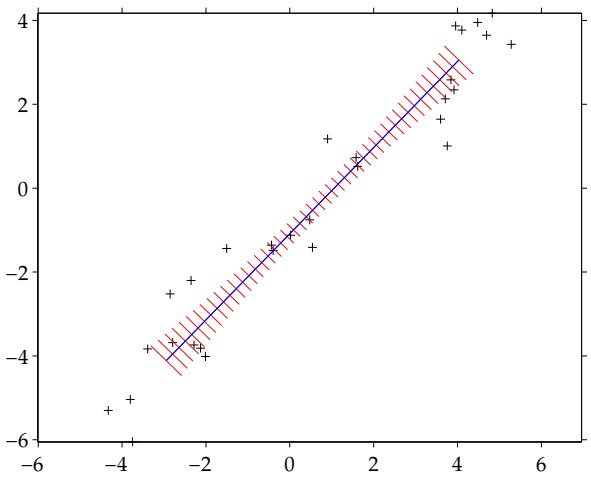

Figure 3.2: Continuation of Figure 3.1: the final estimate for the line together with the original points. For a selection of points lying on the line, we have visualised the variance of their distance to the line by drawing vertical bars with length proportional to the variance.

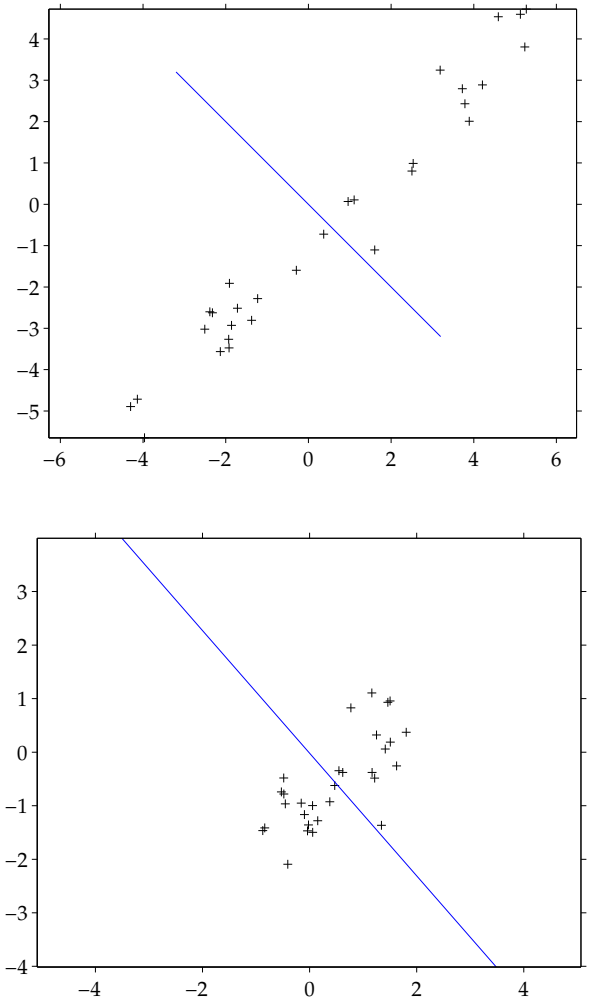

Figure 3.3: Using the Gauss-Helmert model for line-fitting with an inappropriate initial estimate for the parameter vector can lead to very poor performance. The first picture (at the top of this page) depicts the observed data and the initial (bad) estimate for the line. The following three drawings (first drawing: this figure, remaining drawings: Figure 3.4) show the iterative learning process which stops when the updates become smaller than a previously chosen threshold. The last picture shows the learned line together with some vertical bars visualising its variance. It also shows the original points to illustrate the discrepancy between the data and the learned line. The data for this example consists of 30 points uniformly distributed on a line segment and deviated perpendicular to the line with normal distributed noise. (The remaining drawings are shown in Figure 3.4.)

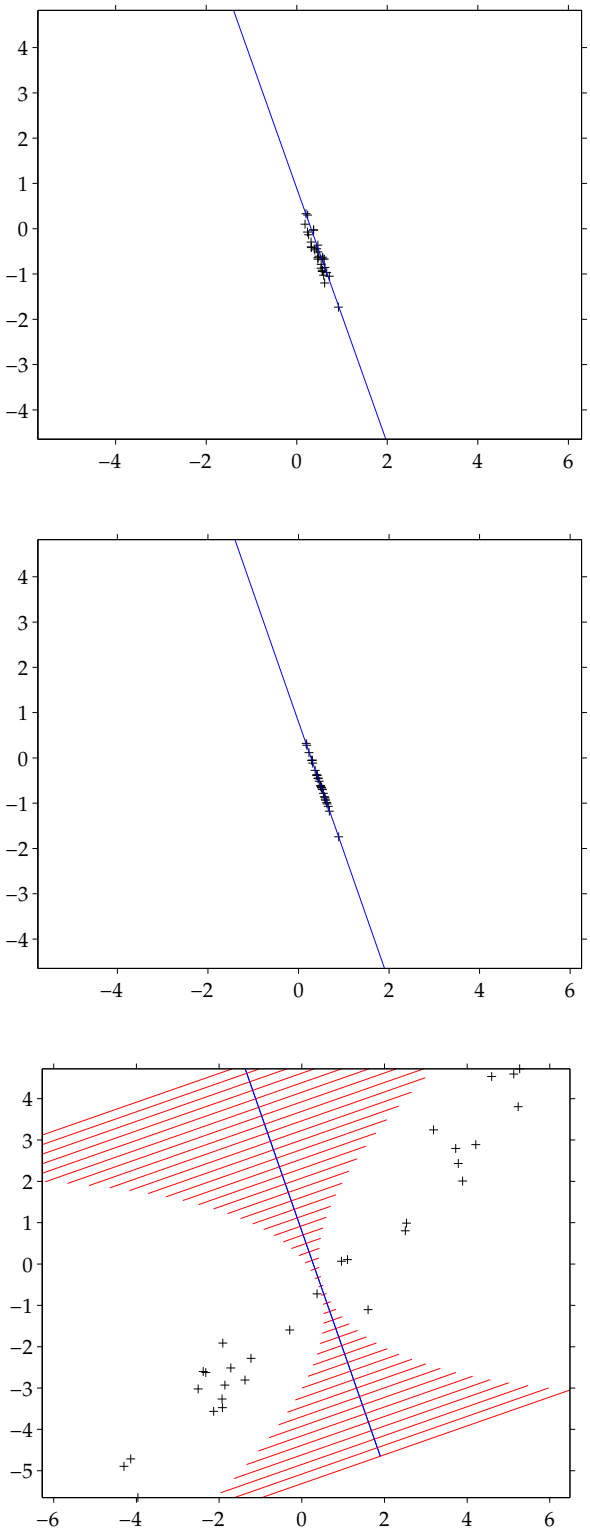

Figure 3.4: Continuation of Figure 3.3.

Chapter 4

Adaptation of the Gauss-Helmert model to linear discrimination

Given our considerations in Sections 2.4 and 2.5, we will now present a method to adapt the Gauss-Helmert model for discrimination tasks. While in this section we will just show an adaptation for *linear* discrimination, we will extend this approach to non-linear discrimination in Chapter 6.

In most cases, the data available for the training of a pattern classification machine is noisy. As a result, a classifier obtaining perfect classification on the training set will usually not perform as good on new data. For this reason, most learning algorithms do not try to learn the training set perfectly but try to learn a smooth decision surface. For many learning algorithms however, the control of the smoothness of the decision surface is the only point where they might – indirectly – get a benefit from a known distribution or at least magnitude of the noise. A possible strategy might be to increase the smoothness for very noisy data and to decrease the smoothness if less noise is present.

Most training algorithms, however, are not able to make direct use of this additional information, especially when it is known that the distribution of noise varies for different observations (e.g. smaller distortions for points in the centre of a picture and larger distortions near the borders). There exist only few algorithms, like the ones presented in Chapter 8, that are able to make use of this information for learning directly.

The approach presented in this section will not only do this but will even go a step further by using the uncertainty information not only for learning the decision boundary but also for calculating a variance for the parameters of the decision boundary. This will then be used to calculate the probability for a new point to lie on one side or the other of a decision boundary drawn from the distribution of the decision boundaries (given by

the parameters and the covariances). This probability will be interpreted as a measure of confidence for the classification result.

4.1 The plain approach

The idea for the adaptation of the Gauss-Helmert model is to fit two parallel hyperplanes through the training data, one through each of the two classes C_+ and C_-, and to use the parallel hyperplane in the middle as the separating hyperplane. This is motivated by the fact that, at least as long as the training set is well balanced, Fisher's linear discriminant is effectively doing the same as the least-squares approach: fitting parallels through the two classes (see Subsection 2.5.2). If the set is unbalanced, i.e. the numbers of positive and negative examples differ considerably, Fisher's linear discriminant will move the separating hyperplane towards the larger class. Since we think that this can become a problem for extremely unbalanced data and for simplicity, we will always take the parallel hyperplane in the middle of the outer parallels as the decision boundary.

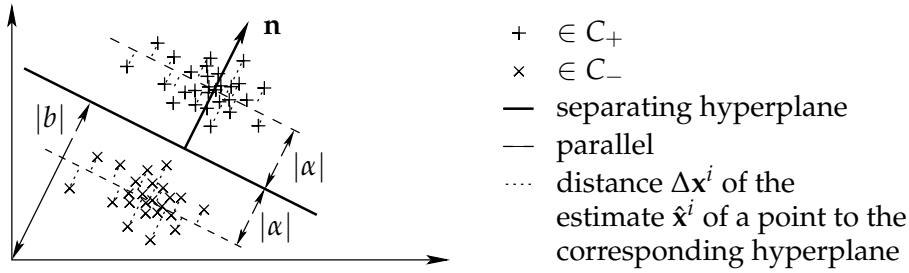

Figure 4.1: Our idea for the adaptation of the Gauss-Helmert model to linear separation: find a separating hyperplane by fitting two parallel hyperplanes through the points of the two classes and take the parallel hyperplane in the middle as the separating hyperplane.

With d-dimensional data, the parameters needed for our model therefore are the normal vector $\mathbf{n} \in \mathbb{R}^d$ for the orientation of the parallel hyperplanes and two additional parameters $b \in \mathbb{R}$ and $\alpha \in \mathbb{R}$ for the distance of the central hyperplane to the origin and for the (negative) distance of the outer parallels to the central parallel.

By extending the approach from the previous chapter (see Subsection 3.2.4), we use a parameter vector

$$\mathbf{w} = \begin{pmatrix} \mathbf{n} \\ b \\ \alpha \end{pmatrix}$$

and extend our original, d-dimensional observation vectors $\tilde{\mathbf{x}}_i$ to $(d+2)$-

dimensional vectors

$$\mathbf{x}_i = \begin{pmatrix} \tilde{\mathbf{x}}_i \\ 1 \\ y_i \end{pmatrix}.$$

The type-one constraints now become

$$\mathbf{g}_i(\mathbf{x}_i, \mathbf{w}) = (\mathbf{n}^\top, b, \alpha) \begin{pmatrix} \tilde{\mathbf{x}}_i \\ 1 \\ y_i \end{pmatrix} = \mathbf{w}^\top \mathbf{x}_i = 0 \tag{4.1}$$

and ensure that all points will finally lie on the correct hyperplane. The type-two constraint becomes

$$\mathbf{h}(\mathbf{w}) = \frac{1}{2}(\mathbf{n}^\top \mathbf{n} - 1) = \frac{1}{2}\left(\mathbf{w}^\top \mathbf{M}_\mathbf{h} \mathbf{w} - 1\right) = 0 \tag{4.2}$$

with

$$\mathbf{M}_\mathbf{h} = \begin{pmatrix} 1 & & & & 0 \\ & \ddots & & & \\ & & 1 & & \\ & & & 0 & \\ 0 & & & & 0 \end{pmatrix} \in \mathbb{R}^{(d+2)}.$$

The covariance matrices for the new $\mathbf{x}_i$ become

$$\mathbf{\Sigma}_{\mathbf{x}_i, \mathbf{x}_i} = \begin{pmatrix} \mathbf{\Sigma}_{\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_i} & & 0 \\ & 0 & \\ 0 & & 0 \end{pmatrix} \in \mathbb{R}^{(d+2)}.$$

Using these modifications, we can use the same procedure as in Chapter 3 and calculate a parameter vector $\mathbf{w} = (\mathbf{n}^\top, b, \alpha)^\top$ together with a covariance matrix $\mathbf{\Sigma}_{\mathbf{w},\mathbf{w}}$.

4.1.1 Uncertainty of the classification

In Section 2.2 we have motivated the approach of linear classification of the data. Using an approach based on the idea of Fisher's linear discriminant, we will find a separating hyperplane which gives optimal separation in the Fisherian sense (at least as long as our data set is well balanced). But, using noisy input, our estimate of the separating hyperplane will be noisy too.

The distance to the separating hyperplane cannot be used to measure the confidence

At first sight, it seems obvious that for two points lying on a line perpendicular to the separating hyperplane, the confidence on the classification of the point with greater distance to the hyperplane is greater. But for two

arbitrary points this does not hold. We will illustrate this with a simple example:

If we are thinking of a distribution of lines through the origin of a two-dimensional coordinate system, it becomes clear that for a point lying near the origin, we can predict with high confidence on which side of a line drawn from the distribution, this point will lie. If however, we take a point with the same distance to the mean line but further away from the origin of the coordinate system, our confidence decreases as illustrated in Figure 4.2.

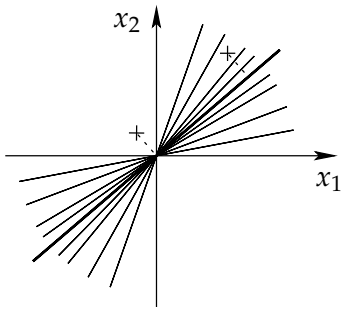

Figure 4.2: This figure shows some lines through the origin drawn from a distribution. The mean of this distribution is drawn with a thick solid line. The two points both have the same distance to the mean of the lines. However, the point closer to the origin lies above all lines shown in this drawing, while the other point lies only above 7 of the 9 lines.

Therefore we have adopted a more sophisticated way to give a measure of confidence for our classification results.

A more sophisticated way to measure the confidence

Using the Gauss-Helmert model, we get the (mean of the) parameters of a separating hyperplane together with a covariance matrix. This allows us for an arbitrary point $\mathbf{x}$ the definition of a function

$$d_{\mathbf{x}} : \begin{pmatrix} \mathbf{n} \\ b \end{pmatrix} \mapsto (\mathbf{n}^\top, b) \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix}$$

that calculates the distance of the point $\mathbf{x}$ to the hyperplane given by the parameters $\mathbf{n}$ and b since the type-two constraint (4.2) ensures that $\|\mathbf{n}\|_2 = 1$. Using error propagation (see Appendix A) we calculate the expectations of the Taylor series expansions of degree two for $d_{\mathbf{x}}$ and $d_{\mathbf{x}}^2$ and get

$$\mathcal{E}[d_{\mathbf{x}}((\mathbf{n}^\top, b)^\top)] = (\mathbf{n}^\top, b) \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix} =: \mu \tag{4.3}$$

and

$$\mathcal{E}[d_{\mathbf{x}}^2((\mathbf{n}^\top, b)^\top)] \approx \left((\mathbf{n}^\top, b)\begin{pmatrix}\mathbf{x}\\1\end{pmatrix}\right)^2 + (\mathbf{x}^\top, 1, 0)^\top \boldsymbol{\Sigma}_{\Delta\mathbf{w},\Delta\mathbf{w}} \begin{pmatrix}\mathbf{x}\\1\\0\end{pmatrix}. \quad (4.4)$$

Combining (4.3) and (4.4) we get

$$\begin{aligned} Var[d_{\mathbf{x}}((\mathbf{n}^\top, b)^\top)] &= \mathcal{E}[d_{\mathbf{x}}^2((\mathbf{n}^\top, b)^\top)] - \mathcal{E}^2[d_{\mathbf{x}}((\mathbf{n}^\top, b)^\top)] \\ &\approx (\mathbf{x}^\top, 1, 0)^\top \boldsymbol{\Sigma}_{\Delta\mathbf{w},\Delta\mathbf{w}} \begin{pmatrix}\mathbf{x}\\1\\0\end{pmatrix} =: \sigma^2. \end{aligned} \quad (4.5)$$

Therefore, given (4.3) and (4.5), we have an estimate for the mean and variance for the distance of a point to the hyperplane. If the mean is less than 0 we assign the point to the class C_-, otherwise to class C_+. The probability for $\mathbf{x}$ being assigned to class C_+ becomes

$$P_{C_+}(\mathbf{x}) = \int_{\mathbb{R}_{\geq 0}} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(t-\mu)^2}{2\sigma^2}} dt,$$

for $\mathbf{x}$ being assigned to class C_-

$$P_{C_-}(\mathbf{x}) = \int_{\mathbb{R}_{\leq 0}} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(t-\mu)^2}{2\sigma^2}} dt.$$

If $\mu < 0$, i.e. $\mathbf{x}$ is assigned to class C_-, we can reformulate the second case as

$$\int_{\mathbb{R}_{\geq 0}} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(t-|\mu|)^2}{2\sigma^2}} dt. \quad (4.6)$$

Since this is equal to the first case if we assign $\mathbf{x}$ to class C_+, we can use (4.6) as the probability of assigning $\mathbf{x}$ to the class with greatest probability. Therefore we use (4.6) as a measure of confidence for our classification.

4.1.2 Experiments

For our first experiments we created some balanced data sets with points drawn from multivariate normal distributions. The covariance matrices for all points are taken to be equal multiples of the unity matrix. In order to be able to create intuitive visualisations of the results, we restricted ourselves mainly to two-dimensional data sets.

As the initial value for $\mathbf{n}$ we used the normalised difference of the class means $(\mathbf{m}_+ - \mathbf{m}_-)/\|\mathbf{m}_+ - \mathbf{m}_-\|_2$. For b we have taken $b = -(\mathbf{m}_- + (\mathbf{m}_+ - \mathbf{m}_-)/2)^\top \mathbf{n}$ and for α we used $\alpha = -\|\mathbf{m}_+ - \mathbf{m}_-\|_2/2$. We stopped iterating when the norm of the difference between the new and the previous estimate for $\mathbf{p}$ dropped below 10^{-4}.

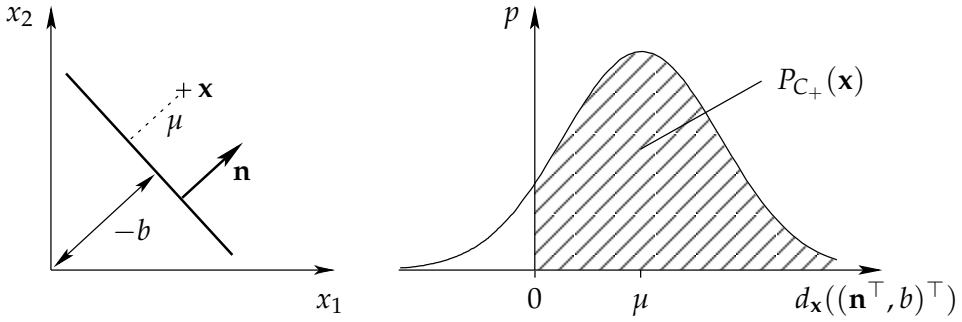

Figure 4.3: Visualisation of the calculation of the confidence for a point $\mathbf{x}$ lying on the positive side of the separating hyperplane specified by $\mathbf{n}$ and b. The left drawing shows the distance μ from $\mathbf{x}$ to the mean of the distribution of the separating hyperplanes. Different hyperplanes drawn from the distribution of separating hyperplanes lead to different distance from $\mathbf{x}$ to the hyperplane. The plot on the right shows the probability density for the varying distance from $\mathbf{x}$ to the hyperplanes. The hedged area represents the probability of $\mathbf{x}$ being classified as positive.

The first data set

The first data set consists of 40 points, 20 points belonging to each of the two classes C_+ and C_-. Points from class C_+ are drawn from the bivariate normal distribution with mean $(1,1)^\top$ and covariance matrix $0.25\,\mathbf{I}$ with $\mathbf{I}$ being the two-dimensional identity matrix. Mean and covariance matrix for class C_- are $(-1,-1)^\top$ and $0.25\,\mathbf{I}$. To add some noise we have added $0.5\,\mathbf{I}$ as covariance matrix to all points, i.e. we assume an equal distribution of the noise on all points.

We stopped after three iterations. Figure 4.4 shows the resulting hyperplane together with the original points and the parallels, and gives some information on the confidence too.

To illustrate that our method will still work when our data is not centred around the origin, we have rotated and moved all points from our first data set. The results for this modified data set is shown in Figure 4.5.

The second data set

The distributions used to draw the points from this data set have the same covariance matrices for both classes. The covariance matrices have eigenvectors $(1,-1)^\top$ and $(1,1)^\top$ with eigenvalues 2 and $1/4$. The class means are $(1,1)^\top$ and $(-1,-1)^\top$, the covariance matrix for the points is $0.5\,\mathbf{I}$ for all points. The dataset consists of 20 points per class. The long clouds of the two classes fit our model well, since it assumes the points to lie on two parallels. Therefore the fit is very good and the variance on the parameters of the separating line very low. This can be seen in Figure 4.6.

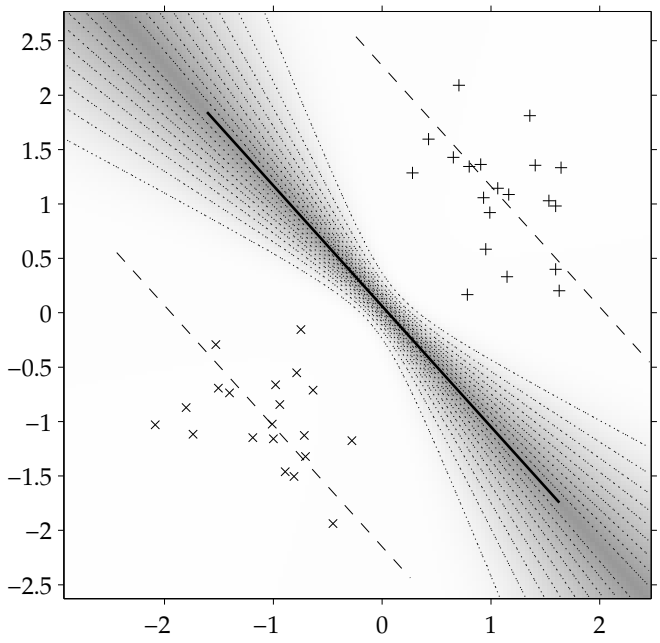

Figure 4.4: The first data set. The points are drawn from two bivariate normal distributions as described in the text. Points belonging to class C_+ are drawn as $+$, points from class C_- as $\times$. The separating hyperplane is the thick solid line, the parallels with distance $-\alpha$ to the separating plane are dashed. The confidence is visualised by the shading of the background. Dark: high uncertainty, light: high confidence. The dotted lines connect points of equal confidence. There are lines for confidence values of $0.55, 0.60, 0.65, \ldots, 0.95$.

The third data set

This dataset is not linearly separable since the classes overlap. Class means are $(1,1)^\top$ and $(0,0)^\top$, covariance matrices for both classes are $2\,\mathbf{I}$ and the covariance matrices for all points are $0.5\,\mathbf{I}$. There are 20 points drawn from each of the two distributions. Results after 4 iterations, when the norm of the alteration of p dropped below 10^{-4} for the first time, are shown in Figure 4.7.

The fourth data set

This data set consists of 40 points drawn from two distributions with different covariance matrices. The covariance matrix for C_+ has eigenvalues $1/16$ and $1/2$ and eigenvectors $(1,-1)^\top$ and $(1,1)$. The matrix for class C_- has eigenvectors $(0,1)^\top$ and $(1,0)^\top$ also with eigenvalues $1/16$ and $1/2$. The class means are $(1,1)^\top$ and $(1.5,-0.4)^\top$. There are 20 points drawn from

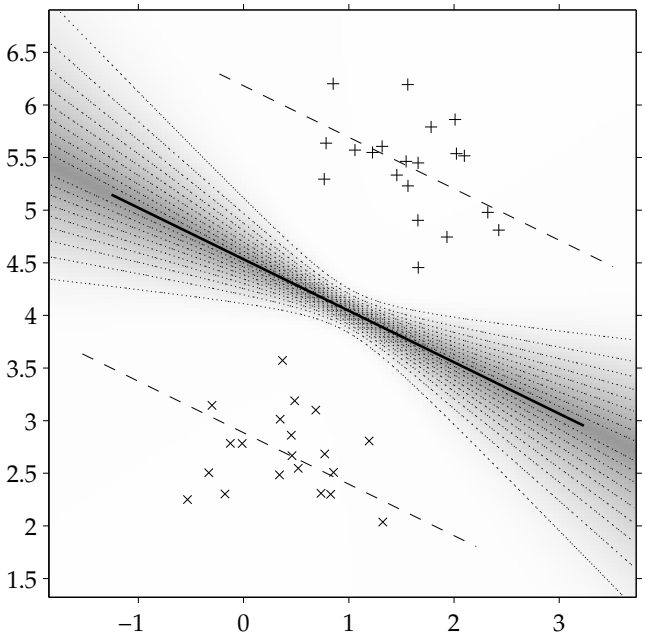

Figure 4.5: The first data set moved and rotated. Results after 3 iterations.

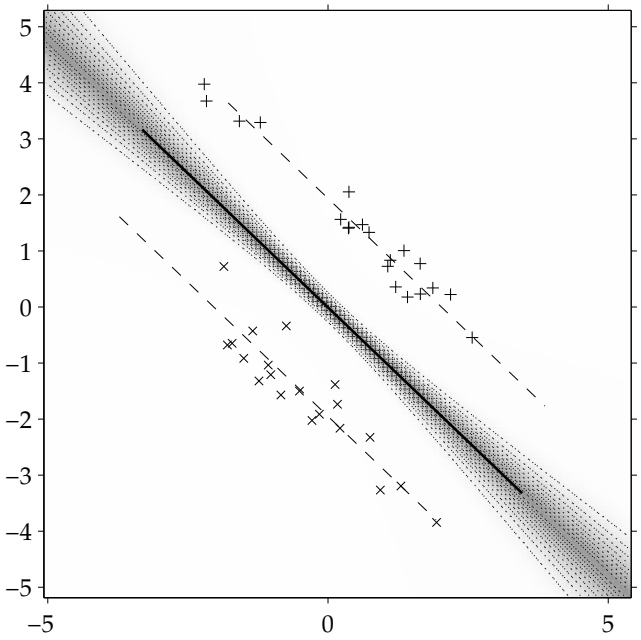

Figure 4.6: The second data set. Results after 3 iterations.

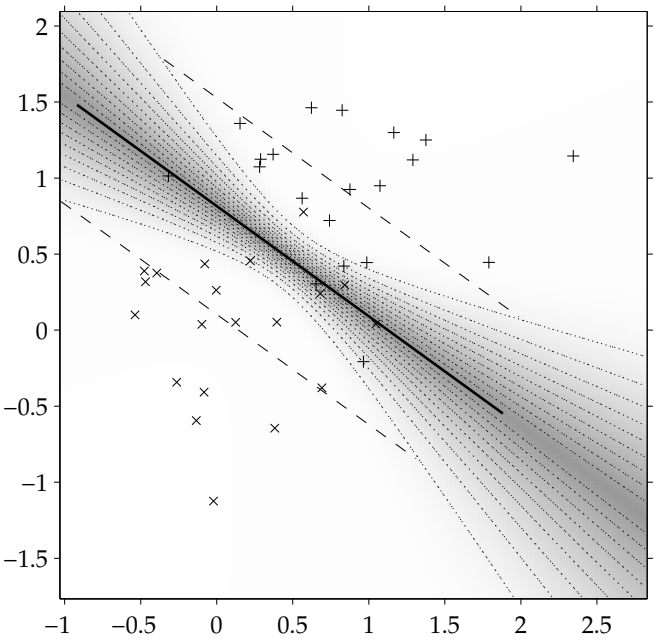

Figure 4.7: The third data set. Results after 4 iterations.

each of the two distributions. The covariance matrices for all points are $0.5\,\mathbf{I}$. For the computed separating hyperplane and confidence see Figure 4.8.

The fifth data set

The fifth data set consists of points from a square of size 8×8 with its lower left corner at $(-2, -2)^\top$. The square is divided by its main diagonal: the points in the lower right triangle belong to class C_+, the ones in the upper left triangle to class C_-. The density of the distribution that the points have been drawn from increases towards the lower left corner. This is due to the fact that the x_1 and x_2 coordinates of the points have not been generated independently from each other. For the right triangle we first generated the x_1 coordinates from the uniform distribution on the interval $[-2, 6]$. After that we generated the x_2 coordinates uniformly from $[-2, x_1]$. For the upper left triangle we did the same with the only difference that we flipped the x_1 and the x_2 coordinates. The covariance matrix used for all points is $\mathbf{I}$.

There are 20 points drawn from each of the two distributions. For the sample used, we get a perfect separation of the training set. Since our separating line is not identical with the diagonal of the 8×8 square, the performance on a new set of points drawn from the same distributions will not be perfect but, keeping in mind the small size of the training set, quite good.

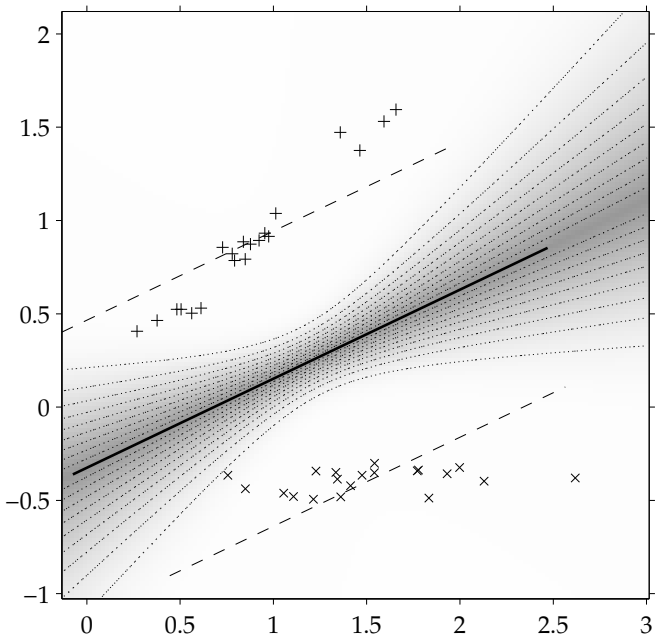

Figure 4.8: The fourth data set. Results after 3 iterations.

Note that, since we are doing effectively the same as Fisher's linear discriminant, we get nearly the same parameters. This can be seen in Table 4.1 that compares the first three components of the parameter vector obtained with our Gauss-Helmert approach with that from Fisher's linear discriminant, from a hard margin support vector machine, and with the 'perfect' solution.

The sixth data set

The sixth data set is a balanced data set with 40 points. The 20 points belonging to class C_+ are drawn from the bivariate normal distribution with the mean $(1,1)^\top$ and the covariance matrix with eigenvectors $(1,-1)^\top$ and $(1,1)^\top$ with corresponding eigenvalues $1/16$ and $1/2$. The distribution for class C_- has the mean $(-1,-1)^\top$ and the same covariance matrix. The covariance matrix for the points is $0.5\,\mathbf{I}$ for all points. Figure 4.10 shows the results of running our Gauss-Helmert based method on this data set.

Table 4.2 compares the parameter vectors obtained by using our Gauss-Helmert approach, Fisher's linear discriminant and an SVM with the parameters of the 'best possible' separating hyperplane.

It can be seen that for this data set the solution found by Fisher's linear discriminant deviates from the one found by the Gauss-Helmert method significantly. This can be explained by the fact that our approach is biased

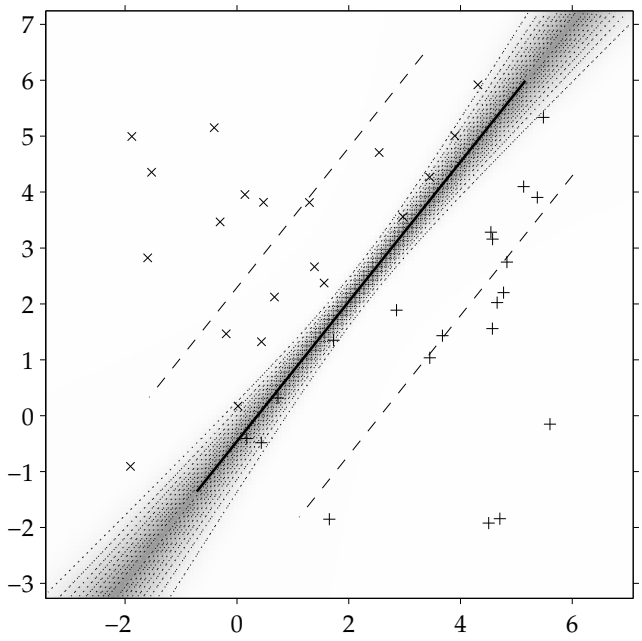

Figure 4.9: The fifth data set. Results after 3 iterations. The dotted lines connect points of equal confidence, lines are drawn for the same confidence values as in Figure 4.4.

	Gauss-Helmert	Fisher's lin. discriminant	SVM	optimal value if underlying distribution known
n_1	0.7809	0.7809	0.7537	0.7071
n_2	-0.6247	-0.6247	-0.6572	-0.7071
b	-0.2858	-0.2857	-0.1235	0

Table 4.1: The parameters for the fifth data set using our Gauss Helmert based approach, Fisher's linear discriminant and a vanilla support vector machine with a linear kernel. The right column shows the best possible solution regarding the distributions used to generate the sample. Since the sample is small, it is far from being able to represent the underlying distributions perfectly. Therefore, taking just the 40 points given, it is probably impossible to find a hyperplane that will perform perfectly on any dataset drawn from the distributions described.

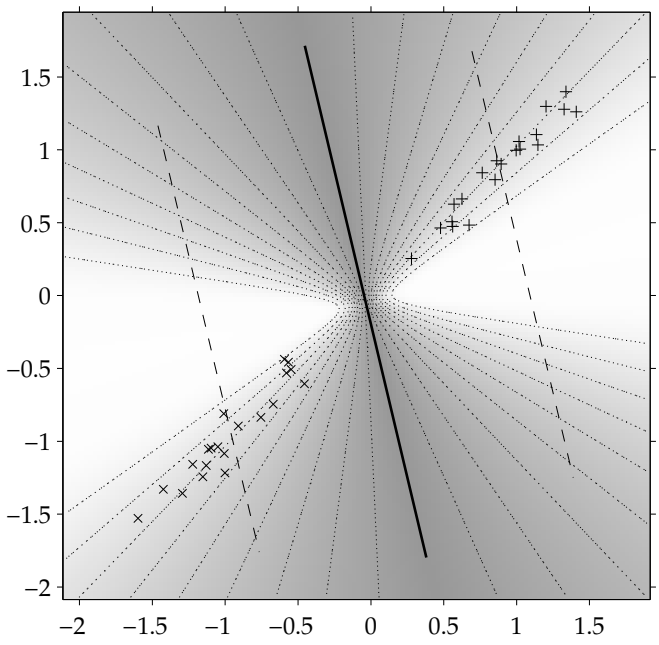

Figure 4.10: The sixth data set. Results after 5 iterations. The confidence is visualised by the shading of the background. Dark: high uncertainty, light: high confidence. The dotted lines connect points of equal confidence. There are lines for confidence values of $0.55, 0.60, 0.65, \ldots, 0.95$.

	Gauss-Helmert	Fisher's lin. discriminant	SVM	optimal value if underlying distribution known
n_1	0.9729	0.9993	0.7649	0.7071
n_2	0.2310	0.0386	0.6441	0.7071
b	0.0454	0.0383	0.1750	0
α	-1.1077			-1.4142

Table 4.2: Entries of the parameter vectors for the sixth data set using our Gauss Helmert based approach, Fisher's linear discriminant and a vanilla support vector machine with a linear kernel. The right column shows the best possible solution regarding the distributions used to generate the sample.

by the initial parameter vector. In the experiments presented so far, we have always used the procedure described in the beginning of this subsection (see page 29) for calculating an initial estimate for the parameter vector. In this experiment (sixth data set), this actually turns out to be the optimal choice of parameters when considering the underlying distributions used for creating the data set.

Although our Gauss-Helmert based approach performs slightly better than Fisher's linear discriminant, the SVM outperforms both.

4.1.3 Problems

Looking at the results for the sixth data set (Figure 4.10 and Table 4.2), we do not only observe that the parameters learned by the Gauss-Helmert approach are much worse than the 'optimal' solution and also worse than the SVM solution. We also notice that the complete training data lies in regions of considerable uncertainty i.e. of low confidence. This is dangerous in so far as the clusters of points from the two classes do not overlap. Consequently the areas where the points are observed should lie within the areas of highest confidence. Therefore the information on the (un)certainty must be used with care.

While the problems described above can be seen as minor problems in the performance, this data set reveals a far greater problem of our approach: even though we start with an initial vector of 'optimal' parameters, we finally get a parameter vector that is *much* worse. The similar performance of Fisher's linear discriminant shows that this is not just a problem of our specific approach but a general problem for a whole class of architectures.

The problem lies in our model. While the original intention of Fisher's linear discriminant was to find a projection that maximises the distance of the projected class means while at the same time keeping the within-class variances small, it has been shown that this is equivalent to a least squares approach (see Subsection 2.5.2). But a least squares approach can be seen as fitting parallels through the data: one through each class such that the sum of the squared distances of the points to their corresponding hyperplane is minimised. A parallel hyperplane between these parallels (in our approach the one in the middle) is then used as the separating hyperplane.

This is the key point: we assume that the data is lying on two parallels that are parallel to a good separating hyper plane. This, however, is not the case for the sixth data set: here the best fitting parallels are close to perpendicular to the best separating hyperplane. This leads to the effect that when starting with the best fitting hyperplanes, i.e. using a normal vector $\mathbf{n}$ that is the 'optimal' $\mathbf{n}$ rotated by 90°, the orientation of the hyperplanes will not change much. We will therefore get very good fitting, very narrow hyperplanes and an unacceptable 'separating' hyperplane. Figure 4.11 shows the result of using the optimal $\mathbf{n}$ rotated by 90° as the initial $\mathbf{n}$.

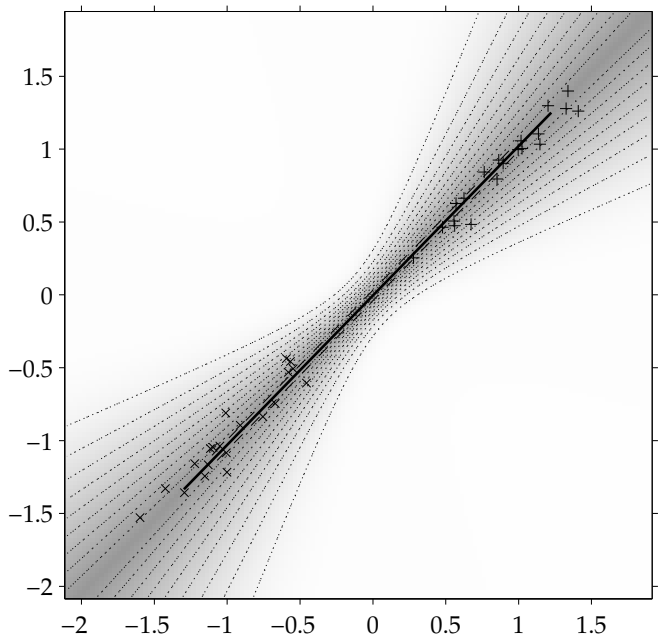

Figure 4.11: The sixth data set. Results after 2 iterations starting with optimal parameters but **n** rotated by 90°.

Taking these observations into account, it seems sensible to investigate means of making our approach more robust against such failures.

4.2 Variants

In the last section we have shown how the Gauss-Helmert model can be used for linear discrimination in general. While testing our implementation with a variety of different datasets, it has turned out that there are some linearly separable datasets for which the obtained separating hyperplane is far from optimal.

As pointed out in the previous subsection, this is due to a deficiency of our model. To compensate for this deficiency, we have investigated a variety of ways to improve the performance of our approach on these 'difficult' datasets.

The probably simplest idea is to use a 'good' initial estimate for the parameter vector. Although this approach looks neat, it has one big disadvantage: it does not work. The reason for this is, that during the optimisation the parameter vector might be altered such that the resulting parameters can become much worse than the initial parameters. Since we effectively

used this approach for the first experiment on our sixth dataset, we have already given an example for the failure of this approach.

This observation led to the next idea for improving the performance: start with a 'good' initial estimate for the parameter vector *and* restrict the adaptation of the parameter vector during the training such that the resulting parameters are close to the 'good' initial estimate and therefore also 'good'. The disadvantage of this approach is that it becomes necessary to calculate a proper estimate for the parameter vector before the optimisation, which originally was intended to 'search' for the parameter vector, because the initial estimate will more or less become the resulting parameter vector. This means that Gauss-Helmert is no longer used for obtaining a parameter vector that describes a separating hyperplane. Instead we use Gauss-Helmert for finding a covariance for the parameter vector. The calculation of the (initial) parameter vector has to be done by another method. When it was not obvious by the shape of the data how the best estimate for the parameter vector would look like, we decided to use a support vector machine (SVM) to calculate the parameters for the best separating hyperplane.

4.2.1 Using a fixed value for α

The most successful way for restricting the adaptation of the parameter vector was the fixing of the value for α.

The idea works as follows: first we calculate the normal vector $\mathbf{n}$ for the separating hyperplane with a more robust method like an SVM. Then we fit two parallels that are orthogonal to the previously calculated normal vector through the data. Taking the hyperplane in the middle between these parallels enables us to calculate the remaining parameters b and α. Finally we start a slightly modified version of our Gauss-Helmert based approach.

The modification is that we use α as a constant instead of as a parameter. Therefore our parameter vector has dimension $d + 1 = 3$ and becomes

$$\mathbf{w} = \begin{pmatrix} \mathbf{n} \\ b \end{pmatrix}$$

and the extended observation vectors become

$$\mathbf{x}_i = \begin{pmatrix} \tilde{\mathbf{x}}_i \\ 1 \end{pmatrix}.$$

The type-one constraints become

$$\mathbf{g}_i(\mathbf{x}_i, \mathbf{w}) = (\mathbf{n}^\top, b) \begin{pmatrix} \tilde{\mathbf{x}}_i \\ 1 \end{pmatrix} + y_i\alpha = \mathbf{w}^\top \mathbf{x}_i + y_i\alpha = 0$$

and the matrices $\mathbf{\Sigma}_{\mathbf{x}_i,\mathbf{x}_i}$ and $\mathbf{M_h}$ are reduced to matrices of size $(d+1) \times (d+1)$ by omitting one of their all-zero columns and rows.

Figure 4.12 shows the resulting hyperplane, parallels and confidence map for the sixth data set when starting with optimal values and using a fixed α.

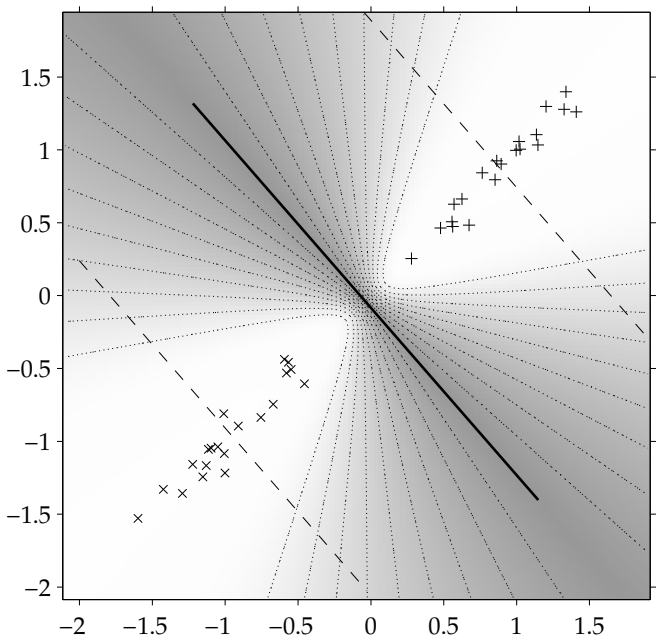

Figure 4.12: The sixth data set. Results after 7 iterations starting with optimal parameters and a fixed optimal value for α.

4.2.2 Minimising the alteration of α

Another less rigorous approach that we tried was the minimisation of the alteration of α. Instead of fixing α completely as in subsection 4.2.1 we start with a good initial estimate for α as well, but instead of fixing α completely, we just tried to keep the alteration of α small in order to stay close to the initial 'good' estimate. This means that this approach can be seen as lying somewhere between the plain approach and the approach using a fixed value for α. However, it turned out that there was no significant difference between the results of our plain approach and of this modified approach.

Chapter 5

Classic architectures for non-linear discrimination

Although linear discrimination has many nice properties, there are some tasks where non-linear classification is necessary.

There have evolved a variety of architectures that are able to perform a non-linear separation of the input space. Many of these architectures are based on linear architectures like the ones presented in Chapter 2.

One of these architectures is the multi-layer perceptron. It consists of several layers of linear perceptrons that are connected via non-linear sigmoid functions. Because of its high non-linear complexity, it does not seem feasible to combine the architecture of a multi-layer perceptron with our architecture for linear classification presented in the previous chapter.

Therefore we will describe other architectures for non-linear classification in this chapter and discuss an extension of our approach from Chapter 4 using ideas from the these architectures in the following chapter.

5.1 RBF networks

The way a radial basis function network classifies new patterns can be seen as a two stage process: first the patterns are non-linearly projected from the input space into a feature space. There, in the feature space, a linear classification takes place.

Classification

While a linear classifier uses a function

$$h_{\text{lin}} : \mathbf{x} \mapsto \text{sgn}(\mathbf{w}^\top \mathbf{x}) = \text{sgn}(\sum_{n=1}^{d} w_n x_n) \in \{-1, 1\}$$

to decide whether a pattern $\mathbf{x}$ belongs to class C_- ($h(\mathbf{x}) = -1$) or class C_+ ($h(\mathbf{x}) = +1$), a radial basis function network usually uses a function

$$h_{\text{rbf}} : \mathbf{x} \mapsto \operatorname{sgn}\Big(\sum_{m=1}^{M} w_m \phi_m(\|\mathbf{x} - \mathbf{x}^m\|_2)\Big).$$

The M functions

$$\mathbf{x} \mapsto \phi_m(\|\mathbf{x} - \mathbf{x}^m\|_2) \tag{5.1}$$

are called *basis functions*[1] with centres $\mathbf{x}^m$. The functions $\phi_m : \mathbb{R}_{\geq 0} \to \mathbb{R}$ are usually chosen to be monotonically decreasing. The most popular choice for the ϕ_m is the Gaussian

$$\phi_m : x \mapsto \exp\Big(-\frac{x^2}{2\sigma_m^2}\Big). \tag{5.2}$$

Although not being a radially symmetric basis function, we can also use

$$\mathbf{x} \mapsto \exp\Big(-\frac{1}{2}(\mathbf{x} - \mathbf{x}^m)^\top \boldsymbol{\Sigma}_m^{-1}(\mathbf{x} - \mathbf{x}^m)\Big) \tag{5.3}$$

as a basis function, possibly scaled by a factor $1/\big((2\pi)^{d/2}|\boldsymbol{\Sigma}_m|^{1/2}\big)$. The matrix $\boldsymbol{\Sigma}_m^{-1}$ is a symmetric positive semidefinite matrix. Since (5.3) is equivalent to using a Gaussian with parameter σ_m when $\boldsymbol{\Sigma}_m = \sigma_m^2 \mathbf{I}$, we can see (5.3) as a generalisation of using (5.2).

Learning

Training a linear classifier can be seen as the learning of a weight vector $\mathbf{w} \in \mathbb{R}^d$ (or $\in \mathbb{R}^{(d+1)}$ if we do not seek a hyperplane through the origin).

An RBF network usually uses a greater number of parameters. While the weight vector $\mathbf{w} \in \mathbb{R}^M$ (or $\in \mathbb{R}^{(M+1)}$) is similar to the weight vector of a linear classifier, there are some additional parameters. These are the number of basis functions M, and the basis functions themselves. Since the type of the basis functions is usually chosen before the training starts, only the parameters of these functions (the centres $\mathbf{x}^m$ and e.g. σ_m when using Gaussians for the ϕ_m or $\boldsymbol{\Sigma}_m$ when using basis functions of the form (5.3)) have to be chosen.

To achieve the best possible classification results, one has to learn all parameters – the number of basis functions M, the centres $\mathbf{x}^m$ and the remaining parameters for the M basis functions – simultaneously. The drawback of such an approach is its high computational costs.

Because of these high computational costs, it is more common to use a two-stage training process where in the first stage unsupervised techniques

[1] In the literature the term basis function is used differently: sometimes it is used for the functions (5.1), sometimes for ϕ_m and sometimes (see e.g. [22]) it is used to denote a function with two parameters: $\mathbf{x}$ and $\mathbf{x}^m$.

are used to determine the basis functions and in the second stage the outputs of the M basis functions are used as the input for the training of a linear classifier.

In this text we will only use the two-stage training. We choose the number M of basis functions first and then use a k-means clustering algorithm to find the centres of the basis functions. Our basis functions have the form of (5.3) where we use a (possibly scaled) estimate of the covariance matrix for the distribution of the points from the mth cluster for $\mathbf{\Sigma}_m$.

An example

To illustrate the process of training an RBF network, we have generated some images. We begin with the data set shown in Figure 5.1.

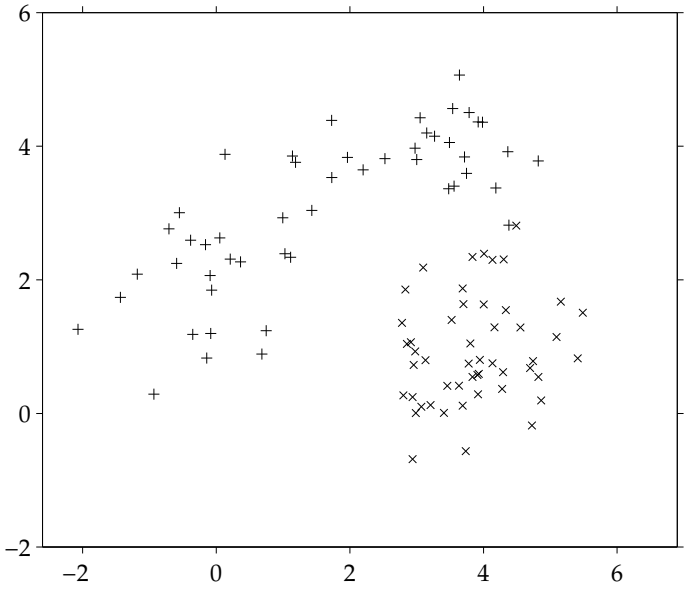

Figure 5.1: The data set. Points drawn as $+$ belong to class C_+, points drawn as $\times$ belong to class C_-.

Our training starts with the unsupervised clustering of the data set. We have decided to use $M = 3$, i.e. we will get 3 clusters. We use a k-means clustering algorithm. Taking the points from one cluster we also calculate an estimate for the covariance matrix for the distribution of the points from this cluster. In Figure 5.2 we visualise the covariance matrices with ellipses and the clustering by using different symbols for the points from each cluster.

Using the means of the clusters together with the covariance matrices allows us to approximate the distributions of the points from the clusters using bivariate normal distributions. These distributions are shown in Fig-

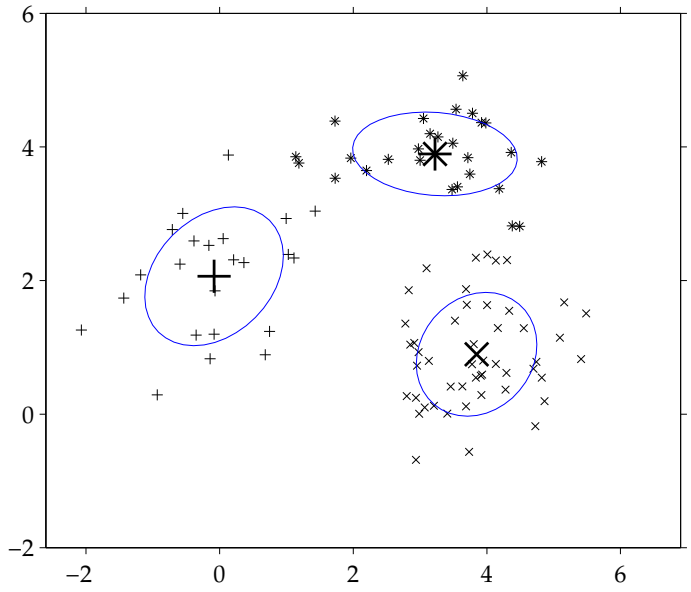

Figure 5.2: The 3 clusters together with their covariances visualised by using ellipses.

ure 5.3. For our RBF network we just use these distributions as basis functions (with the covariance matrices scaled by a factor slightly greater than one). This means that the ith component of the input used for the linear classifier is the probability density of this point for the distribution of the ith cluster.

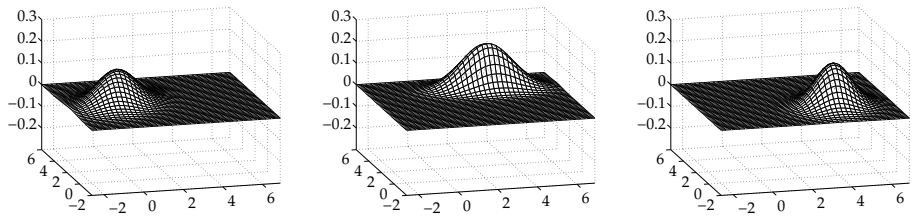

Figure 5.3: The distributions of the clusters – the basis functions.

Training the linear classifier with the projected training data gives a weight vector $\mathbf{w} \in \mathbb{R}^M$. In the space of the projected data the weight vector defines a separating hyperplane. Figure 5.4 shows this hyperplane together with the projected sample.

Forming the weighted sum of the M probability distributions weighted by the corresponding entry of $\mathbf{w}$ gives a new function that is shown in Figure 5.5.

A linear classifier classifies a pattern as belonging to class C_+ if the prod-

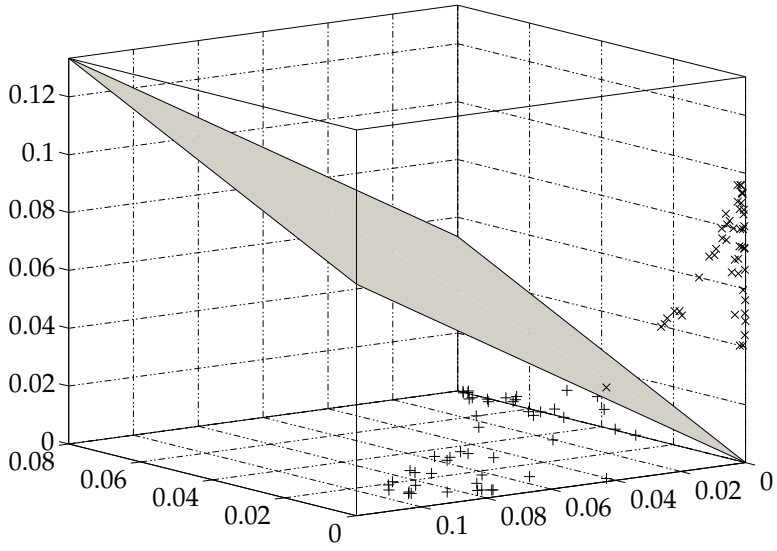

Figure 5.4: The hyperplane defined by the weight vector **w** in the projected space.

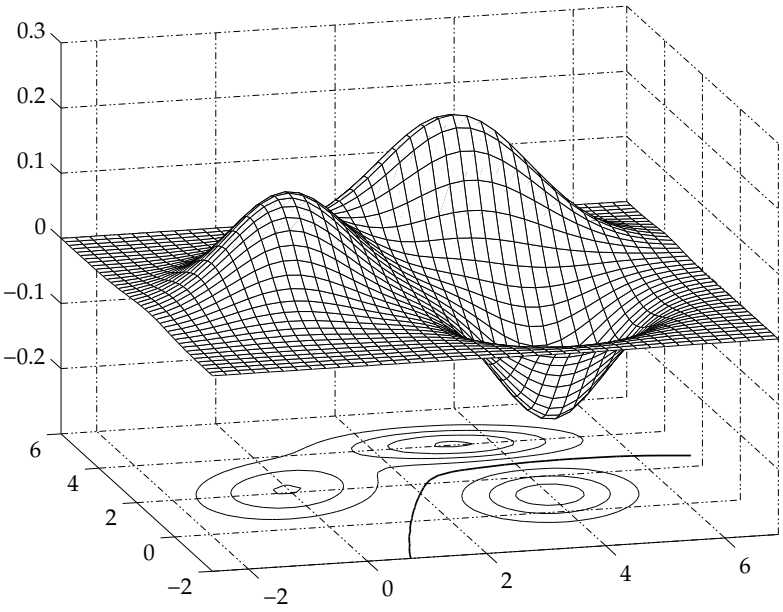

Figure 5.5: The weighted sum of the basis functions.

uct of the pattern with the weight vector is greater or equal to zero and as belonging to class C_- if the output is less than zero. Therefore all points lying on or above the plane formed by the first two coordinate axes are classified as belonging to class C_+, all points lying below this plane are classified as belonging to class C_-. Hence the contour line corresponding to the zero level can be seen as the decision boundary of this classifier. In Figure 5.5 this is the contour line which is drawn as a thick solid line. Figure 5.6 shows this decision boundary in the two-dimensional input space together with the original data points. It can be seen that this RBF network obtains perfect classification on the training set that is not linearly separable.

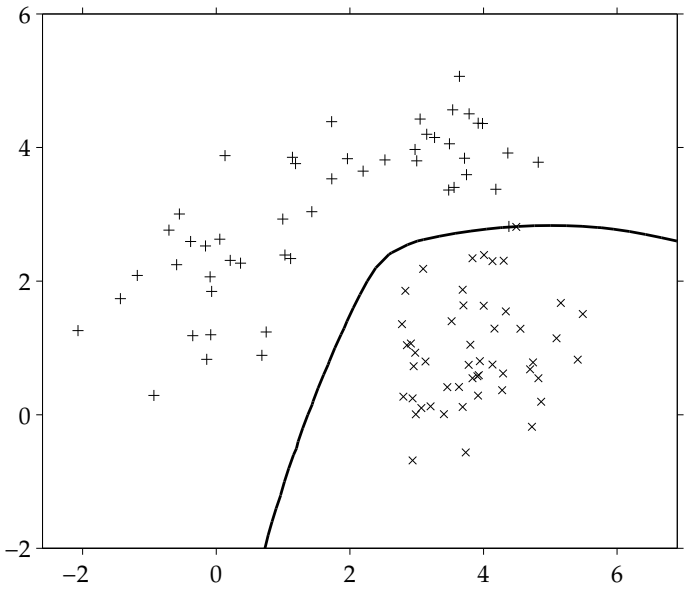

Figure 5.6: The projection surface in the input space.

5.2 SVMs using non-linear kernels

Another method to obtain a non-linear classification boundary is the use of non-linear kernels. The basic idea is the same as with RBF networks: project the data from the input space into a feature space and do a linear separation in the feature space. The hope is that by using non-linear projections and feature spaces of possibly much higher dimension than the input space, it becomes possible to achieve a good linear separation in the rich feature space. This usually corresponds to a non-linear separation in the input space.

The key ingredient of this approach is the kernel. Let $\mathbf{\Phi}$ be the mapping from the input space X into the feature space F. A kernel K is defined as

a function that takes two examples from the input space and calculates the inner product in the feature space:

$$\begin{aligned} K : X \times X &\rightarrow \mathbb{R} \\ (\mathbf{x}, \mathbf{z}) &\mapsto \langle \mathbf{\Phi}(\mathbf{x}), \mathbf{\Phi}(\mathbf{x}) \rangle. \end{aligned}$$

If we can reformulate our linear classifier in a way that the only places where examples appear are in inner products, we can replace all inner products by our kernel function. This is equivalent to first projecting the data into the feature space and then running the linear classifier in the feature space.

But, compared to the explicit use of the projected data, the method of replacing the inner products by the kernel has some advantages. Since the projected examples are never calculated explicitely, their size and the computational costs for the generation can be neglected as long as there exists an efficient way of computing the kernel.

Therefore, when using kernels, it is even possible to use feature spaces of potentially infinite dimension. One example of such a kernel is the string kernel[2] presented by Viswanathan and Smola ([36]), where the size (or at least the number of non-zero entries) of a feature vector grows with the length of the string.

Another kernel is the so called Gaussian kernel. It is defined as

$$K(\mathbf{x}, \mathbf{z}) = \exp\Big(- \frac{\|\mathbf{x} - \mathbf{z}\|_2^2}{2\sigma^2} \Big).$$

When using a 'linear' classifier where the implicit weight vector $\mathbf{\Phi}(\mathbf{w})$ is given as $\mathbf{\Phi}(\mathbf{w}) = \sum_{i=1}^{N} \alpha_i \mathbf{\Phi}(\mathbf{x}^i)$, the classification of a new example $\mathbf{x}$ is done as

$$\begin{aligned} \operatorname{sgn}\big(\mathbf{\Phi}(\mathbf{w})^\top \mathbf{\Phi}(\mathbf{x})\big) &= \operatorname{sgn}\Big(\sum_{i=1}^{N} \alpha_i K(\mathbf{x}^i, \mathbf{x}) \Big) \\ &= \operatorname{sgn}\Big(\sum_{i=1}^{N} \alpha_i \exp\Big(- \frac{\|\mathbf{x}^i - \mathbf{x}\|_2^2}{2\sigma^2} \Big) \Big). \end{aligned} \tag{5.4}$$

Looking at (5.1) and (5.2), we see that (5.4) is equivalent to an RBF network with weight vector $(\alpha_1, \ldots, \alpha_N)^\top$ and basis functions with Gaussians using the training vectors as centres and $\sigma_m = \sigma$ for all m.[3]

[2] A string kernel is a kernel that computes the inner product of strings mapped to a real valued vector space.

[3] Note that, when using a Gaussian kernel in a 'linear' classifier, the effective dimensionality is only bounded by the number of training vectors. When using an SVM however, the dimensionality will also be bounded by the number of support vectors. This is usually much smaller than the number of training vectors.

Chapter 6

The Gauss-Helmert method using RBFs

In the previous chapter we have shown how an RBF network can be set up and that the use of a Gaussian kernel in a 'linear' classifier can be equivalent to the use of an RBF network. Therefore we will use the RBF approach to extend our Gauss-Helmert based method to non-linear classification.

6.1 How it works

The idea can be seen as a direct descendent of the RBF network presented in the previous chapter.

First we choose a number M of basis functions and cluster the data into M clusters. For each cluster we calculate an estimate for the covariance matrix.

After this we project the data into $\mathbb{R}^M$ using the distributions of the points from the clusters as basis functions as described in the previous chapter.

Taking the variance of the points in the original space, we use error propagation (see Appendix A) to calculate an estimate for the variance in the projected space (see Appendix C).

Using the projected data together with the estimates of the variance, we can use the techniques presented in Chapter 4 to calculate the remaining parameters of a decision boundary together with a covariance matrix.

6.2 Experiments

A circle

This dataset (see Figure 6.1) consists of 200 points uniformly distributed in the 8×8 square centred around $(2,2)^\top$. All 120 points lying within the

circle with radius $8/\sqrt{2\pi}$ and centre $(2,2)^\top$ belong to class C_+, all 80 points outside this circle belong to class C_-.

We used a k-means clustering algorithm with $M = 16$ clusters. In order to get a smooth decision surface, we scaled the covariance matrices obtained for the clusters with the factor 15. The estimation of the parameters in the projected space has been done following the approach described in subsection 4.2.1. The circle shows the border between the two classes used for generating the points. When classifying the training set, only 2 errors (both false positives) occur.

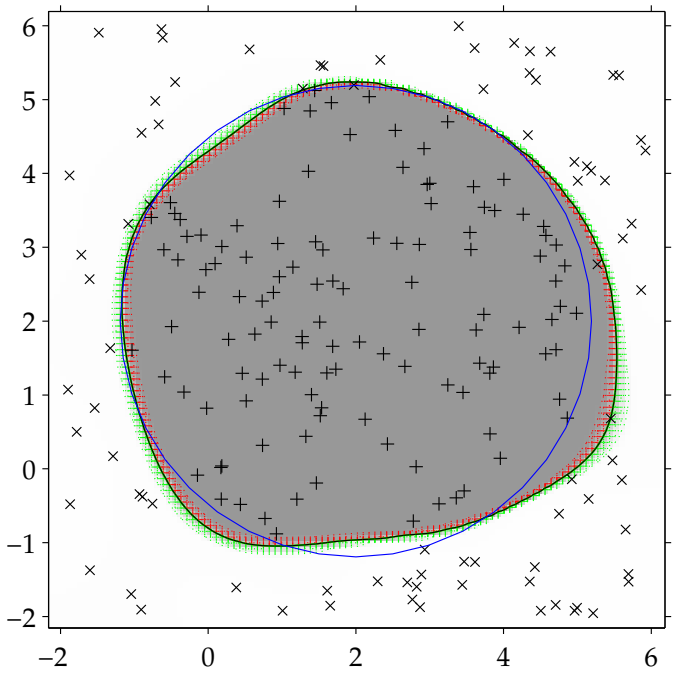

Figure 6.1: Points uniformly distributed within a square. Points within the circle belong to C_+, points outside the circle to C_-. The dotted lines connect points of equal confidence as e.g. in Figure 4.4.

Chessboards

Further experiments have been conducted using chessboard-like training sets. Figure 6.2 and Figure 6.3 show the results of learning two chessboards of different sizes.

The circles show the centres of the clusters learned by the k-means clustering algorithm. The number M of clusters used are $M = 20$ for the 2×2 chessboard and $M = 64$ for the large board. Both data sets are balanced

with 400 points for the small chessboard and 1600 points for the larger chessboard. There occur 8 errors when classifying the training set for the 2×2 chessboard.

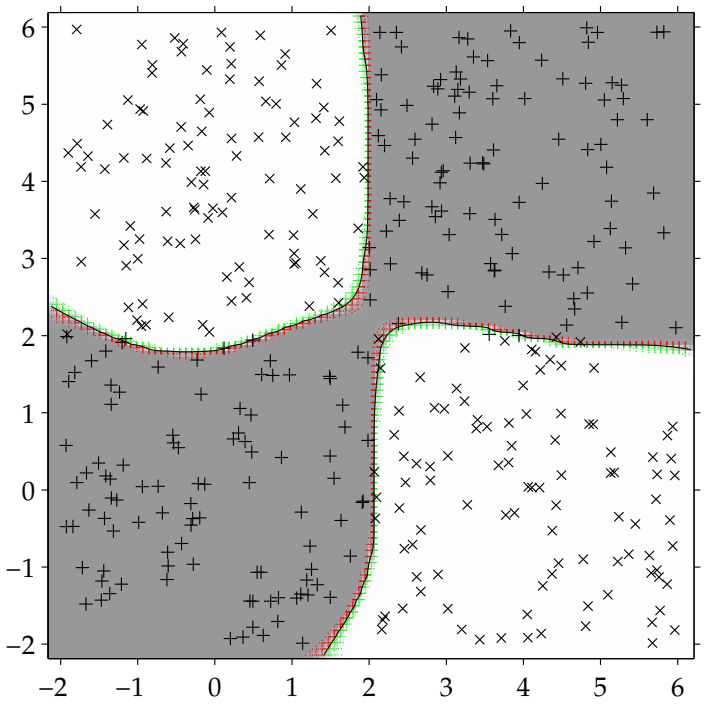

Figure 6.2: Chessboard of size 2×2.

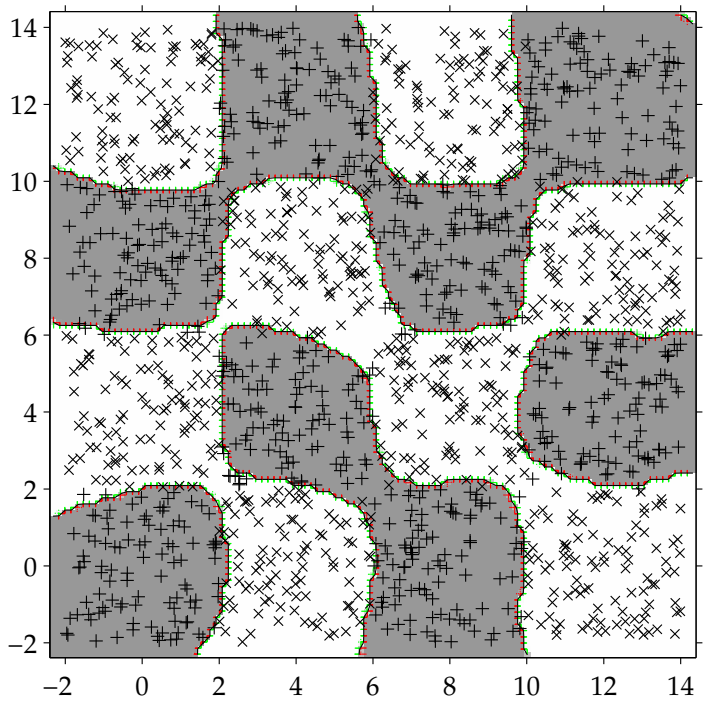

Figure 6.3: Chessboard of size 4×4.

Chapter 7

An idea inspired by statistical testing theory

While in subsection 4.1.1 we have shown a way to compute the probability of a decision, we will take a different route in this chapter. The approach is inspired from statistical testing theory where one states a hypothesis, observes some data, and then decides to reject the hypothesis or to not reject it.

For each classification we want to know whether it is made with confidence or if it is vague. The idea is that a point $\mathbf{x}$ lying on the separating hyperplane cannot be classified with confidence. Therefore we form the hypothesis that the point $\mathbf{x}$ is lying on the hyperplane. If we accept this hypothesis we judge the classification of this point as unreliable; if we reject the hypothesis we accept the classification as reliable.

7.1 The computations

To perform the test whether a point $\mathbf{x}$ is lying on the separating hyperplane or not, we follow the approach proposed by Heuel in [13] in Section 4.5. We calculate the distance of the point to the hyperplane as

$$\bar{d} = (\mathbf{n}^\top, b) \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix}.$$

Since we assume that we do not have any uncertainty information for the point $\mathbf{x}$, we use the zero matrix as covariance matrix $\mathbf{\Sigma}_{\mathbf{x},\mathbf{x}}$ for $\mathbf{x}$. Using this matrix, the variance for the distance $\bar{d}$ becomes

$$\begin{aligned} \mathbf{\Sigma}_{\bar{d},\bar{d}} &= (\mathbf{n}^\top, b)\mathbf{\Sigma}_{\mathbf{x},\mathbf{x}}(\mathbf{n}^\top, b)^\top + (\mathbf{x}^\top, 1, 0)\mathbf{\Sigma}_{\Delta\mathbf{w},\Delta\mathbf{w}}(\mathbf{x}^\top, 1, 0)^\top \\ &= (\mathbf{x}^\top, 1, 0)\mathbf{\Sigma}_{\Delta\mathbf{w},\Delta\mathbf{w}} \begin{pmatrix} \mathbf{x} \\ 1 \\ 0 \end{pmatrix}. \end{aligned}$$

Testing whether $\mathbf{x}$ is lying on the separating hyperplane, we form the hypothesis $H_0 : \bar{d} = 0$. The optimal test statistic for this hypothesis is given by

$$T = \bar{d}\,\boldsymbol{\Sigma}_{\bar{d},\bar{d}}^{-1}\bar{d} \sim \chi_1^2,$$

where χ_1^2 denotes the χ^2-distribution with one degree of freedom.

To decide whether we reject the hypothesis or not, we first have to choose a significance level $\hat{\alpha}$ that specifies the probability of rejecting a true hypothesis (type I error).

Using the $(1-\hat{\alpha})$-quantile of the χ_1^2 distribution, which we will write as $\chi^2_{1,(1-\hat{\alpha})}$, the probability $P(T > \chi^2_{1,(1-\hat{\alpha})}|H_0)$ will be $\hat{\alpha}$. Therefore, since we want to reject the hypothesis H_0 with a significance level of $\hat{\alpha}$, we have to reject the hypothesis H_0 when

$$T = \bar{d}\,\boldsymbol{\Sigma}_{\bar{d},\bar{d}}^{-1}\bar{d} > \chi^2_{1,(1-\hat{\alpha})}.$$

Figure 7.1 shows the results of a first experiment using this approach. The two curved solid lines show the borders between the regions of acceptance (between the two lines) and rejection (outside the two lines) of the hypothesis for which we will define a classification as unreliable or reliable.

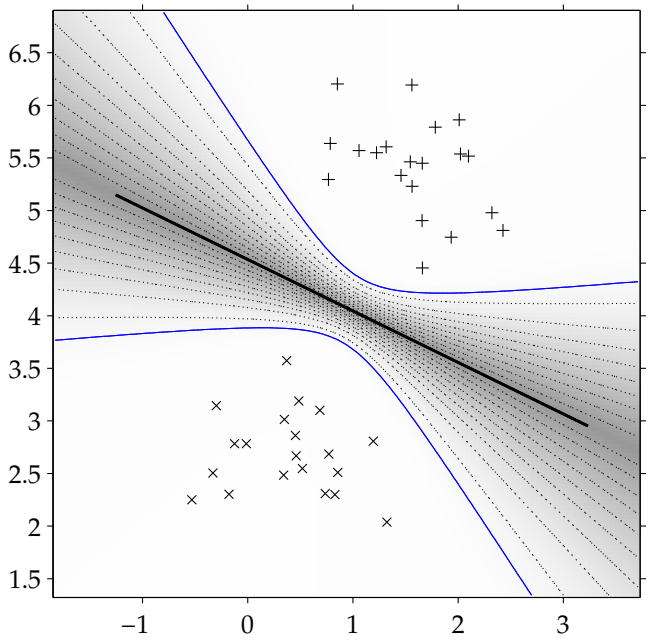

Figure 7.1: A dataset with two classes. Shown are the confidence (shading and dotted lines as in Figure 4.4) and the borders between 'reliable' and 'not so reliable' classification (curved solid lines). We used $\hat{\alpha} = 0.05$.

7.2 Some experiments with photos of coloured building bricks

To test our approach with some more realistic data, we have taken photos from coloured building bricks under different lighting conditions. These photos are shown in Figure 7.3. From these photos we have taken the RGB-values from single pixels (from the bricks, not from the background) as training points after adding a covariance matrix as a measure for our assumed uncertainty.

7.2.1 Calculating a covariance matrix

We assumed that the brightness varies a lot on the rough surface of the bricks. Therefore we assumed that in the direction of a point, when taken as a vector in RGB-space, the variance is twice as high as in the directions perpendicular to it.

We further assumed that a point with high saturation has a more reliable colour information than a point with low saturation, and that the colour information for a point with extremely high or low lightness is more unreliable than for a point with a medium lightness. Using a conversion from RGB (red green blue) to HLS (hue lightness saturation) as described in [9], we used

$$\big(1 - \text{saturation} * (1 - 2 * |\text{lightness} - 0.5|)\big)/8 \tag{7.1}$$

as variance in the direction of the greatest variance and the half of (7.1) for the directions perpendicular to it. Figure 7.2 visualises the covariance matrices for some points in a projection of the RGB-space to its first two coordinates.

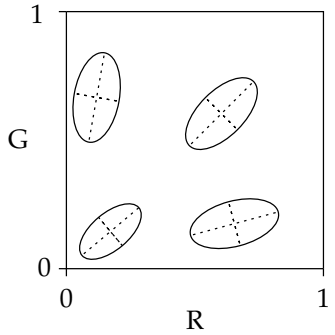

Figure 7.2: Visualisation of the covariances for points in the RGB-space (here only the R and G coordinates are shown). The dotted lines show the directions of the eigenvectors of the covariance matrices.

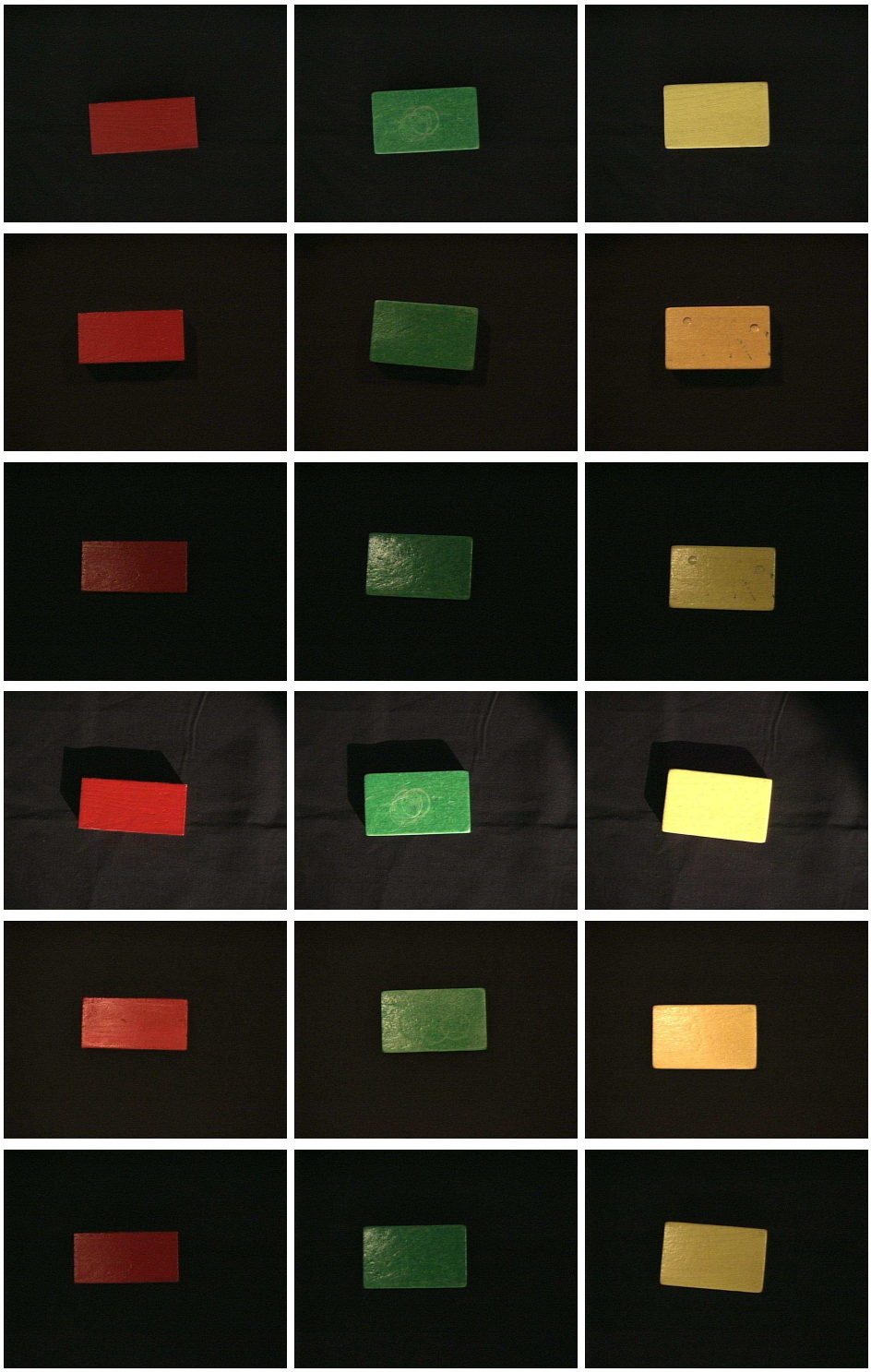

Figure 7.3: Pictures of coloured building bricks taken using a SONY CCD-IRIS/RGB colour video camera (model DXC-151AP). Pictures in the same line are taken with the same illumination. The order of the six different lighting conditions is the same as in Figure 7.4.

7.2.2 The experiments

In order to allow for a two-dimensional visualisation we have only taken the first two components of the RGB representation. Figure 7.4 shows the data taken from photos of red and green building bricks, and Figure 7.5 shows the data together with the variances calculated as described above. For each colour and illumination we have taken 122 points. To increase the readability of Figure 7.4 and Figure 7.5 we have only depicted every fourth point in these figures.

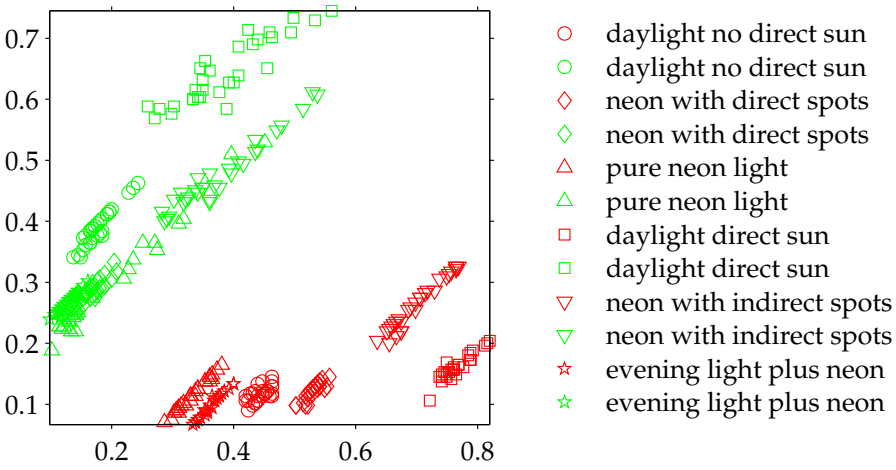

Figure 7.4: Red and green values (from an RGB representation) of points from images taken from red and green building bricks under different lighting conditions.

Training a linear Gauss-Helmert-classifier with the data from red and green bricks illuminated with pure neon light gives the separating hyperplane shown in Figure 7.6. The figure also shows the boundary between the regions of 'reliable' and 'unreliable' classification when using a parameter $\hat{\alpha} = 0.05$.

From Figure 7.7 it can be seen that the separating hyperplane learned from the red and green bricks illuminated with pure neon light is able to achieve perfect classification for all data from red and green bricks. For all red and most of the green bricks the classification will also be judged as 'reliable'.

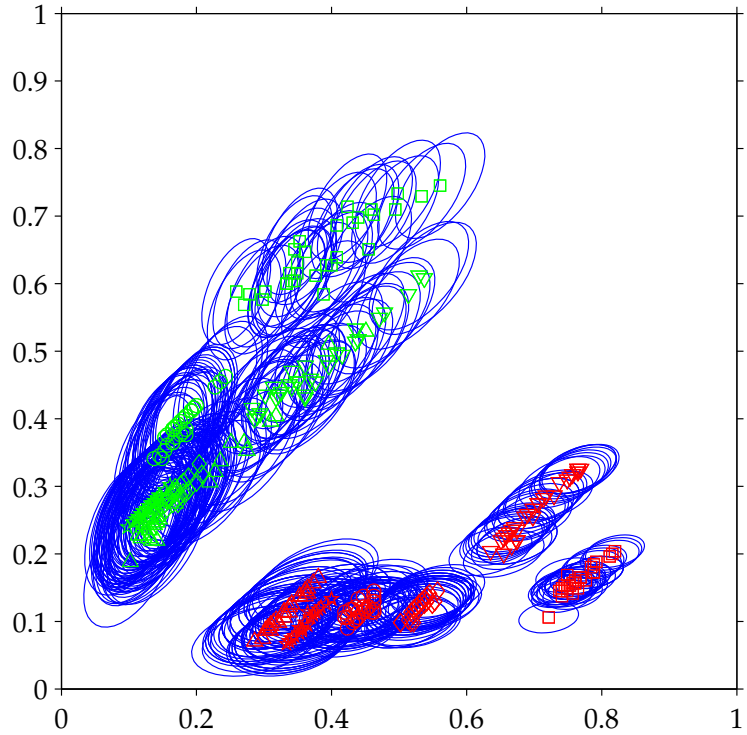

Figure 7.5: The points from Figure 7.4 together with their covariances visualised by ellipses.

When taking data from yellow building bricks, it can be seen that all of this data lies in the region of 'unreliable' classification. Since no data from yellow bricks has been used for training the classifier, data taken from yellow bricks should be entirely new to the classifier. Judging any classification results on this data as unreliable is therefore the best thing the classifier can do.

Changing the parameter $\hat{\alpha}$ results in a change of the boundary between the regions of 'reliable' and 'unreliable' classification. Figure 7.8 shows the different boundaries for different choices of $\hat{\alpha}$.

To show that the distributions of the noise on the points affects the result of the test as well, we have changed the covariance matrices by scaling the largest eigenvalue by 2 and the smallest by $1/2$ such that the largest eigenvalue is 8 times as big as the smallest (instead of 2 times as big as in the previous experiments). The results of training with the data from red and green bricks illuminated with pure neon light but with the changed covariance matrices is shown in Figure 7.9. While the resulting separating hyperplane is virtually identical to the one learned with the original covariance matrices, the boundaries between 'reliable' and 'not so reliable' are

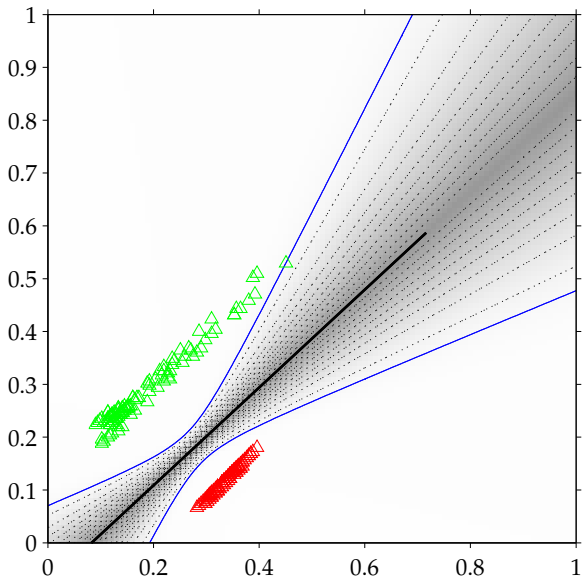

Figure 7.6: The results of training with data from photos of red and green bricks illuminated with pure neon light ($\hat{\alpha} = 0.05$).

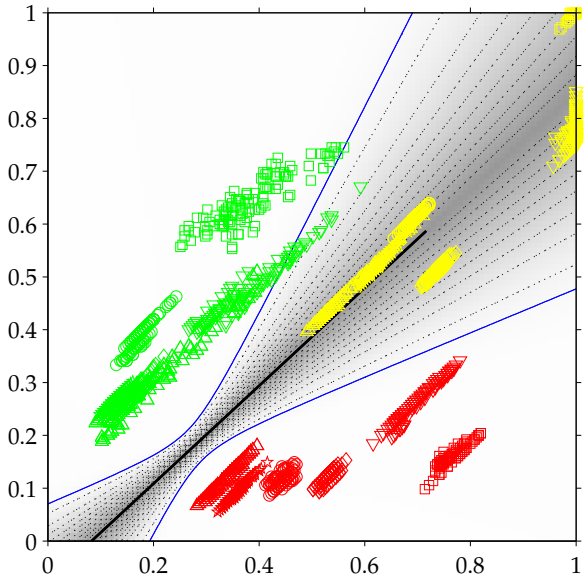

Figure 7.7: The results of training with data from photos of red and green bricks illuminated with pure neon light together with data from other photos of coloured bricks (red, green and yellow).

better for the modified covariance matrices in so far as the amount of data from green bricks lying in the region of 'unreliable' classification is reduced (compared to Figure 7.7). All of the data from yellow bricks is still lying in the region of 'unreliable' classification.

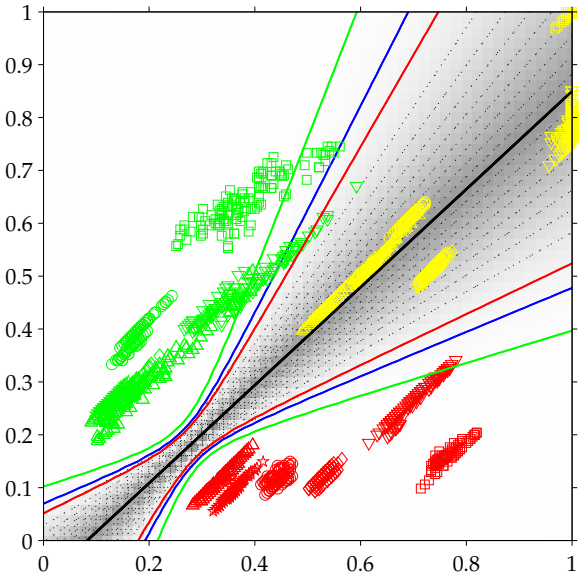

Figure 7.8: The results of training with data from photos of red and green bricks as in Figure 7.7 but with different values for $\hat{\alpha}$. Red lines: boundaries for $\hat{\alpha} = 0.1$, blue lines: $\hat{\alpha} = 0.05$ and green lines: $\hat{\alpha} = 0.01$.

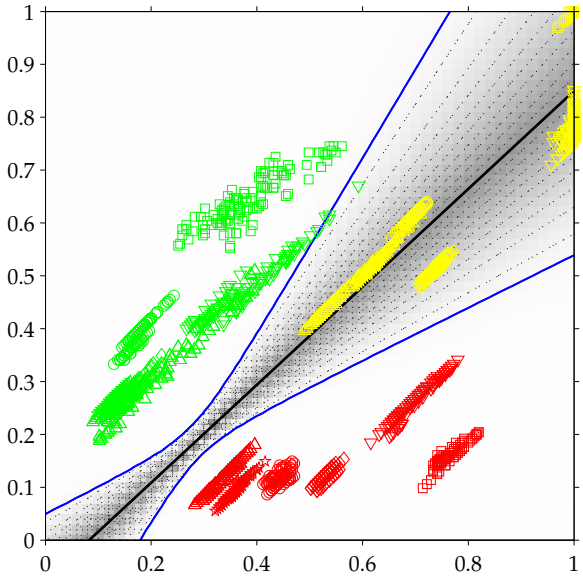

Figure 7.9: The results of training as in Figure 7.7 with $\hat{\alpha} = 0.05$ but with different covariance matrices for the points.

Chapter 8

Other ways to incorporate uncertainty

When looking for other classification algorithms that make use of uncertainty information on the training data for obtaining some measure of confidence on the decision boundary, we could not find any such algorithms. What we have found, however, are two algorithms that do not give a measure of confidence for the decision boundary but that are able to use uncertainty information on the training data in their learning process. These two algorithms have both been presented only recently in the proceedings of the 2004 annual conference on Neural Information Processing Systems ([31]).

Since they do not provide any measure of confidence for the classification or at least for the decision boundary, they are not directly comparable with our approach.

One problem with our experiments was that we could not find any 'real world' multi-dimensional data set to test our algorithms. The fact that there have evolved other algorithms that rely on data with similar properties made us hope to find some appropriate data sets.

Since the conference papers [2] and [1] reported experiments on artificial data sets only, we contacted the first authors by email. Although both authors replied to our request, the answers were not very encouraging: Jinbo Bi wrote that he did have some data from a medical application with uncertainty for each feature. However, this was confidential Siemens data. Chiranjib Bhattacharyya wrote: *'Actually we are also in the look out for real world data to test the proposed algorithms. [...] I would also request you to let me know if you come across such real life datasets. '*

Ralph Herbrich, who is working as a Researcher at Microsoft Research Cambridge, suggested to use data from a reviewing system (of e.g. conferences) where reviewers give a score together with their confidence on that score. The problem with this idea is that it would give only one-dimensional data (or at least only a confidence measure for one dimension).

8.1 Total support vector classification

In the proceedings of the 2004 conference on Neural Information Processing Systems Jinbo Bi and Thong Zhang ([2]) presented a way to perform support vector classification with the ability to consider uncertainty information on the training data. Their algorithm can be seen as an extension to the standard SVM learning algorithm.

Instead of assuming a Gaussian distribution of the noise on the training data, they use a simple bounded distribution of the uncertainty. If $\mathbf{x}_i$ is an observed pattern (with noise) and $\mathbf{x}'_i$ the original pattern (without noise), then the noise is given by $\Delta\mathbf{x}_i = \mathbf{x}'_i - \mathbf{x}_i$. The assumed bounded distribution is now given by $\|\Delta\mathbf{x}_i\| \leq \delta_i$. The SVM optimisation problem (2.9) is reformulated as

$$\begin{array}{ll} \text{minimise} & \|\mathbf{w}\|_2 \\ \text{subject to} & y_i(\mathbf{w}^\top(\mathbf{x}_i + \Delta\mathbf{x}_i) + b) \geq 1, \\ & \|\Delta\mathbf{x}_i\| \leq \delta_i, \\ & i = 1, \ldots, N. \end{array} \tag{8.1}$$

This can be interpreted as allowing the $\mathbf{x}_i$ to be moved up to δ_i in any direction.

If the original data set is linearly separable and all δ_i have the same value δ, we will get exactly the same hyperplane as without this modification. The only difference is that, due to the possible 'movement' of the $\mathbf{x}_i$, the geometrical margin will grow by δ.

If linear separation is not possible but there exists a hyperplane such that for each misclassified $\mathbf{x}_i$ the absolute value of the geometric margin γ_i is not greater than δ_i, then the vanilla total support vector machine (8.1) will find a solution.

The hope is that outliers will have a large δ_i such that the vanilla total SVM will still find a solution. The soft margin total SVM (also presented in [2]) will probably take advantage from this too. They also present a way to kernelise their total SVM.

8.2 Classification with gaussian uncertainty

Also in the proceedings of the 2004 conference on Neural Information Processing Systems Chiranjib Bhattacharyya, Pannagadatta K. Shivaswamy and Alex Smola ([1]) presented another approach for the training of a linear classifier considering uncertainty on the training data. Here the motivation is the use of incomplete data. While there might exist methods to estimate values for missing variables, the uncertainty on these 'guessed' values is assumed to be much greater than on the observed values. The authors of this paper assume the noise to be normal distributed.

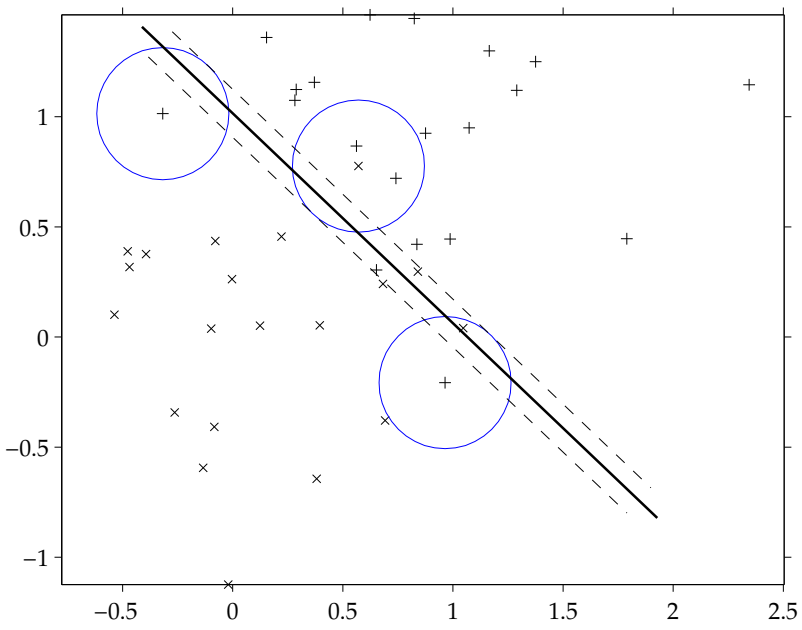

Figure 8.1: An example for total support vector classification. The third data set classified with a total SVM. The circles illustrate the possible variation of the three support vectors. It can be seen that the dashed parallels that show the margins are tangents to the circles of largest variation on the support vectors.

The idea is to reformulate the optimisation problem of the soft margin SVM

$$\begin{array}{ll} \text{minimise} & \frac{1}{2}\|\mathbf{w}\|_2 + C\sum_{i=1}^{N}\xi_i \\ \text{subject to} & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \\ & i = 1, \ldots, N \end{array}$$

as

$$\begin{array}{ll} \text{minimise} & \sum_{i=1}^{N}\xi_i \\ \text{subject to} & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \\ & \|\mathbf{w}\| \leq W, \\ & i = 1, \ldots, N \end{array} \tag{8.2}$$

with a user defined constant W.

Assuming that the $\mathbf{x}_i$ are not given exactly but just as the mean of a known distribution, the constraints of the reformulated problem are now

replaced by

$$\begin{aligned} \text{subject to} \quad & P\big(y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i\big) \geq \kappa_i, \\ & \xi_i \geq 0, \\ & \|\mathbf{w}\| \leq W, \\ & i = 1, \ldots, N. \end{aligned}$$

This gives rise to a set of new parameters κ_i. Choosing $\kappa_i = 1/2$ for $i = 1, \ldots, N$ should be equal to using (8.2) directly.

Using values greater than $1/2$ for the κ_i can be interpreted as using a classifier that tries to find a hyperplane with functional margin greater than 1. For points with great variance, the functional margin will tend to become greater than for points with small variance.

Using values smaller than $1/2$ for the κ_i will have the opposite effect: functional margins less than 1 become 'sufficient' and points with great variance will tend to get a smaller functional margin than points with small variance.

Note that although the authors allow values from $(0, 1]$ for the κ_is, the use of a value of 1 for a κ_i does not make sense when the $\mathbf{x}_i$ are normal distributed. The reason for this is that we would need an infinite distance of the hyperplane to $\mathbf{x}_i$ or an infinite value for ξ_i to fulfil the constraints. Both would make the optimisation impossible.

Chapter 9

Conclusions

In this text we have presented a novel architecture for linear and non-linear discrimination. Compared to most other such algorithms, it has the advantage that it does not only classify new examples as belonging to one of two classes. It is also able to deliver a measure of confidence for such decisions. This enables the user to detect unreliable classificated patterns and to forward them to a more sophisticated classifier or a human expert. The second advantage is its ability to make use of prior knowledge in the form of uncertainty information on the training data during the training process.

We have constructed this architecture by combining the ideas from Fisher's linear discriminant (and least squares) and the Gauss-Helmert model. The pure classification performance is therefore usually comparable with if not better than Fisher's linear discriminant. Other more modern architectures like the support vector machine sometimes outperform our approach, especially when the data set does not fit to our model. To deal with these cases, we have presented a method to overcome this difficulty in section 4.2.1. It works by using a superior classifier or some prior knowledge to get an initial estimate for the parameters and then uses a slightly modified version of our architecture that will finish with a parameter vector that is close to the initial estimate.

Besides this linear version we have also presented an extension able to perform non-linear classification in Chapter 6. This extension is based on the concepts of RBF networks described in Chapter 5.

The confidence values obtained when using our architecture give the user a hint to accept or not to accept a classification result. In Chapter 7 we have used ideas from statistical testing theory to make the decision of accepting or rejecting a classification result based on a parameter $\hat{\alpha}$, which gives the probability that the true hypothesis is not accepted. This frees the user from having to deal with purely heuristic confidence values.

As far as we know, this is the first architecture that is able to use uncertainty information on the training data and delivers a confidence value for its classification results. Therefore we have not been able to compare the performance of our architectures with similar architectures. The exper-

iments reported in this text are therefore mainly intended to proof that the concept works in principle. To conduct more sophisticated experiments we would have needed appropriate data from real applications which seems to be extremely rare and not available so far (see introduction of the previous chapter).

Appendix A

Error propagation

If the arguments of a function $g : X \rightarrow \mathbb{R}$ are uncertain, the result will be uncertain too. Error propagation is a means for estimating the noise of the function output when the distribution of the noise on the input data and the first derivatives of the function are known.

In this text we assume that the noise on the input is normal distributed and that the mean $\bar{\mathbf{x}}$ and the (co)variance $\mathbf{\Sigma}_{X,X}$ of the probability density function $f_{X,X}$ of the noisy input are known.

Using a second-order Taylor series expansion around the mean, we get

$$g(\mathbf{x}) \approx g(\bar{\mathbf{x}}) + \sum_i (x_i - \bar{x}_i)\partial_i g(\bar{\mathbf{x}}) + \frac{1}{2}\sum_{i,j}(x_i - \bar{x}_i)(x_j - \bar{x}_j)\partial_i\partial_j g(\bar{\mathbf{x}}).$$

The expectation $\mathcal{E}[g(X)]$ now evaluates to

$$\begin{aligned}
\mathcal{E}[g(X)] &= \int_X g(\mathbf{x}) f_X(\mathbf{x}) d\mathbf{x} \\
&\approx g(\bar{\mathbf{x}}) + \sum_i \partial_i g(\bar{\mathbf{x}}) \int_{X_i} (x_i - \bar{x}_i) f_{X_i}(x_i) dx_i \\
&\quad + \frac{1}{2}\sum_{i,j} \partial_i\partial_j g(\bar{\mathbf{x}}) \int_{X_i}\int_{X_j} (x_i - \bar{x}_i)(x_j - \bar{x}_j) f_{X_i,X_j}(x_i, x_j) dx_i dx_j \\
&= g(\bar{\mathbf{x}}) + \frac{1}{2}\sum_{i,j} \partial_i\partial_j g(\bar{\mathbf{x}}) \mathbf{\Sigma}_{X_i,X_j}
\end{aligned}$$

since $\int_{X_i} (x_i - \bar{x}_i) f_{X_i}(x_i) dx_i = 0$.

Doing the same for g^2 and calculating the expectation $\mathcal{E}[g^2(X)]$ for the approximation, we can calculate the variance as

$$\mathcal{C}[g(X)] = \mathcal{E}[g^2(X)] - \mathcal{E}^2[g(X)].$$

Appendix B

Calculating the Φ-function in Matlab

From the Matlab documentation ([24]) and from mathworld ([37]):
The Matlab error function is given as

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2) dt,$$

that is we can express the cumulative distribution function for the normal distribution

$$D(x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{-(x-\mu)^2/(2\sigma^2)} dx$$

as

$$D(x) = \frac{1}{2}\left(1 + \texttt{erf}\left(\frac{x-\mu}{\sqrt{2\sigma^2}}\right)\right).$$

In our case we want the probability that the distance of a point to the hyperplane is greater than 0. I.e. we have μ = distance, σ^2 = 'covariance' and we have to evaluate the integral from 0 to ∞ which is the same as from $-\infty$ to $2 * \mu$. Therefore we have to calculate

$$D(2\mu) = \frac{1}{2}(1 + \texttt{erf}(\mu/\sqrt{\sigma^2 * 2})).$$

Appendix C

Error propagation through a multivariate normal distribution

First we use a Taylor expansion to calculate an approximation of the Gaussian g and its expectation $\mathcal{E}[g(X)]$:

$$
\begin{aligned}
g(\mathbf{x}) &= \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^\top \Sigma^{-1}(\mathbf{x}-\mu)\right) \\
&\approx g(\bar{\mathbf{x}}) + \sum_i (x_i - \bar{x}_i)\partial_i g(\bar{\mathbf{x}}) + \frac{1}{2}\sum_{i,j}(x_i-\bar{x}_i)(x_j-\bar{x}_j)\partial_i\partial_j g(\bar{\mathbf{x}}) \\
&= g(\bar{\mathbf{x}}) + \sum_i (x_i - \bar{x}_i) g(\bar{\mathbf{x}})(-\frac{1}{2})\sum_l 2\Sigma^{-1}_{i,l}(\bar{x}_l - \mu_l) \\
&\quad + \frac{1}{2}\sum_{i,j}(x_i-\bar{x}_i)(x_j-\bar{x}_j) \\
&\quad \Big(g(\bar{\mathbf{x}})(-\frac{1}{2})\sum_l 2\Sigma^{-1}_{j,l}(\bar{x}_l-\mu_l)(-\frac{1}{2})\sum_l 2\Sigma^{-1}_{i,l}(\bar{x}_l-\mu_l) \\
&\quad + g(\bar{\mathbf{x}})(-\Sigma^{-1}_{i,j})\Big) \\
&= g(\bar{\mathbf{x}}) - \sum_i (x_i-\bar{x}_i) g(\bar{\mathbf{x}}) \underbrace{\sum_l \Sigma^{-1}_{i,l}(\bar{x}_l-\mu_l)}_{e_i^\top \Sigma^{-1}(\bar{\mathbf{x}}-\mu)} \\
&\quad + \frac{1}{2}\sum_{i,j}(x_i-\bar{x}_i)(x_j-\bar{x}_j) g(\bar{\mathbf{x}}) \\
&\quad \Big(\underbrace{\sum_l \Sigma^{-1}_{j,l}(\bar{x}_l-\mu_l)\sum_l \Sigma^{-1}_{i,l}(\bar{x}_l-\mu_l)}_{(\bar{\mathbf{x}}-\mu)^\top\Sigma^{-1}e_j e_i^\top \Sigma^{-1}(\bar{\mathbf{x}}-\mu)} - \Sigma^{-1}_{i,j}\Big)
\end{aligned}
$$

$$
\begin{aligned}
&= g(\bar{\mathbf{x}}) - \sum_i (x_i - \bar{x}_i) g(\bar{\mathbf{x}}) e_i^\top \Sigma^{-1} (\bar{\mathbf{x}} - \mu) \\
&\quad + \frac{1}{2} \sum_{i,j} (x_i - \bar{x}_i)(x_j - \bar{x}_j) g(\bar{\mathbf{x}}) \\
&\quad \Big((\bar{\mathbf{x}} - \mu)^\top \Sigma^{-1} e_j e_i^\top \Sigma^{-1} (\bar{\mathbf{x}} - \mu) - \Sigma_{i,j}^{-1} \Big),
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{E}[g(X)] &\approx g(\bar{\mathbf{x}}) + \sum_i \partial_i g(\bar{\mathbf{x}}) \underbrace{\int_{\mathbb{D}_{X_i}} (x_i - \bar{x}_i) f_{X_i}(x_i)\, dx_i}_{0} \\
&\quad + \frac{1}{2} \sum_{i,j} \partial_i \partial_j g(\bar{\mathbf{x}}) \int_{\mathbb{D}_{X_i}} \int_{\mathbb{D}_{X_j}} (x_i - \bar{x}_i)(x_j - \bar{x}_j) f_{X_i, X_j}(x_i, x_j))\, dx_i\, dx_j \\
&= g(\bar{\mathbf{x}}) + \frac{1}{2} \sum_{i,j} g(\bar{\mathbf{x}}) \Big((\bar{\mathbf{x}} - \mu)^\top \Sigma^{-1} e_j e_i^\top \Sigma^{-1} (\bar{\mathbf{x}} - \mu) - \Sigma_{i,j}^{-1} \Big) (\Sigma_{X,X})_{i,j}.
\end{aligned}
$$

Then we do the same for $g^2(\mathbf{x})$ and $\mathcal{E}[g^2(X)]$:

$$
\begin{aligned}
g^2(\mathbf{x}) &= \frac{1}{(2\pi)^d |\Sigma|} \exp\Big(-(\mathbf{x} - \mu)^\top \Sigma^{-1} (\mathbf{x} - \mu) \Big) \\
&\approx g^2(\bar{\mathbf{x}}) + \sum_i (x_i - \bar{x}_i) 2 g^2(\bar{\mathbf{x}}) (-\frac{1}{2}) \sum_l 2 \Sigma_{i,l}^{-1} (\bar{x}_l - \mu_l) \\
&\quad + \frac{1}{2} \sum_{i,j} (x_i - \bar{x}_i)(x_j - \bar{x}_j) \\
&\quad \Big(4 g^2(\bar{\mathbf{x}}) (-\frac{1}{2}) \sum_l 2 \Sigma_{j,l}^{-1} (\bar{x}_l - \mu_l) (-\frac{1}{2}) \sum_l 2 \Sigma_{i,l}^{-1} (\bar{x}_l - \mu_l) \\
&\quad + 2 g^2(\bar{\mathbf{x}}) (-\Sigma_{i,j}^{-1}) \Big) \\
&= g^2(\bar{\mathbf{x}}) - \sum_i (x_i - \bar{x}_i) 2 g^2(\bar{\mathbf{x}}) \sum_l \Sigma_{i,l}^{-1} (\bar{x}_l - \mu_l) \\
&\quad + \sum_{i,j} (x_i - \bar{x}_i)(x_j - \bar{x}_j) g^2(\bar{\mathbf{x}}) \\
&\quad \Big(2 \sum_l \Sigma_{j,l}^{-1} (\bar{x}_l - \mu_l) \sum_l \Sigma_{i,l}^{-1} (\bar{x}_l - \mu_l) - \Sigma_{i,j}^{-1} \Big) \\
&= g^2(\bar{\mathbf{x}}) - \sum_i (x_i - \bar{x}_i) 2 g^2(\bar{\mathbf{x}}) e_i^\top \Sigma^{-1} (\bar{\mathbf{x}} - \mu) \\
&\quad + \sum_{i,j} (x_i - \bar{x}_i)(x_j - \bar{x}_j) g^2(\bar{\mathbf{x}}) \\
&\quad \Big(2 (\bar{\mathbf{x}} - \mu)^\top \Sigma^{-1} e_j e_i^\top \Sigma^{-1} (\bar{\mathbf{x}} - \mu) - \Sigma_{i,j}^{-1} \Big),
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{E}[g^2(X)] &\approx g^2(\bar{\mathbf{x}}) - 0 \\
&\quad + \sum_{i,j} g^2(\bar{\mathbf{x}}) \Big(2 (\bar{\mathbf{x}} - \mu)^\top \Sigma^{-1} e_j e_i^\top \Sigma^{-1} (\bar{\mathbf{x}} - \mu) - \Sigma_{i,j}^{-1} \Big) (\Sigma_{X,X})_{i,j}.
\end{aligned}
$$

And finally we use these approximations to calculate an estimate for the variance $\mathcal{C}[g(X)]$:

$$\begin{aligned}
\mathcal{C}[g(X)] &= \mathcal{E}[g^2(X)] - \mathcal{E}^2[g(X)] \\
&\approx g^2(\bar{\mathbf{x}}) - 0 \\
&\quad + \sum_{i,j} g^2(\bar{\mathbf{x}}) \Big(2(\bar{\mathbf{x}} - \mu)^\top \Sigma^{-1} e_j e_i \Sigma^{-1} (\bar{\mathbf{x}} - \mu) - \Sigma_{i,j}^{-1} \Big) (\Sigma_{X,X})_{i,j} \\
&\quad - \Big(g(\bar{\mathbf{x}}) + \frac{1}{2} \sum_{i,j} g(\bar{\mathbf{x}}) \Big((\bar{\mathbf{x}} - \mu)^\top \Sigma^{-1} e_j e_i^\top \Sigma^{-1} (\bar{\mathbf{x}} - \mu) - \Sigma_{i,j}^{-1} \Big) (\Sigma_{X,X})_{i,j} \Big)^2 \\
&= 0 + \sum_{i,j} g^2(\bar{\mathbf{x}}) \Big((\bar{\mathbf{x}} - \mu)^\top \Sigma^{-1} e_j e_i \Sigma^{-1} (\bar{\mathbf{x}} - \mu) - \Sigma_{i,j}^{-1} \Big) (\Sigma_{X,X})_{i,j} \\
&\quad - \frac{1}{4} \Big(\sum_{i,j} g(\bar{\mathbf{x}}) \Big((\bar{\mathbf{x}} - \mu)^\top \Sigma^{-1} e_j e_i^\top \Sigma^{-1} (\bar{\mathbf{x}} - \mu) - \Sigma_{i,j}^{-1} \Big) (\Sigma_{X,X})_{i,j} \Big)^2 \\
&= -\frac{1}{4} \Big(g(\bar{\mathbf{x}}) \Big((\bar{\mathbf{x}} - \mu)^\top \Sigma^{-1} \Sigma_{X,X} \Sigma^{-1} (\bar{\mathbf{x}} - \mu) - \langle \Sigma^{-1}, \Sigma_{X,X} \rangle_{\text{FRO}} \Big) \Big)^2,
\end{aligned}$$

where $\langle \cdot, \cdot \rangle_{\text{FRO}}$ denotes the Frobenius product[1] .

[1]The Frobenius product of two matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}$ is defined as

$$\langle \mathbf{A}, \mathbf{B} \rangle_{\text{FRO}} = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{i,j} b_{i,j}.$$

References

[1] Chiranjib Bhattacharyya, Pannagadatta K. Shivaswamy, and Alex Smola. A second order cone programming formulation for classifying missing data. In Saul et al. [31], pages 153–160.

[2] Jinbo Bi and Tong Zhang. Support vector classification with input data uncertainty. In Saul et al. [31], pages 161–168.

[3] Chistopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995. Reprinted 2000.

[4] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM Press, 1992.

[5] Ilja N. Bronštein, Konstantin A. Semendjajev, Gerhard Musiol, and Heiner Mühlig. *Taschenbuch der Mathematik*. Verlag Harri Deutsch, 4th edition, 1999.

[6] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

[7] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

[8] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley-Interscience, New York, 2nd edition, 2001.

[9] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Huges. *Computer Graphics: principles and practice*. Addison-Wesley Systems Programming Series. Addison-Wesley, 2nd edition, July 1996. Edition in C.

[10] Wolfgang Förstner. Uncertainty and projective geometry. In Eduardo Bayro Corrochano, editor, *Handbook of Geometric Computing – Applications in Pattern Recognition, Computer Vision, Neuralcomputing, and Robotics*, pages 493–534. Springer-Verlag, Heidelberg, 2005.

[11] Jerome H. Friedman and John W. Tukey. A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers*, C-23(9):881–890, Semptember 1974.

[12] Keinosuke Fukunaga. *Introduction to statistical Pattern Recognition*. Computer Science and Scientific Computing. Academic Press, San Diego, 2nd edition, 1990.

[13] Stephan Heuel. *Statistical Reasoning in Uncertain Projective Geometry for Polyhedral Object Reconstruction*. PhD thesis, Rheinische Friedrich-Wilhelms-Universität zu Bonn, October 2002.

[14] Peter J. Huber. Projection pursuit. *The Annals of Statistics*, 13(2):435–475, 1985.

[15] Gerhard Hübner. *Stochastik – Eine anwendungsorientierte Einführung für Informatiker, Ingenieure und Mathematiker*. Vieweg, Braunschweig/Wiesbaden, 1996.

[16] Karl-Rudolf Koch. *Parameter Estimation and Hypothesis Testing in Linear Models*. Springer-Verlag, Berlin Heidelberg, 2nd edition, 1999. Translation of the 3rd German Edition „Parameterschätzung und Hypothesentests in linearen Modellen“, Ferd. Dümmlers Verlag, 1997.

[17] Wolfgang König, Heinrich Rommelfanger, Dietich Ohse, Markus Hofmann, Klaus Schäfer, Helmut Kuhnle, and Andreas Pfeifer, editors. *Taschenbuch der Wirtschaftsinformatik und Wirtschaftsmathematik*. Verlag Harri Deutsch, Frankfurt am Main, Thun, 1st edition, 1999.

[18] Peter Kosmol. *Methoden zur numerischen Behandlung nichtlinearer Gleichungen und Optimierungsaufgaben*. Teubner Studienbücher. Teubner, Stuttgart, 2nd edition, 1989.

[19] Peter Kosmol. *Optimierung und Approximation*. De-Gruyter-Lehrbuch. Walter de Gruyter & Co., Berlin, 1991.

[20] Ulrich Krengel. *Einführung in die Wahrscheinlichkeitstheorie und Statistik*. Vieweg Studium: Aufbaukurs Mathematik. Vieweg, Wiesbaden, 7th edition, 2003.

[21] Erwin Kreyszig. *Statistische Methoden und ihre Anwendungen*. Vandenhoeck & Ruprecht, Göttingen, 7th edition, 1979. 5th reprint 1998.

[22] Sun Yuan Kung. *Digital Neural Networks*. Prentice-Hall information and system sciences series. PTR Prentice-Hall, Inc., Anglewood Cliffs, New Jersey, 1993.

[23] Yaoyong Li, Hugo Zaragoza, Ralf Herbrich, John Shawe-Taylor, and Jaz Kandola. The perceptron algorithm with uneven margins. In Sammut and Hoffmann [30], pages 379–386.

[24] The MathWorks, Inc. *MATLAB online documentation*, 2005.

[25] Edward M. Mikhail. *Observations and least squares*. University Press of America, Lanham, 1976.

[26] Christian Perwass. Error propagation in Clifford algebra. Obtained directly from the author at CAU Kiel, October 2004.

[27] Christian Perwass and Christian Gebken. The Gauss-Helmert model applied to elements of conformal GA. Obtained directly from the first author at CAU Kiel, October 2004.

[28] Camillo Ressl. *Geometry, Constraints and Computation of the Trifocal Tensor*. PhD thesis, Technische Universität Wien, June 2003.

[29] Horst Rinne. *Taschenbuch der Statistik*. Wissenschaftlicher Verlag Harri Deutsch GmbH, Frankfurt am Main, 3rd edition, 2003.

[30] Claude Sammut and Achim Hoffmann, editors. *Proceedings of the Nineteenth International Conference on Machine Learning, 8–12 July 2002, University of New South Wales, Sydney, Australia*, volume 19, San Francisco, California, 2002. Morgan Kaufmann Publishers.

[31] Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors. *Advances in Neural Information Processing Systems 17*, Cambridge, MA, 2005. MIT Press.

[32] Bernhard Schölkopf and Alex Smola. *Learning with Kernels*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, 2002.

[33] Gerald Sommer. Skript zur Vorlesung *Einführung in die Neuroinformatik*. Christian-Albrechts-Universität zu Kiel, Sommersemester 2005.

[34] H. L. Vacher. Coputational geology 16 – the Taylor series and error propagation. *Journal of Geoscience Education*, 49(3):305–313, May 2001. (edits, June 2005).

[35] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Adaptive and Learning Systems for Signal Processing, Communications, and Control. Springer, 1995.

[36] S.V.N. Viswanathan and Alexander J. Smola. Fast kernels for string and tree matching. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 569–576. MIT Press, Cambridge, MA, 2003.

[37] Eric W. Weisstein. *Erf*. http://mathworld.wolfram.com/Erf.html, 2005.

[38] Gabriele Widmann. *Künstliche Neuronale Netze und ihre Beziehungen zur Statistik*, volume 2739 of *Reihe V: Volks- und Betriebswirschaft*. Peter Lang: Europäischer Verlag der Wissenschaften, Frankfurt am Main, 2001.

Notation

$\|\cdot\|_2$	Euclidean norm
$\langle\cdot,\cdot\rangle_{\mathrm{FRO}}$	Frobenius product
$\langle\mathbf{a},\mathbf{b}\rangle = \mathbf{a}^\top\mathbf{b}$	inner product between column vectors $\mathbf{a}$ and $\mathbf{b}$
$\mathbf{x}^\top, \mathbf{X}^\top$	transpose of vector, matrix
$\|\alpha\|$	distance of the outer parallels to the central parallel that is the discriminating hyperplane
α_i	expansion coefficient
$\hat{\alpha}$	probability of making a type I error
b	parameter specifying the distance of the line/hyperplane to the origin
γ	margin
c	number of classes (if $c = 2$ then $C_+ = C_1$ and $C_- = C_2$)
d	dimension of the input space
e_i	ith basis vector, i.e. ith component is 1, all other components are 0
F	feature space
$\mathbf{I}$	unity matrix
$J(\mathbf{w})$	Fisher criterion
μ	mean of distribution
M	number of basis functions
N	size of the training set
$N_1 = N_+, N_2 = N_-$	number of training vectors with label $+1, -1$
$P(\cdot)$	probability
$p(\cdot)$	probability density
$\mathbb{R}$	real numbers
$\mathbb{R}_{>0}, \mathbb{R}_{\geq 0}$	positive, non-negative real numbers
σ^2	variance of distribution
$S \subset X \times Y$	finite training set
$\mathrm{sgn}(\cdot)$	signum function ($\mathrm{sgn}(x) = 1$ if $x \geq 0$, -1 otherwise)
τ	margin for the perceptron algorithm with margins
$\mathbf{w}$	weight vector
ξ_i	slack variable
X	input space
$\mathbf{x}^i$	the ith training vector
Y	output space (for binary classification: $Y = \{-1, +1\}$)
y_i	target label for the ith training vector

Acknowledgements

This thesis was written under the supervision of Dr. Christian Perwass at the Christian-Albrechts-University of Kiel at the Chair of Cognitive Systems led by Prof. Dr. Gerald Sommer. Therefore I would like to thank Christian for all the time he spent with me discussing possible solutions for shortcomings of our architecture.

I would also like to thank Professor Sommer for providing me with a comfortable working environment not only consisting of a desk and a computer but of many nice colleagues as well. My special thanks go to Christian Gebken for sharing his office with me.

For the support from home I would like to thank my parents, and for all her patience, proofreading and much more I am also indebted to Susanne.

Hiermit versichere ich, nur die angegebenen Quellen und Hilfsmittel benutzt zu haben.